LF-LoRA: A Memory-Efficient Fused-Backward Backend for LoRA-Adapted Transformers

Anonymous ACL submission

Abstract

We present LF-LoRA, a checkable, Triton-free backward backend for LoRA-adapted transformers. For each $\mathrm{Act} \circ \mathrm{Norm} \circ \mathrm{Linear}$ sub-block under per-sample RMSNorm/LayerNorm and SwiGLU gating, we factor the Jacobian transpose into four forward-cached terms and run them inside three custom `autograd.Functions` per sub-block, replacing the per-op nodes (and the intermediate activations they save) that the standard backward traverses. The result matches BP-LoRA's gradient at median cosine ≥ 0.999 and tracks its downstream accuracy within seed-σ on six benchmarks (MMLU, GSM8K, AlpacaEval-2 LC, TruthfulQA, MT-Bench, full-suite IFEval) across LLaMA-7B and Llama-3.1-8B-Instruct, the lone exception being Llama-3.1-8B IFEval (-1.9 pp). On LLaMA-7B at sequence length 1024, the production-relevant comparison is compile-to-compile: composed with `torch.compile`, LF-LoRA beats compile-matched BP-LoRA by $+19\%$ throughput and 7.3 GB lower reserved memory at default mode, rising to $+24\%$ under reduce-overhead capture. In pure-eager mode the gain over eager BP-LoRA is $+34\%$ throughput and 13.7 GB (27%) lower peak reserved memory. Stacking NF4 and gradient checkpointing reaches 9.26 GB, a footprint driven by NF4 that LF-LoRA composes with at preserved throughput and quality. All memory numbers use `max_memory_reserved`; `max_memory_allocated` (used in prior PEFT efficiency work) misses CUDA-graph private pools and would overstate reduce-overhead capture's advantage by 18 GB here.

1 Introduction

Backpropagation through a frozen large language model is the memory bottleneck of parameter-efficient fine-tuning. LoRA (Hu et al., 2022) stores only a low-rank adapter pair $\{A, B\}$, yet still materializes a global backward graph over the frozen base to transport the error signal. For a 7B model at rank 8, sequence length 1024, and batch size 4, that graph consumes roughly half the activation memory of the run; the trainable parameters take less than 1% of the cost. Production tooling attacks this with hand-written fused backward kernels: Unsloth and Liger-Kernel emit Triton for the adapted projections and their RMSNorm/SwiGLU surroundings. Those kernels are fast, but they are hand-derived per operator and per shape, depend on Triton, and offer no way to check that the gradient they compute is the one autograd would.

We rewrite the standard reverse-mode chain rule for each LoRA-adapted $\mathrm{Act} \circ \mathrm{Norm} \circ \mathrm{Linear}$ sub-block as a layer-local closed-form composition of four forward-cached Jacobian factors—a base-inclusive weight transpose $W_0^\top + A^\top B^\top$, a per-sample normalization Jacobian (scalar gain $s = \gamma/\rho$ plus a rank-one projector), the SwiGLU gate Jacobian, and the activation derivative—run inside three custom `autograd.Functions` (Linear, RMSNorm, SwiGLU) per sub-block. These replace the chain of per-op autograd nodes the standard graph traverses and release intermediates eagerly between firings; softmax, RoPE, residual, and embedding/`lm-head` paths stay in the standard graph.

The closed form is the same Jacobian transpose any autograd engine computes; its value is that it is layer-local, forward-cached, and—unlike a hand-written kernel—auditable against stock autograd, matching BP-LoRA's gradient at median cosine ≥ 0.999 across all 448 (layer, module, $A|B$) entries (§5.1). The production-relevant comparison is compile-to-compile: composed with `torch.compile`, LF-LoRA beats compile-matched BP-LoRA by $+19\%$ throughput and 7.3 GB lower reserved memory at default mode ($+24\%$ under reduce-overhead capture). A Kineto trace attributes the win to intermediate-copy elision—a 70% drop in `aten::copy_` time, not reduced autograd dispatch—together with bf16

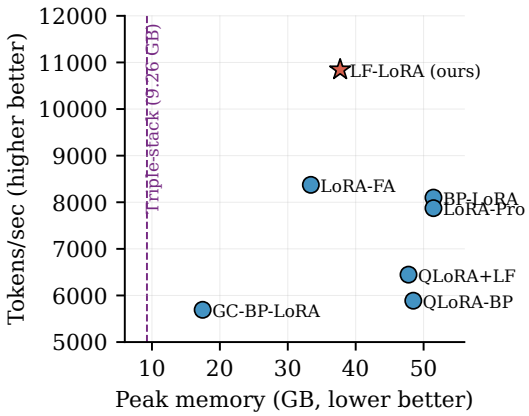

Figure 1: Eager-mode memory–throughput frontier on LLaMA-7B (bs=4, seqlen=1024, 1×H100; up-and-left is better). Eager LF-LoRA sits at the throughput-maximum corner of the LoRA family; the dashed line marks the 9.26 GB consumer-card target hit by the LF + NF4 + gradient-checkpointing triple stack. Compile-mode points are in Table 2, omitted here to keep the eager frontier readable.

LoRA-path matmuls (§5.3); the pure-eager gain over eager BP-LoRA (the baseline no practitioner runs, kept for mechanism attribution) is correspondingly larger at $+34\%$ throughput and 13.7 GB (27%) lower memory, and the fused `Function` composes with 4-bit NF4 and gradient checkpointing down to 9.26 GB. Whole-model fused stacks such as Unsloth, which also rewrite attention and cross-entropy, stay faster in absolute terms; our aim is a checkable, Triton-free, drop-in backend whose gains compose with the existing toolchain, not a raw-speed record.

Contributions.

- **A checkable closed-form backward.** For LoRA-adapted Act ∘ Norm ∘ Linear sub-blocks under per-sample RMSNorm/LayerNorm and SwiGLU gating, the Jacobian transpose decomposes into four forward-cached terms, run inside three custom `autograd.Functions` with no Triton dependency and no per-shape autotuning; the resulting gradient matches BP-LoRA's at median cosine ≥ 0.999 across all 448 per-(layer, module, $A|B$) entries on LLaMA-7B, so the backend is auditable against stock autograd rather than trusted kernel-by-kernel.
- **BP-quality at lower memory and higher throughput.** LF-LoRA matches BP-LoRA within seed-σ on six benchmarks (MMLU, GSM8K, AlpacaEval-2 LC, TruthfulQA-MC1, MT-Bench, full-suite IFEval) across LLaMA-7B and Llama-3.1-8B-Instruct—the lone exception being Llama-3.1-8B IFEval loose-instruction, where it trails by 1.9 pp. In the production-relevant regime—both backends under `torch.compile` at seqlen 1024—it beats compile-BP-LoRA at matched default mode by $+19\%$ throughput and 7.3 GB lower reserved memory ($+24\%$ under reduce-overhead). In pure-eager mode the gain over eager BP-LoRA is $+34\%$ throughput and 13.7 GB (27%) lower peak reserved memory.
- **Consumer-card fine-tuning with honest memory accounting.** Stacking with NF4 and gradient checkpointing fits 7B fine-tuning into 9.26 GB (14.6 GB at 13B). All numbers are reported as `max_memory_reserved`; `max_memory_allocated`, used in prior PEFT efficiency work, does not observe CUDA-graph private pools and would overstate reduce-overhead capture's memory advantage by 18 GB on this workload.

2 Related Work

LoRA and its descendants. LoRA (Hu et al., 2022) freezes the base and trains rank-r adapters $A \in \mathbb{R}^{r\times d}$, $B \in \mathbb{R}^{d\times r}$ such that $h = (W_0 + BA)x$. Subsequent work tightens LoRA along several axes: LoRA-FA (Zhang et al., 2023) freezes A to halve trainable parameters and activation memory; LoRA-Pro (Wang et al., 2025) adds a closed-form gradient correction so LoRA optimization tracks full-FT; BeamLoRA (Yang et al., 2025) adapts rank-subspace allocation via beam search; LoRA-X (Farhadzadeh et al., 2025) projects adapters into the base-model subspace for training-free cross-model transfer. These methods sit downstream of standard backprop; the global backward graph is unchanged. LF-LoRA targets the orthogonal axis: *how* the gradient on $\{A, B\}$ is computed.

Memory-efficient fine-tuning. QLoRA (Dettmers et al., 2023) compresses the static base with 4-bit NF4 quantization, and gradient checkpointing trades wall-clock for activation memory. A growing line of quantization-aware PEFT methods targets the same static-storage axis with adapter-side modifications that absorb the quantization noise. LF-LoRA is orthogonal: it elides the activation-graph axis on the adapted-linear paths

while leaving the static base unchanged. §5.3 verifies composability with 4-bit NF4 (QLoRA backbone) and gradient checkpointing; composition with quantization-aware initialization variants is straightforward in principle (the factorization is independent of how W_0 is stored) and is the natural next step.

Fused-backward kernels and whole-graph compile. Production transformer tooling ships analytical backward implementations for the heavy primitives—fused LayerNorm/RMSNorm kernels in Apex and HuggingFace `transformers`, FlashAttention-style softmax—and `torch.compile`'s Inductor backend further fuses the autograd subgraph across $\mathrm{Linear} \to \mathrm{Norm} \to \mathrm{Act}$ at compile time. LF-LoRA is in the same family: a layer-local fusion plus eager release of intermediates, implemented as three custom `autograd.Functions` (Linear, RMSNorm, SwiGLU) per adapted sub-block. The difference is scope and dispatch: eq. 1 replaces the entire backward of the adapted sub-block with four forward-cached matmuls, so neither the autograd graph nor Inductor needs to traverse it. §5.3 shows that the two contributions compose, with the composed backend beating compile-BP-LoRA at matched default compile mode by $+19\%$ throughput and 7.3 GB lower reserved memory.

Hand-written fused LoRA backends. The closest prior art is the line of hand-written fused Triton/CUDA backward kernels for LoRA-adapted blocks, most prominently Unsloth and the Liger-Kernel family; they fuse the LoRA-adapted backward by emitting Triton kernels for the projections and their RMSNorm, SiLU, and SwiGLU surroundings. LF-LoRA differs on two axes: it stays inside PyTorch's `autograd.Function` surface (no Triton, no per-shape autotuning), and its factorization is closed-form and parametric in the activation/norm VJP rather than hand-derived per kernel and shape. We are careful about what "checkable" buys: a Triton kernel is equally auditable against the same cosine probe (§5.1), so the claim is not that hand-written kernels are unverifiable but the narrower one—a derived VJP is easier to implement correctly and to extend to a new norm or gate, and what must be trusted is one Jacobian transpose, not a kernel per operator and shape. Against Liger—the matched per-op-fusion baseline, installable in our pinned stack (Table 2)—LF-LoRA leads throughput by 8% (10,845 vs. 10,032 tok/s) at 8.7 GB less reserved memory, since Liger fuses the per-op norm/activation kernels but still routes the LoRA backward through autograd. Unsloth sits elsewhere: it fuses the *whole* model—including a linear-cross-entropy that never materializes the logits—and at seqlen 1024 is faster and lighter than LF-LoRA (243.6 ms/step, 27.7 GB) even with FlashAttention-2 disabled. The difference is scope: LF-LoRA fuses only the adapted-linear sub-blocks and leaves attention and `lm-head` in the standard graph, so a whole-model fused stack wins on raw efficiency, while LF-LoRA's edge here is portability—a Triton-free, drop-in `peft` backend with a checkable gradient.

Orthogonal memory axes. GaLore/APOLLO (low-rank optimizer moments), activation offloading, FlashAttention tiling, and 8-bit AdamW target memory axes LF-LoRA does not touch; the axes are disjoint, so they are expected to stack additively, and we leave those head-to-heads to follow-up work.

Random-feedback alignment as a control. Random-feedback alignment (Lillicrap et al., 2016) replaces the symmetric weight transpose with fixed random matrices. We use a randomized-W_0 variant as a control in §5: the forward pass keeps the true frozen W_0, and only the backward transport $\mathrm{grad}_x = g_y(W_0^\top + s\,A^\top B^\top)$ swaps $W_0^\top$ for a fixed random-orthogonal matrix. This is a true feedback-alignment control on the gradient path—not a random-everywhere model whose collapse would be trivial—and it drives MMLU to chance, ruling out any explanation in which LF-LoRA's accuracy comes from random transport rather than the true Jacobian.

3 Layer-Local Backward Decomposition

A LoRA-adapted transformer sub-block has the shape

$$g(x) = \mathrm{Act}\big(\mathrm{Norm}\big(L(x)\big)\big),$$

with $L(x) = (W_0 + BA)x$, W_0 frozen and $\{A, B\}$ trainable. Norm is RMSNorm or LayerNorm with per-sample statistics; Act is SiLU, GELU, or a SwiGLU multiplicative gate. Proposition 1 expresses $J_g^\top$ as a product of four forward-cached factors—the form the fused `autograd.Function` consumes.

Proposition 1 (Layer-local Jacobian factorization)**.** Let $g(x) = \mathrm{Act}(\mathrm{Norm}(L(x)))$ with $L(x) = (W_0 + BA)x$, write $h = L(x)$ for the input to Norm,

and assume per-sample RMSNorm or LayerNorm. For any cotangent vector v at the output of g,

$$J_g^\top \cdot v \;=\; (W_0^\top + A^\top B^\top)\, J_{\text{Norm}}^\top \big(J_{\text{Act}}^\top \cdot v\big), \quad (1)$$

where $J_{\text{Act}}^\top = \text{diag}(\sigma'(u))$ for SiLU/GELU and $J_{\text{Act}}^\top \cdot v = \big(\varphi'(u) \odot v_g \odot v,\ \varphi(u) \odot v\big)$ for SwiGLU (with gate u and up-projection v_g, both equal to $\text{Norm}(h)$ split along the SwiGLU pair). For RMSNorm,

$$J_{\text{Norm}}^\top \cdot v \;=\; s \odot v - \tfrac{h}{d\,\rho^2} \langle s \odot v,\ h \rangle, \quad (2)$$

with $s = \gamma/\rho$ and $\rho = \sqrt{\|h\|^2/d + \varepsilon}$ the per-sample RMSNorm radius (distinct from the LoRA rank r); the LayerNorm form adds a mean-subtraction term (Appendix A). The gradient on the adapter pair is the standard local outer product

$$\Delta B = -\eta\, e\, (Ax)^\top, \quad \Delta A = -\eta\, (B^\top e)\, x^\top,$$

where $e = J_{\text{Norm}}^\top (J_{\text{Act}}^\top \cdot v)$ is the cotangent at the input to Norm (we use e to avoid clashing with the RMSNorm stability constant ε); the activation VJP $J_{\text{Act}}^\top$ is applied first, then the normalization VJP, matching the backward order in Figure 5 and Appendix A. All factors $\{W_0^\top, A, B, \sigma', s, \rho, x, h, \hat{h}, u, v_g\}$ are forward-cached or statically available at layer ℓ.

The algebra is chain-rule composition applied to the explicit Jacobians of RMSNorm/LayerNorm and SwiGLU; the contribution is not algebraic novelty but the resulting layer-local, forward-cached form, which lets the entire adapted-linear sub-block's backward run as one fused `autograd.Function`. All four factors are required for the bf16-cosine match of §5.1, but for downstream *quality* the picture is narrower: §5.4 shows that a principled pointwise treatment of the SwiGLU gate (APPROX-B) still trains to within seed-σ of full LF, whereas dropping the per-sample rank-one normalization correction in eq. 2 (APPROX-A) collapses training. The rank-one correction is thus load-bearing for quality; the SwiGLU multiplicative-gate term is required only for the exact gradient match.

Scope. Eq. 1 covers only the LoRA-adapted linear paths: $\{q, k, v, o, \text{gate}, \text{up}, \text{down}\}$-proj at every layer. The attention softmax, RoPE rotation, KV-cache slicing, residual sum, and embedding/`lm-head` paths remain in the standard autograd graph. §5.3 reports the residual non-adapted backward cost (about 3.0 GB on LLaMA-7B at bs=4, seqlen=1024). A full derivation appears in Appendix A.

Extending to new activations and norms. Eq. 1 is parametric in the activation and normalization VJPs, so adding a variant is a local derivation, not a new kernel: GELU/SiLU already enter through $J_{\text{Act}}^\top = \text{diag}(\sigma'(u))$; LayerNorm (with or without learnable bias) adds a mean-subtraction term (Appendix A); a new gating or norm rule needs only its own Jacobian transpose. Mixture-of-Experts fits unchanged—each expert FFN is an $\text{Act} \circ \text{Norm} \circ \text{Linear}$ sub-block, while the non-adapted router stays in the standard graph.

4 LF-LoRA Algorithm

For each adapted block ℓ we replace the standard backward pass with the layer-local update of Proposition 1. Concretely, every adapted Linear is wrapped by an `LFLoRALinear` module whose forward computes $y = xW_0^\top + s_{\text{LoRA}}\,(xA^\top)B^\top$ inside a custom `autograd.Function`; the backward returns $\text{grad}_x = g_y\,(W_0^\top + s_{\text{LoRA}}\,A^\top B^\top)$ in closed form. The RMSNorm and SwiGLU forwards are monkey-patched to route through custom `autograd.Functions` that produce eq. 2 and the SwiGLU gate Jacobian respectively. The outer optimizer is AdamW; nothing else in the training loop changes. The full per-block backward as ordered pseudocode—four factor lines plus the LoRA outer-product update, applying $J_{\text{Act}}^\top$ first then $J_{\text{Norm}}^\top$—is boxed in Figure 5 (Appendix A).

Compatibility with 4-bit base weights. For QLoRA-style 4-bit NF4 bases, `LFLoRALinear4bitFunction` keeps the bitsandbytes `Linear4bit` as a child module and avoids materializing a bf16 base. The forward uses the fused `matmul_4bit` kernel; the backward dequantizes W_0 to a transient bf16 buffer just long enough to compute $g_y\,W_0^\top$, then releases it. The NF4 round-trip stays within the bf16-noise floor empirically: median gradient cosine ≥ 0.998 vs. QLoRA-BP in §5.1.

Compatibility with gradient checkpointing. Gradient checkpointing wraps each decoder layer in `torch.utils.checkpoint`; the wrapped layer is re-executed on backward recompute, so the LF-LoRA custom `Function` instances inside it run twice, but the math is unchanged.

5 Experiments

We fine-tune LLaMA-7B (Touvron et al., 2023) and Llama-3.1-8B-Instruct (Grattafiori et al., 2024)

on Alpaca-52k (Taori et al., 2023), with LoRA rank $r = 8$ on $\{q, k, v, o\} \oplus \{\text{gate,up,down}\}$ at every layer. AdamW, learning rate 2×10^{-4}, batch size 4 with gradient accumulation 8 (effective 32), sequence length 1024, bf16, 3 epochs. All numbers are mean $\pm$ std over 3 seeds on 1×H100. Evaluations: MMLU 5-shot (Hendrycks et al., 2021), GSM8K 8-shot CoT (Cobbe et al., 2021), AlpacaEval-2 length-controlled (Dubois et al., 2024) (logprob judge), TruthfulQA-MC1 (Lin et al., 2022), MT-Bench (Zheng et al., 2023) judged by `gpt-4o-2024-11-20` (12/1920 Azure rejections excluded), and IFEval (Zhou et al., 2023) over all 541 prompts with the full official instruction-checker set (Appendix G).

§5.2 (downstream-quality match) and §5.3 (memory–throughput frontier, including cross-architecture, long-context, and rank corners) are the load-bearing experiments: together they establish that the LF-LoRA backend produces BP-quality models at lower memory and higher throughput. §5.1 (gradient cosine) is the bf16 sanity check; §5.4 establishes the necessity of the four Jacobian factors, distinguishing the structurally required ones from those a port may approximate.

5.1 Implementation sanity check: gradient cosine against BP-LoRA

On LLaMA-7B, the LF-LoRA gradient agrees with BP-LoRA at cosine 0.9997 median (0.9992 min) across all 448 per-(layer, module, $A|B$) entries (32 layers $\times$ 7 adapted modules $\times \{A, B\}$), with 100% clearing 0.998 (Figure 2). The attention and MLP module classes are indistinguishable (medians 0.99974 vs. 0.99972, both above the 0.998 floor on every entry). The probe runs both methods on identical forward-cached inputs after a 10-step warmup to break the LoRA $B = 0$ initialization degeneracy. Under 4-bit NF4 base weights the median cosine against QLoRA-BP is 0.9996. Magnitudes match: the median gradient-norm ratio $\|g_{\text{LF}}\|/\|g_{\text{BP}}\|$ is within 0.06% of 1, ruling out a systematic effective-learning-rate shift; norm-ratio and relative-L_2 details are in Appendix B.

In real arithmetic eq. 1 is an exact reordering of what BP computes, but under bf16 (and, for QLoRA, the NF4 dequant round-trip) the per-element matmul ordering differs, so we treat the BP agreement as an empirical claim bounded by the cosine floor above, not a bit-identity. A BP-vs-BP control (Appendix B) confirms this is the bf16 floor, not LF error: against a shared fp32

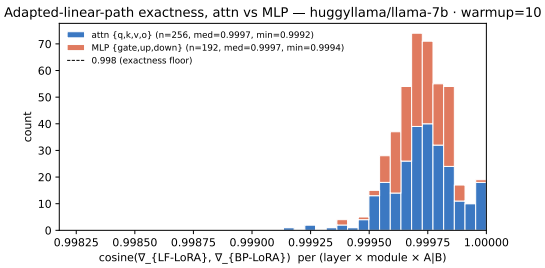

Figure 2: Per-(layer, module, $A|B$) gradient cosine between LF-LoRA and BP-LoRA on LLaMA-7B, after a 10-step warmup. Attention ($n = 256$) and MLP ($n = 192$) entries both clear the 0.998 floor at 100% frequency; medians are 0.99974 and 0.99972.

reference, bf16-LF sits no farther from the true gradient (3.2–3.3% rel-L_2) than bf16-BP itself (3.3–3.6%). The ≥ 0.998 floor on all 448 LLaMA-7B entries together with the seed-σ-matched downstream evals of §5.2 are the two pieces of evidence we claim; cross-family results push the floor to a single-minibatch 0.975 outlier on Qwen2.5 (Appendix H).

5.2 Quality matches BP-LoRA at lower memory

On Llama-3.1-8B-Instruct (GQA), LF-LoRA matches BP-LoRA within seed-σ on five of six evaluations: MMLU 63.14 vs. 63.51, GSM8K 59.39 vs. 57.44, AlpacaEval-2 LC 81.08 vs. 81.20, TruthfulQA 28.23 vs. 28.11, and MT-Bench (gpt-4o judge) 6.03 vs. 6.09 on the 1–10 scale. The exception is full-suite IFEval loose-instruction, where LF-LoRA trails by 1.87 pp (66.81 vs. 68.68, 5 seeds)—the only cell whose gap exceeds seed-σ, which we flag rather than round away. Since the gradient control (§5.1) shows LF adds no error beyond BP's bf16 floor, we read this gap as seed-level variance on one sensitive metric, not LF-specific divergence. Peak training memory drops from 64.5 to 50.1 GB (22%).

The LLaMA-7B (MHA) stanza of Table 1 reports the same matching pattern on a weaker base. LLaMA-1 7B's pretraining is light on math and instruction-following, so absolute downstream levels (GSM8K 7.63 vs. 8.01, MMLU 37.64 vs. 37.36, MT-Bench 3.44 vs. 3.50) sit at the noise floor regardless of fine-tuning method; we treat Llama-3.1-8B as the modern baseline. The randomized-W_0 control drives MMLU to chance (23.10%), ruling out a random-feedback explanation of LF-LoRA's accuracy.

	MMLU (5-sh)	GSM8K (8-sh)	AlpEval (LC WR)	TQA (MC1)	MT-B (gpt-4o)	IFE-li (%)	Peak (GB)
Llama-3.1-8B-Instruct (NousResearch redistribution, GQA + RMSNorm + SwiGLU)							
BP-LoRA	63.51 ± 0.25	57.44 ± 3.66	81.20 ± 0.38	28.11 ± 0.55	6.09 ± 0.03	$68.68 \pm 1.18^{\dagger}$	64.5 ± 0.1
LF-LoRA (ours)	63.14 ± 0.42	59.39 ± 1.19	81.08 ± 0.59	28.23 ± 0.63	6.03 ± 0.13	$66.81 \pm 1.11^{\dagger}$	50.1 ± 0.1
LLaMA-7B (huggyllama/llama-7b, MHA; LLaMA-1 pretraining limits absolute downstream)							
BP-LoRA	37.36 ± 0.94	8.01 ± 0.83	25.05 ± 3.05	23.22 ± 0.25	3.50 ± 0.27	39.13 ± 0.48	51.8 ± 0.0
LF-LoRA (ours)	37.64 ± 0.23	7.63 ± 1.03	23.81 ± 1.01	23.22 ± 0.28	3.44 ± 0.15	39.05 ± 0.90	38.2 ± 0.0
LF-LoRA-randomW0	23.10	0.00	20.99	–	–	–	51.1 ± 0.0

Table 1: LF-LoRA matches BP-LoRA within seed-σ on every evaluation across both base models except Llama-3.1-8B IFE-li (where it trails by 1.9 pp on the full 541-prompt suite), with 25–27% less peak memory; the randomized-W_0 control collapses MMLU and GSM8K to chance. MT-Bench is on a 1–10 scale (gpt-4o); IFE-li is full-suite IFEval loose-instruction; other columns are accuracy (%). † Llama-3.1-8B IFE-li is over five seeds; other cells are three.

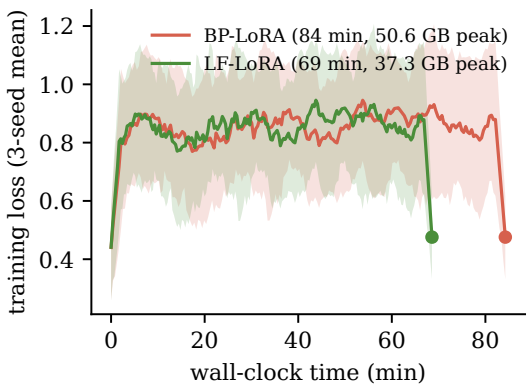

Figure 3: LF-LoRA tracks BP-LoRA loss-for-loss across the full 3-epoch SFT run (LLaMA-7B + Alpaca-52k, rank-8 LoRA, 3 seeds, 11-point smoothed) and finishes about 15 minutes earlier. Peak-memory annotations are the training-loop `torch.cuda` peak; the $\sim$1 GB gap to Table 1 is periodic eval-pass tensors.

The Peak (GB) column is the maximum CUDA memory over the full 3-epoch SFT run (AdamW state and grad-accumulation buffer included; no in-training evaluation runs). The full-run BP→LF gap—14.4 GB on Llama-3.1-8B, 13.6 GB on LLaMA-7B—is the activation-graph elision at the training sequence length, which the seqlen-1024 microbench reproduces (13.7 GB reserved, Table 2); it scales with sequence length, so a short-batch microbench understates it (Appendix J).

5.3 Memory–throughput frontier

Two findings drive Table 2. First, eager LF-LoRA reaches 10,845 tokens/s at 37.71 GB `peak_reserved`, against eager BP-LoRA's 8,100 tokens/s at 51.45 GB—a 34% throughput gain and 13.7 GB (27%) memory reduction in pure-PyTorch eager mode, with no Inductor or Triton dependency. Second, the layer-local form composes with `torch.compile` sub-additively: the composed backend reaches 13,801 tokens/s at 30.83 GB in default mode (14,253 tokens/s at 31.15 GB under reduce-overhead capture), the Pareto-optimal corner of the LoRA-family frontier, and beats compile-BP-LoRA by +19% throughput and 7.3 GB at default mode (+24% under reduce-overhead). The default-mode delta is the production-relevant comparison. The composition is not seamless: the custom `autograd.Functions` graph-break Dynamo, so the LF backward stays in eager while Inductor compiles the surrounding non-adapted paths—the +19% is what survives that graph break (Appendix J), so the per-mechanism split below is a decomposition, not a clean additive sum; making the `Functions` Dynamo-traceable is future work.

Per-mechanism attribution. Table 3 isolates the mechanisms: eager activation release (MLP-checkpointing) costs 17% throughput; per-op fusion (fused-RMSNorm) adds 10%; whole-model Inductor compile fuses the autograd subgraph for +43%. The layer-local form addresses a disjoint cost—the materialization and copying of per-op intermediate activations, plus the fp32 LoRA-path matmuls a stock `peft` adapter issues (Appendix J profiles both)—so of the +70% over eager BP-LoRA, Inductor contributes +43% and the layer-local form a further +19% at 7.3 GB lower reserved memory.

Memory metric. Prior PEFT efficiency work reports `max_memory_allocated`, which does not observe the private pool that reduce-overhead CUDA-graph capture uses: the reduce-overhead rows read 13.16 GB allocated but 31–38 GB reserved (the

Method	step (ms)	tok/s	peak_alloc (GB)	peak_reserved (GB)
BP-LoRA	505.7	8100	49.46	51.45
LoRA-FA	489.2	8373	31.60	33.40
LoRA-Pro	520.2	7873	49.46	51.45
GC-BP-LoRA	719.6	5692	16.33	17.46
QLoRA-BP	695.7	5887	45.04	48.48
QLoRA+LF	635.4	6447	45.05	47.78
LF-LoRA (ours, eager)	377.7	10845	36.48	37.71
Fused-backward BP-LoRA baselines				
MLP-ckpt-BP-LoRA	608.4	6733	32.78	34.91
fused-norm-BP-LoRA	458.6	8932	45.33	49.12
compile-BP-LoRA (default)	353.3	11595	37.76	38.12
compile-BP-LoRA (reduce-overhead)	357.3	11465	13.16	38.33
Hand-written fused LoRA backends (reviewer baselines)				
Liger-BP-LoRA	408.3	10032	44.66	46.38
Unsloth-LoRA[‡] (FA2=off)	243.6	16818	27.52	27.71
Compose with `torch.compile` *(ours)*				
compile-LF-LoRA (default)	**296.8**	**13801**	**30.56**	**30.83**
compile-LF-LoRA (reduce-overhead)	**287.4**	**14253**	**13.16**	**31.15**
QLoRA+LF+gc	859.5	4766	8.83	10.32

Table 2: LLaMA-7B memory–throughput frontier (bs=4, seqlen=1024, $1\times$H100; 30 warmup + 100 measure full fwd+bwd+opt steps on a full-length, fully-attended batch). Memory is `peak_reserved`: `max_memory_allocated` does not observe CUDA-graph private pools and understates reduce-overhead capture memory by ~ 18 GB here. The QLoRA+LF+gc row uses the same microbench protocol (10.32 GB reserved); the 9.26 GB consumer-card number cited in §5.3 and Figure 1 is the full 3-epoch SFT-run peak. [‡] Unsloth row was measured with xformers attention in its isolated `.venv-unsloth` (PyTorch 2.10.0; see Appendix K); FlashAttention-2 would not build against that pinned stack, so this is not Unsloth's peak on a tuned stack. See §2.

Mechanism	tok/s	Δ vs BP	peak_res (GB)
BP-LoRA (eager baseline)	8100	—	51.45
Eager release (MLP-ckpt)	6733	-17%	34.91
Per-op kernel fusion (fused-norm)	8932	$+10\%$	49.12
Inductor compile (BP)	11595	$+43\%$	38.12
LF-LoRA (eager, ours)	10845	$+34\%$	37.71
LF + compile (ours)	**13801**	$+70\%$	**30.83**

Table 3: Per-mechanism attribution on LLaMA-7B (bs=4, seqlen=1024, $1\times$H100). Each row isolates one prior-art mechanism against the eager BP-LoRA baseline; the layer-local form (LF-LoRA) addresses a disjoint cost and composes with Inductor compile for the bottom-row $+70\%$ throughput / -40% memory at default mode.

18–25 GB gap is the graph pool), so Table 2 and all comparative claims use `max_memory_reserved` (per-submodule and step-time decompositions in Appendix J).

Of LF-LoRA's 37.71 GB `peak_reserved` at seqlen 1024, ~ 13.5 GB is the static base; LF elides the adapted-linear activation share (the 13.7 GB saving over BP), leaving the non-adapted graph (attention softmax, RoPE, `lm_head` logits) as the sequence-length-scaling residual.

Stacking with NF4 and gradient checkpointing. The three orthogonal mechanisms (LF-LoRA for activation-graph elision, 4-bit NF4 for static-storage compression, gradient checkpointing for activation re-materialization) stack to 9.26 GB peak memory, deterministic across 3 seeds—within the 12 GB envelope of a consumer card, though the 9.26 GB is an H100 `peak_reserved` figure, so the consumer-card fit is inferred, not demonstrated on-card (fragmentation and kernel availability could shift it). We are explicit about attribution: the dominant lever is NF4 (a 4-bit base is what brings 7B under the 12 GB line at all—bf16 alone is ~ 13.5 GB), with gradient checkpointing removing most of the residual activation memory. LF-LoRA's own activation-graph elision—13.7 GB on a bf16 base—is largely *subsumed* once the base is quantized: QLoRA-BP and QLoRA+LF microbench at 48.48 vs. 47.78 GB reserved (LF only 0.7 GB lower at seqlen 1024), because the NF4 backward dequantizes W_0 to a transient bf16 buffer that re-injects the tensor the elision would avoid. LF's role

Model	LF-LoRA		BP-LoRA		Δ (%)	
	ms/step	GB	ms/step	GB	wall	mem
LLaMA-7B	377.7	37.7	505.7	51.5	−25	−27
Mistral-7B-Inst.	384.6	42.1	519.0	56.0	−26	−25
Qwen2.5-7B-Inst.	371.2	50.8	504.3	63.7	−26	−20
LLaMA-13B	627.2	62.6	OOM	OOM	—	—

Table 4: Cross-family / cross-scale memory–throughput summary. Microbench, bs=4, seqlen=1024, 1×H100. Gradient cosine medians ≥ 0.999 across every (layer, module) entry.

in the quantized stack is to preserve throughput and BP-level quality, not to drive the memory figure. Under matched 3-epoch compute the triple stack reaches MMLU 35.26 ± 0.46 vs. QLoRA-BP's 35.20 ± 1.06 and matches QLoRA+LF on full-suite IFEval-loose (38.53 vs. 38.29). At LLaMA-13B it reaches 14.6 GB at MMLU 46.74, matching plain LF-LoRA (46.72).

Cross-architecture and scale. Table 4 extends the frontier to Mistral-7B (sliding-window attention), Qwen2.5-7B (GQA), and LLaMA-13B. LF-LoRA holds a 25–26% wall-clock and 13–14 GB (20–27%) memory advantage at gradient cosine ≥ 0.999, with downstream MMLU/GSM8K/AlpacaEval within 0.76 pp of BP-LoRA (Appendix H); at LLaMA-13B, BP-LoRA OOMs at bs=4/seqlen=1024 while LF-LoRA completes (62.6 GB microbench; full run 108 min at 63.7 GB vs. BP+gc's 190.5 min). The bf16 elision also opens one extra long-context octave—LF reaches seqlen 8k at 59.1 GB where BP and both QLoRA variants OOM (Appendix D)—and across $r \in \{4, \ldots, 64\}$ the memory advantage holds flat at $\sim$13.7 GB while the per-step throughput advantage grows from 114 to 171 ms as BP's fp32 LoRA matmuls scale with rank (Appendix E).

5.4 Necessity of the layer-local factorization

Proposition 1 carries four Jacobian factors. Figure 4 partitions eight LLaMA-7B / Alpaca-52k SFT runs into a trichotomy. Full LF-LoRA and APPROX-B—which treats the SwiGLU multiplicative gate as a pointwise activation—both train to within seed-σ of full LF on MMLU (37.6% vs. 36.7%). The four single-factor strips (A1–A4), APPROX-A (which freezes the RMSNorm radius ρ to a detached running statistic, 22.9%), and the randomized-W_0 control (P1) all collapse to within 2 pp of chance. So the per-sample rank-one normalization correction in eq. 2 is structurally required for training, not only for the bf16-cosine identity,

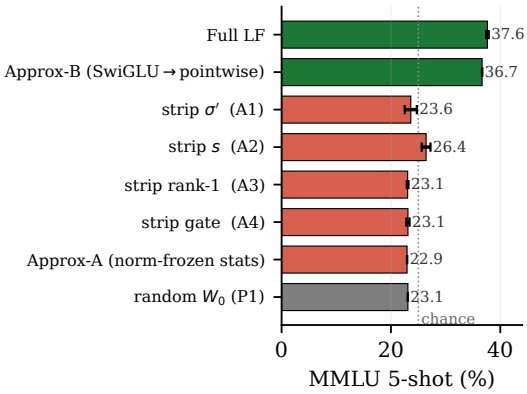

Figure 4: Trichotomy on LLaMA-7B (3 seeds for Full LF and strips A1–A4; single seed for APPROX-A/B; chance = 25%). *Green*: factorizations that train within seed-σ of full LF. *Red*: backward forms with one mis-computed factor; all collapse to chance. *Gray*: random-W_0 feedback control. The split shows that not every Jacobian factor is equally load-bearing for downstream quality, but every factor is required for the bf16-cosine match of §5.1.

while the SwiGLU gate term is an engineering simplification a port can drop if it accepts breaking the cosine match (per-seed numbers and full APPROX-A/B evals in Appendix I).

6 Conclusion

LF-LoRA gives each adapted $\mathrm{Act} \circ \mathrm{Norm} \circ \mathrm{Linear}$ backward a closed form—four forward-cached Jacobian factors in three custom `autograd.Functions`—auditable against stock autograd at cosine ≥ 0.999, matching BP-LoRA across six evaluations on two base models while beating compile-BP-LoRA by +19% throughput and 7.3 GB at default mode and stacking with NF4 and gradient checkpointing to 9.26 GB.

Three implications follow. First, the 9.26 GB triple-stack footprint sits within a 12 GB consumer card's envelope (an H100 measurement, not demonstrated on-card)—primarily an NF4 result, with LF contributing matched throughput and quality, not the memory reduction. Second, as a backward *backend* rather than an architecture or training-rule change, LF composes with the LoRA toolchain (Table 3): the production-relevant delta is +19% / 7.3 GB over compile-BP-LoRA, the headline +70% over *eager* BP-LoRA being mostly Inductor (+43%). Third, the non-adapted graph (attention, RoPE, `lm-head`) that LF leaves intact is the natural next target.

Limitations

Scope of the gradient claim. Eq. 1 is the standard reverse-mode chain rule for $\mathrm{Act} \circ \mathrm{Norm} \circ L$ rewritten in a layer-local, forward-cached form; in real arithmetic it is exact, and under bf16/NF4 the empirical bound is the cosine probe in §5.1. We do not evaluate approximate-gradient alternatives (random feedback, frozen-statistics norm, dropped-factor variants) as training algorithms in their own right—the strip ablations show only that omitting any factor breaks the BP-LoRA match, not whether other transport rules might also work.

Scope. The bf16-noise-level gradient match holds on the LoRA-adapted linear paths: $\{q, k, v, o, \text{gate}, \text{up}, \text{down}\}$-proj. The attention softmax (cross-token coupling), RoPE rotation, KV-cache slicing, residual sum, and embedding/`lm-head` paths remain in the standard autograd graph. A closed-form softmax / cross-token decomposition is future work.

Whole-model fused stacks outperform on raw speed. LF-LoRA fuses only the adapted-linear sub-blocks. Stacks that also rewrite attention and the cross-entropy/`lm-head`—most prominently Unsloth—cut more of the activation graph, and at seqlen 1024 run faster and lighter than LF-LoRA in absolute terms (Table 2), even with FlashAttention-2 disabled on our cluster. LF-LoRA's contribution against this line is a checkable, Triton-free, drop-in `peft` backend whose gains compose with the existing toolchain, not a raw-speed record; closing the residual non-adapted graph (§6) would narrow the gap and is future work.

Statistical scope of the equivalence claim. We report whether LF and BP downstream metrics overlap within seed-σ (3 seeds; 5 for Llama-3.1 IFEval), not a formal equivalence test such as TOST, which on three seeds and noise bands as wide as ± 3.66 (BP GSM8K on Llama-3.1) would have little power. The claim is therefore the weaker one—no metric separates beyond seed noise except Llama-3.1-8B IFEval loose-instruction, which we flag (§5.2)—rather than a positive equivalence guarantee. The gradient-level control of §5.1 (LF sits at BP's own bf16 noise floor, not beyond it) is the stronger evidence that the two backends compute the same update.

Single dataset; bounded scale and family breadth. The factorization is verified on five base models spanning two families and two scales: LLaMA-7B (MHA), Llama-3.1-8B-Instruct (GQA, 8 KV heads, 32 query heads), Mistral-7B-Instruct-v0.1 (sliding-window attention), Qwen2.5-7B-Instruct (GQA, 4 KV heads), and LLaMA-13B (Appendix H). The headline SFT and downstream-eval matrix (six benchmarks $\times$ 3 seeds) is run on the first two; the latter three are validated at the gradient-identity and memory-frontier level. Scale beyond 13B (e.g. LLaMA-2-70B), instruction-tuning corpora beyond Alpaca-52k (ShareGPT, FLAN, OpenHermes), and non-SFT regimes (DPO, continued pretraining) remain untested. The factorization in eq. 1 is architecture-agnostic over $\mathrm{Act} \circ \mathrm{Norm} \circ L$ blocks, so the engineering port is straightforward; the open question is whether scale or different data distributions surface any of the higher-order interactions that the strip ablations rule out at 7B.

LoRA rank: headline at $r = 8$, sweep at $r \in \{4, 16, 32, 64\}$. The headline configuration uses $r = 8$. Table 7 sweeps the memory–throughput frontier and gradient cosine over $r \in \{4, 8, 16, 32, 64\}$; downstream quality at non-headline ranks is not separately evaluated.

Single-H100; distributed training untested. All experiments run on one H100-80GB. Tensor/pipeline parallelism, FSDP/ZeRO sharding, and multi-node setups are untested. The custom `autograd.Function` per adapted sub-block is data-parallel-safe (no cross-rank state), but whether the memory and throughput gains compose with TP/PP graph cuts and collective-communication schedules—where the autograd graph crosses device boundaries and activation memory composes with comms—is open. Comparison against distributed efficient-training stacks (DeepSpeed ZeRO-3, FSDP + activation checkpointing, Megatron fused kernels) is the natural next system-integration step.

Judges and rejections. AlpacaEval-2 LC uses a logprob judge (no OpenAI API), which is known to bias toward shorter outputs; absolute win-rates should be read with that caveat. MT-Bench is judged by gpt-4o (Azure snapshot `gpt-4o-2024-11-20`); 12 of 1920 judge calls ($< 0.7\%$) were rejected by the Azure content filter and excluded from the mean (the exclusion is paired across methods; see Appendix G).

Triple-stack IFEval gap (resolved on the full suite). An earlier 15-id IFEval subset showed the triple stack (LF-LoRA + NF4 + gradient checkpoint-

ing) trailing its no-gc counterpart by 1.05 pp on loose-instruction. On the full 541-prompt suite the sign reverses to a 0.24 pp lead (38.53 vs. QLoRA+LF 38.29), so the apparent deficit was a subset-sampling artifact, not a stack effect—consistent with gradient checkpointing producing identical gradients up to dropout/RNG ordering. We keep the note because the original subset value appeared in an earlier draft.

Ethical Considerations

LF-LoRA is a low-risk methods contribution. End-to-end compute is approximately 100 H100-GPU-hours; the downstream artifact is a fine-tuned LLM adapter on a base model whose licensing already covers research use (LLaMA-1 community / Llama-3 community). No human subjects, no PII, no new deployment surface beyond what the base model itself presents. The memory-reduction nature of the contribution is dual-use: it lowers the compute barrier to fine-tuning frontier LLMs, which makes both legitimate research and adversarial misuse cheaper. We flag this trade-off explicitly. All scientific contributions—theorem, mechanism, experimental design, claim audit—are author work; standard AI coding and writing tools were used in a supporting role under author verification, consistent with the ACL policy.

The implementation is a focused set of custom `autograd.Functions` and monkey-patches over standard `transformers` / `peft` / `bitsandbytes` modules ($\sim$2k LoC in total) plus the training and evaluation scripts used in §5; it has no Triton dependency and no per-shape autotuning, and changes neither the optimizer, learning rate, nor training rule, so the gains of §5.3 are attributable to the backward backend in isolation. We plan to release the full source, the SLURM job scripts, and the evaluation harness under a permissive license on acceptance, and to package the backend as a drop-in `peft`-compatible plugin—a backend flag on the existing LoRA config rather than a fork—so that it drops into any existing `peft`-based fine-tuning pipeline as a backend swap.

References

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient finetuning of quantized LLMs. In *Advances in Neural Information Processing Systems*.

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B. Hashimoto. 2024. Length-controlled AlpacaEval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*.

Farzad Farhadzadeh, Mehdi Hosseinzadeh, Joseph Paul Cohen, and Shahab Bakhtiari. 2025. LoRA-X: Bridging foundation models with training-free cross-model adaptation. In *International Conference on Learning Representations*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, and 1 others. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *International Conference on Learning Representations*.

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

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. 2016. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 7:13276.

Stephanie Lin, Jacob Hilton, and Owain Evans. 2022. TruthfulQA: Measuring how models mimic human falsehoods. In *Annual Meeting of the Association for Computational Linguistics*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford Alpaca: An instruction-following LLaMA model. https://github.com/tatsu-lab/stanford_alpaca.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, and 1 others. 2023. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Zhengbo Wang, Jian Liang, Ran He, Zilei Wang, and Tieniu Tan. 2025. LoRA-Pro: Are low-rank adapters properly optimized? In *International Conference on Learning Representations*.

Naibin Yang, Yilong Jia, Huasong Liu, Hua Li, Mingxu Liu, Furu Wei, Yanyan Lin, Wayne Xin Zhou, and Ji-Rong Wen. 2025. BeamLoRA: Beam-constraint low-rank adaptation. In *Annual Meeting of the Association for Computational Linguistics*.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. 2023. LoRA-FA: Efficient and effective low rank representation fine-tuning. *arXiv preprint arXiv:2308.03303*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A Derivation of Proposition 1

We derive eq. 1 by composing the chain rule along the three local maps L, Norm, Act and tracking each Jacobian transpose to a forward-cached factor.

Setup. Fix an adapted block ℓ with input $x \in \mathbb{R}^d$. Let

$$h = L(x) = (W_0 + BA)\, x,$$

$$y = \mathrm{Norm}(h), \qquad a = \mathrm{Act}(y),$$

where y denotes the activation input—either a single vector u for SiLU/GELU, or the pair (u, v_g) for SwiGLU. For a downstream cotangent v at the output of a, define

$$\begin{aligned} v_{\mathrm{Norm}} &= J_{\mathrm{Act}}^{\top} \cdot v, \\ v_L &= J_{\mathrm{Norm}}^{\top} \cdot v_{\mathrm{Norm}}, \\ v_x &= J_L^{\top} \cdot v_L. \end{aligned}$$

We compute each factor in turn.

Linear factor. L is linear with matrix $W_0 + BA$. Therefore

$$J_L^{\top} \cdot v_L = (W_0^{\top} + A^{\top} B^{\top})\, v_L, \tag{3}$$

which is the first factor of eq. 1. Note that $W_0^{\top}$ is the *forward* weight transposed (not a separate feedback matrix), so no extra storage is required at layer ℓ.

Activation factor. For SiLU/GELU $\mathrm{Act}(u) = \sigma(u)$ pointwise (with $u = \mathrm{Norm}(h)$), $J_{\mathrm{Act}}^{\top} = \mathrm{diag}(\sigma'(u))$ and $v_{\mathrm{Norm}} = \sigma'(u) \odot v$. For SwiGLU $\mathrm{Act}(u, v_g) = \varphi(u) \odot v_g$ with gate u, up-projection v_g, and $\varphi = \sigma$ (SiLU),

$$\frac{\partial(\varphi(u) \odot v_g)}{\partial(u, v_g)} = \big(\, \varphi'(u) \odot v_g,\ \varphi(u)\, \big),$$

so $v_{\mathrm{Norm}} = (\varphi'(u) \odot v_g \odot v,\ \varphi(u) \odot v)$. Both u, v_g, $\varphi'(u)$, $\varphi(u)$ are forward-cached.

Normalization factor. For RMSNorm with per-sample radius $\rho = \sqrt{\|h\|^2/d + \varepsilon}$ (distinct from the LoRA rank r) and gain γ (acting on $h = L(x)$),

$$\mathrm{Norm}(h) = \frac{\gamma}{\rho} \odot h = s \odot h, \quad s \equiv \gamma/\rho.$$

Although Norm looks linear in h, ρ depends on h through $\|h\|^2 = \sum h_i^2$. Differentiating,

$$\frac{\partial(s_i h_i)}{\partial h_j} = s_i\, \delta_{ij} - \frac{\gamma_i}{\rho^3} \frac{h_i h_j}{d}.$$

LF-LoRA backward at one adapted $\mathrm{Act} \circ \mathrm{Norm} \circ$ Linear sub-block.
Input: cotangent v at $g(x)$; cached $\{x,\ h,\ s,\ \rho,\ \sigma',\ u,\ v_g\}$. *Output*: $\nabla x,\ \nabla A,\ \nabla B$.

$$\begin{aligned} v_a &\leftarrow J_{\mathrm{Act}}^{\top} \cdot v && \text{(act VJP; SwiGLU folds in } v_g\text{)} \\ e &\leftarrow s \odot v_a - \tfrac{h}{d\rho^2}\, \langle s \odot v_a,\ h \rangle && \text{(RMSNorm rank-1 projector)} \\ \nabla x &\leftarrow (W_0^{\top} + A^{\top} B^{\top})\, e && \text{(base-inclusive transpose)} \\ \nabla B &\leftarrow e\, (Ax)^{\top}, \quad \nabla A \leftarrow (B^{\top} e)\, x^{\top} && \text{(LoRA outer products)} \end{aligned}$$

Figure 5: LF-LoRA's per-block backward, run as one fused `autograd.Function` per sub-block. The activation VJP $J_{\mathrm{Act}}^{\top}$ is applied first, then the RMSNorm VJP (eq. 2), then the base-inclusive transpose; each row maps to one factor of eq. 1, and the autograd graph is never materialised between g's output and x. Under QLoRA, W_0 is dequantised to a transient bf16 buffer only for the line-3 matmul, then released.

Transposing and contracting with v_{Norm}, and writing $\tilde{v} = v_{\mathrm{Norm}}$ for brevity,

$$\begin{aligned} (J_{\mathrm{Norm}}^{\top} \cdot \tilde{v})_j &= s_j\, \tilde{v}_j - \tfrac{h_j}{d\,\rho^3} \langle \gamma \odot \tilde{v}, h \rangle \\ &= s_j\, \tilde{v}_j - \tfrac{h_j}{d\,\rho^2} \langle s \odot \tilde{v}, h \rangle, \end{aligned}$$

which is the RMSNorm line of eq. 2. The LayerNorm case follows by adding a mean-subtraction term; details omitted for brevity but identical in structure.

Composition. Composing the three factors in order:

$$\begin{aligned} v_x &= J_L^{\top} \cdot \big(J_{\mathrm{Norm}}^{\top} \cdot J_{\mathrm{Act}}^{\top} \cdot v\big) \\ &= (W_0^{\top} + A^{\top} B^{\top})\, J_{\mathrm{Norm}}^{\top} \big(J_{\mathrm{Act}}^{\top} \cdot v\big), \end{aligned}$$

which is eq. 1.

Gradient on the adapter pair. The adapter loss gradient flows through L:

$$\begin{aligned} \tfrac{\partial L_\ell}{\partial B} &= v_L\, (Ax)^{\top}, \\ \tfrac{\partial L_\ell}{\partial A} &= (B^{\top} v_L)\, x^{\top}, \end{aligned}$$

which gives the local outer-product update $\Delta B = -\eta\, v_L (Ax)^{\top}$, $\Delta A = -\eta\, (B^{\top} v_L)\, x^{\top}$; the symbol e used in Proposition 1 is the same quantity, $e \equiv v_L$ (renamed to avoid clashing with the RMSNorm stability constant ε).

Limiting case: frozen-statistics norm and pointwise activation. If normalization statistics are frozen (e.g., BatchNorm running averages) rather than per-sample, the dependence of r on h is severed, the

second term in the RMSNorm line of eq. 2 vanishes, and

$$J_{\text{Norm}}^{\top} \cdot v \;=\; s \odot v.$$

If additionally the activation is pointwise (no SwiGLU), eq. 1 reduces to $J_g^{\top} \cdot v = L^{\top}\big(s \odot \sigma'(u) \odot v\big)$. The two factors required to lift this simplified form into a modern transformer are therefore (i) the rank-one projector $-h\langle s \odot v, h\rangle/(d\,\rho^2)$ from per-sample statistics, and (ii) the SwiGLU v_g-multiplication term; Figure 4 (A3, A4) confirms that these are the two strips that collapse MMLU farthest when removed.

B Gradient-cosine probe details

The probe behind Figure 2 (main body) constructs parallel LF-LoRA and BP-LoRA models on the same base with synchronized A, B initialization, runs N minibatches with a 10-step warmup to break the LoRA $B = 0$ initialization degeneracy, then computes per-(layer, module) cosine on ∇_A and ∇_B.

Norm-ratio and relative-L_2 details. On the same probe, the median gradient-norm ratio $\|g_{\text{LF}}\|/\|g_{\text{BP}}\|$ is 0.99954 (A) and 1.00062 (B); the median relative L_2 error $\|g_{\text{LF}} - g_{\text{BP}}\|/\|g_{\text{BP}}\|$ is 2.2% (A) and 2.4% (B), with worst-case per-entry rel-L_2 of 6.3% (A) and 5.3% (B). The $\sim 2\%$ rel-L_2 floor is the bf16 numerical-noise scale we expect from per-element matmul reordering and is consistent with the cosine ≥ 0.999 direction-agreement number.

BP-vs-BP control: is the $\sim$2% the bf16 floor or LF-introduced noise? To rule out that the layer-local form adds error beyond the bf16 floor, we compare each bf16 path against a shared fp32 reference. In real arithmetic eq. 1 is an exact reordering of BP, so both compute the identical gradient; the fp32 BP-LoRA gradient is the common "truth" that bf16-BP and bf16-LF each approximate, and the rel-L_2 of each bf16 path to it gives two independent bf16 computations of the same quantity. On the same 10-step-warmup probe (LLaMA-7B, 3 minibatches, 672 per-(layer, module, $A|B$) comparisons), bf16-BP deviates from its *own* fp32 result by median rel-L_2 3.60% (A) / 3.32% (B), while bf16-LF deviates from the same reference by 3.33% (A) / 3.18% (B)—LF is, if anything, marginally *closer* to the real-arithmetic gradient than BP itself, and certainly no farther. The bf16-LF-vs-bf16-BP rel-L_2 on this run is 2.21% (A) / 2.45% (B), reproducing the $\sim$2.2% headline above (the two bf16 paths are closer to each other than to fp32 because they share bf16 forward caches). The control thus establishes that the LF-vs-BP rel-L_2 is the bf16 numerical-noise floor for a gradient of this kind, not excess error from the factorization.

Variant	MMLU (%)	GSM8K (%)
Full LF-LoRA	37.64 ± 0.23	7.63 ± 1.03
randomW0 (P1)	23.10	0.00
A1 sigma'-strip	23.63 ± 1.11	0.15 ± 0.26
A2 s-strip	26.43 ± 0.80	0.00 ± 0.00
A3 rank-1-strip	23.05 ± 0.15	0.53 ± 0.92
A4 gate-strip	23.11 ± 0.28	0.00 ± 0.00
A5 W0-strip	23.39 ± 0.80	1.24 ± 1.29
Approx-A	22.95	0.00
Approx-B	36.67	7.88

Table 5: Per-seed MMLU 5-shot and GSM8K 8-shot CoT for the strip ablations and the two non-zero approximations (LLaMA-7B; strips/full over 3 seeds, APPROX-A/B single-seed), against full LF-LoRA and the randomized-W_0 control. Zeroing any single factor drops MMLU to within 2 pp of chance; among non-zero approximations, frozen-statistics norm (APPROX-A) collapses while pointwise-gate SwiGLU (APPROX-B) trains—so all four factors are needed for the cosine identity, but only three are load-bearing for downstream quality (§5.4).

Method	1k	2k	4k	8k	16k	32k
BP-LoRA	22.1	31.1	49.0	OOM	OOM	OOM
LF-LoRA	18.8	24.7	36.0	59.1	OOM	OOM
QLoRA-BP	14.5	24.6	44.8	OOM	OOM	OOM
QLoRA+LF	14.5	24.6	44.8	OOM	OOM	OOM

Table 6: Peak memory (GB) at varying sequence lengths (LLaMA-7B, batch size 1, rank 8, single 80 GB H100). The activation-graph cost dominates over base-weight compression at long context: QLoRA-BP, which shares BP's activation graph, OOMs at the same seqlen step as BP-LoRA. LF-LoRA completes one extra octave.

C Per-strip ablation per-seed details

Table 5 reports the per-seed numbers behind Figure 4. The standard deviations across the five strips are $\leq$ 1 pp on MMLU, indicating that the strip collapses are not seed-fragile.

D Long-context full peak-memory grid

Table 6 reports the full four-method $\times$ six-seqlen grid summarised in §5.3. Two qualitative regimes emerge: at seqlen $\leq$ 4k, peak memory is dominated by parameters and per-token activations, and the four methods are within a factor of 2 of each other; at seqlen $\geq$ 8k, the activation-graph cost saturates

	LF-LoRA		BP-LoRA		Δ		cos
r	ms/step	GB	ms/step	GB	ms	GB	median
4	387.8	37.57	501.3	51.25	−114	−13.7	0.9996
8	377.7	37.71	505.7	51.45	−128	−13.7	0.9997
16	373.7	38.12	508.9	51.86	−135	−13.7	0.9997
32	373.9	38.75	515.9	52.41	−142	−13.7	0.9997
64	376.0	40.10	547.5	53.77	−171	−13.7	0.9997

Table 7: LoRA-rank sweep on LLaMA-7B (bs=4, seqlen=1024, 1×H100). *cos median* is the median per-(layer, module) gradient cosine between LF-LoRA and BP-LoRA after a 10-step warmup. Across the five ranks (synthetic seqlen 1024) the per-step throughput advantage grows from -114 to -171 ms as BP's fp32 LoRA-path matmuls scale with r, the gradient cosine median holds at ≥ 0.9996, and the memory advantage stays flat at ~ 13.7 GB—the activation-graph saving dominating the linearly-growing adapter.

the 80 GB H100 budget for any method that retains it. LF-LoRA, alone among the four, releases that cost, because the closed-form decomposition of Proposition 1 does not require materializing the full backward graph.

E LoRA-rank sweep

F Extended quality on LLaMA-7B

Method	TQA-MC1 (%)	MT-Bench (1–10)	IFE loose instr. (%)	IFE strict prompt (%)
BP-LoRA	23.22 ± 0.25	3.50 ± 0.27	39.13 ± 0.48	21.93 ± 0.75
LF-LoRA (ours)	23.22 ± 0.28	3.44 ± 0.15	39.05 ± 0.90	21.13 ± 0.53
QLoRA-BP	–	–	38.61 ± 0.86	21.87 ± 0.85
QLoRA+LF	–	–	38.29 ± 1.16	20.76 ± 0.38
Triple-stack (3ep)	–	–	38.53 ± 0.98	20.95 ± 0.53

Table 8: Extended quality on LLaMA-7B: TruthfulQA-MC1, MT-Bench gen (gpt-4o judge, snapshot `gpt-4o-2024-11-20`; 12/1920 Azure rejections excluded), IFEval loose-instruction and strict-prompt. The triple stack (*Triple-stack (3ep)*) is LF + 4-bit NF4 base + gradient checkpointing trained for 3 epochs at peak training memory 9.26 GB.

Table 8 reports TruthfulQA-MC1, MT-Bench, and IFEval for LLaMA-7B. The pattern from the main quality matrix holds: LF-LoRA matches BP-LoRA across all four columns within seed-σ, QLoRA-BP and QLoRA+LF inherit the well-known QLoRA degradation on knowledge tasks, and the triple stack lands within 1 pp of QLoRA-BP at 5× less peak memory.

G IFEval subset and MT-Bench exclusions

Full-suite IFEval. All reported IFEval numbers are over the full 541-prompt benchmark with the complete official instruction-checker set (all 25 `instruction_id` categories: `length_constraints`, `detectable_content`, `detectable_format`, `combination`, `change_case`, `keywords`, `language`, `punctuation`, `startend`). We generate one response per prompt per (method, seed, model) and score strict and loose accuracy following the official protocol; `language:response_language` uses `langdetect`. We report loose-instruction accuracy (IFE-li) in the main tables. An earlier draft scored only the 15 most-common instruction-ids (306/541 prompts fully evaluable); the full-suite numbers are slightly lower in absolute terms (more constraints checked) but preserve the LF-vs-BP ordering, with one consequence noted in §5.2: on Llama-3.1-8B the LF-vs-BP IFE-li gap (1.87 pp) is the single cell that exceeds seed-σ.

MT-Bench exclusions. The MT-Bench judge (gpt-4o, Azure snapshot `gpt-4o-2024-11-20`) rejects 12 of 1920 judge calls ($< 0.7\%$) on three prompt categories: persuasive-essay, villain-roleplay, and lying-prisoners. These are non-random—they are systematically the categories the content filter flags—so the means we report are computed over the remaining 1908 prompts. The exclusion is applied identically to LF-LoRA and BP-LoRA outputs (same prompts dropped for both), so the comparison is paired; the absolute MT-Bench scores are mildly biased upward (the excluded categories include some of the harder edge cases), but the LF-LoRA vs. BP-LoRA delta is not.

H Cross-family and scale: 13B, Qwen2.5, Mistral

The factorization of eq. 1 requires only RMSNorm and a SwiGLU- or pointwise-activation MLP, which all current open transformer families satisfy. Table 4 (main body) reports the memory–throughput summary. Gradient cosine median against BP-LoRA on the same minibatch is 0.9998 (Mistral), 0.9990 (Qwen2.5), and 0.9995 (LLaMA-13B), each across the full set of (layer, module) entries (Mistral: 32×7; Qwen2.5: 28×7; LLaMA-13B: 40×7). The Qwen2.5 raw probe contains a single-minibatch A-matrix value at `layer_0.q_proj` of 0.975 that we report honestly in §5.1; the per-(layer, module) median on the same probe stays ≥ 0.987 across all 196 Qwen entries. The outlier is the A-matrix at the first layer's q-proj, a low-magnitude gradient entry on one of the two probe minibatches;

a small gradient norm makes the cosine numerically sensitive to bf16 rounding. It does not recur at any later layer or on any other family—every other Qwen entry clears 0.987 and the overall median is 0.999—so we read it as a low-magnitude-entry artifact rather than a structural corner-case bug in the factorization.

Downstream quality at the new scales/families. After 3-epoch Alpaca-52k SFT (single seed, $r = 8$), LF-LoRA matches BP-LoRA on MMLU 5-shot, GSM8K 8-shot CoT, and AlpacaEval-2 LC logprob: Qwen2.5-7B-Inst. LF vs. BP: $69.53/69.67$ MMLU, $72.78/73.54$ GSM8K, $81.61/82.36$ AlpacaEval ($|\Delta| \leq 0.76$). Mistral-7B-Inst.: $48.00/48.04$ MMLU, $15.31/17.44$ GSM8K (low-baseline CoT, single-seed variance), $81.37/80.87$ AlpacaEval. At LLaMA-13B, BP-LoRA at bs$=$4/seqlen$=$ 1024 OOMs on an H100 (80 GB) at step 710; `-gradient-checkpointing` fits at 30.5 GB but takes 190.5 min wall ($+76\%$ over LF-LoRA's 108 min at 63.7 GB). Against BP+gc, LF-LoRA reaches MMLU $46.72/45.28$, GSM8K $14.18/13.95$, AlpacaEval $23.85/22.11$; the 0.2–1.7 pp directional advantage is consistent with the gc-axis confound and we do not credit it to the method.

I Ablation strips: per-factor MMLU drop

Figure 4 (main body) summarises the strip-ablation MMLU. Per-strip drops: per-sample rank-one projector (A3, -14.7 pp), SwiGLU multiplicative gate (A4, -14.5 pp), $\sigma' \to 1$ (A1, -14.0 pp), RMSNorm gain s (A2, -11.2 pp), and LoRA-only transport (A5, -14.3 pp). P1 (randomized W_0) is a separate control. The rows are not training algorithms in their own right—each strip is a mis-computed gradient with one factor set to zero or one, and a mis-computed gradient harming training is expected—they show that no single factor can be omitted without collapse.

Plausible non-zero approximations. Beyond zero-substitution strips, two reasonable approximate backward rules on LLaMA-7B / Alpaca-52k / 1-epoch (single seed) split cleanly. APPROX-A (frozen-statistics RMSNorm radius: ρ_{running} as a detached running-stat, severing ρ's dependence on h) collapses outright—training loss 6.62, MMLU 22.95% ($<$ random), GSM8K $0/1319$, full-suite IFEval-li 22.78%—so the per-sample rank-one correction is required as a training-rule property, not only as a cosine-identity property. APPROX-B (treat SwiGLU as pointwise: drop the v_g multiplicative-gate term while keeping $\sigma'(u) \odot v_g$ as a frozen multiplier) trains and matches downstream: loss 1.22 (vs. full LF ~ 1.0), MMLU 36.67%, GSM8K 7.88%, full-suite IFEval-li 38.37%, all within seed-σ of full LF-LoRA at 3-epoch ($37.64/7.63/39.05$). The four factors are not equally load-bearing for downstream quality.

J Memory–throughput accounting

Per-submodule memory. At LLaMA-7B settings (bs$=4$, seqlen$=1024$, hidden$=4096$), an attention-path (B, T, H) bf16 tensor occupies ~ 32 MB; the gate/up branches project to intermediate dim 11008 and a $(B, T, 11008)$ tensor occupies ~ 86 MB. BP-LoRA's autograd graph stores the linear input x, the post-norm activation y, and (for SwiGLU) the gate and up branches $\{u, v_g\}$. LF-LoRA saves only x, $\sigma'(u)$ (or $\{\varphi(u), \varphi'(u), u, v_g\}$ for SwiGLU), and the scalar gain $s = \gamma/\rho$, releasing per-op intermediates immediately after the layer-local update fires. Peak memory is a max-over-time, so the per-step elision measured at the training sequence length (Table 2, 13.7 GB reserved at seqlen 1024) is the attribution to the layer-local form; because activation memory—and hence the elision—scales with sequence length, a short cached-batch microbench understates it (a seqlen-157 batch reproduces only 1.9 GB).

Step-time accounting. At seqlen 1024, eager BP-LoRA $\to$ eager LF-LoRA saves 128 ms/step ($505.7 \to 377.7$ ms); compile-BP-LoRA $\to$ compile-LF-LoRA saves a further 56 ms ($353.3 \to 296.8$ ms at default mode). A Kineto trace attributes the eager delta and corrects a framing in earlier drafts: the step is GPU-bound (GPU-busy is $\approx 90\%$ of wall, and the custom `Functions` raise the kernel-launch count by ~ 510/step rather than lowering it), so the win is reduced GPU *work*, not reduced dispatch overhead. Two effects dominate: a 70% drop in `aten::copy_` time ($111 \to 33$ ms/step)—the per-op intermediate activations the standard backward materializes and reloads—and a $\sim 14\%$ drop in matmul time, traced to the fp32 LoRA-path GEMMs a stock `peft` adapter issues (`sm80_..f32f32` kernels, ~ 42 ms/step) but the bf16 closed form does not. What remains in the step is forward matmuls (dominated by the frozen 7B base), the non-adapted backward graph (attention softmax, RoPE, embedding/`lm_head`), and AdamW.

Decomposing the gain by mechanism. Table 3 (main body) attributes the gains: MLP-only-checkpointed BP-LoRA isolates the memory effect with a 17% throughput penalty; fused-RMSNorm BP-LoRA isolates per-operator kernel fusion (the +10% row); `torch.compile(BP-LoRA)` fuses the full backward subgraph (+43%); and `torch.compile(LF-LoRA)` adds another +19% throughput and 7.3 GB lower memory over compile-BP-LoRA, since the layer-local form addresses a cost (the materialization and copying of per-op intermediate activations, plus the fp32 LoRA-path matmuls) that Inductor itself does not remove.

Dynamo integration. The three custom `autograd.Functions` per sub-block are opaque to Dynamo, which graph-breaks around each `Function.apply`. The custom backward therefore stays in eager while Inductor compiles the surrounding paths (attention softmax, RoPE, residual sums, embedding, `lm_head`, AdamW). The 19–24% throughput gap between the two compiled methods is, by the Kineto trace above, the matmul-time and intermediate-copy savings the custom Function realizes even after Inductor fuses the surrounding paths; the residual non-adapted graph (attention softmax, RoPE, `lm_head`) is untouched by either.

K Reproducibility

Implementation. The training entry point is a single Python script (`experiments/train_pc_lora.py`, $\approx$ 1700 LOC) with `-method` $\in$ {`bp-lora`, `lf-lora`, `lf-lora-randomW0`, `fa-lillicrap`, `lora-fa`, `lora-pro`}, `-strip` $\in$ {`none`, `sigma`, `s`, `rank1`, `gate`, `W0`}, `-quant` $\in$ {`none`, `4bit`}, and `-gradient-checkpointing`. Evaluation scripts (one per task) live alongside.

Hyperparameters. AdamW; learning rate 2×10^{-4}; batch size 4 with gradient accumulation 8 (effective 32); sequence length 1024; bf16; 3 epochs of Alpaca-52k; LoRA rank 8 on $\{q, k, v, o, \text{gate}, \text{up}, \text{down}\}$-proj at all 32 (LLaMA-7B) / 32 (Llama-3.1-8B) decoder layers. All methods share the same outer optimizer and learning rate to keep the comparison protocol-matched.

Hardware + libraries. 1×H100-80GB per job, 32 GB system RAM, on an HPC cluster with H100 GPUs. All LF-LoRA, BP-LoRA, QLoRA, and ablation runs use one shared environment: PyTorch 2.5.1, transformers 4.46.3, peft, accelerate, bitsandbytes 0.49.2. The Unsloth head-to-head (Table 2) is the sole exception: Unsloth pins its own stack and applies import-time monkey-patches that would corrupt the shared environment, so it runs in an isolated `.venv-unsloth` (PyTorch 2.10.0, Triton 3.6.0, xformers 0.0.35) on the same H100 node. The two stacks therefore differ, which is one more reason we read the Unsloth row as indicative rather than a matched peak-to-peak delta (§2). End-to-end compute including failed and exploratory runs is approximately 100 H100-GPU-h.

Datasets and licenses. Alpaca-52k (`tatsu-lab/alpaca`, CC-BY-NC 4.0); MMLU (`cais/mmlu`, MIT); GSM8K (`gsm8k` 'main', MIT); TruthfulQA (`truthful_qa` MC1, Apache-2.0); MT-Bench (`lmsys/mt_bench_human_judgments`, Apache-2.0); IFEval (`google/IFEval`, Apache-2.0). Base models: `huggyllama/llama-7b` (Meta LLaMA research license); `NousResearch/Meta-Llama-3.1-8B-Instruct` (Meta Llama 3.1 community license redistribution).