PRECISION-AWARE LEARNING RATE TRANSFER ACROSS BLOCK COUNT IN LLM POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

The post-training of pretrained large language models, including supervised fine-tuning (SFT) and direct preference optimization (DPO), is very sensitive to the learning rate, and tuning it through repeated runs is costly in both computation and memory. In this work, we propose TipSweep, a method to transfer the learning rate across the number of trainable transformer blocks. To do so efficiently, we search for the best learning rate by training only the last k_0 blocks, and show how to scale the selected rate for a subsequent set of k blocks. Across SFT and DPO experiments with models ranging from 1 to 7 billion parameters, we observe that the optimal learning rate follows precision-dependent scaling rules: the optimal rate scales approximately as k^{-1} with float32 (FP32) master weights, but as $k^{-1/2}$ when updates are applied directly to bfloat16 (BF16) weights without a master copy. We explain both rules using an analytical model that accounts for update rounding, under assumptions supported by profiling experiments. Finally, we develop a prototype that demonstrates the feasibility of learning rate search for a 6.7-billion-parameter model on a single consumer GPU with 16 GiB of memory.

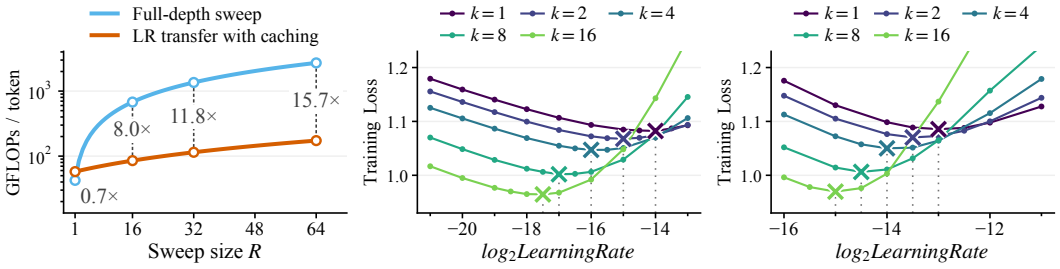
1 INTRODUCTION

Pretrained large language models (LLMs) (Yang et al., 2025) are publicly available for adaptation through offline post-training on fixed datasets, including supervised finetuning (SFT) (Ouyang et al., 2022) and direct preference optimization (DPO) (Rafailov et al., 2023). However, post-training performance is sensitive to the learning rate, and selecting it is costly because each candidate value requires a separate run (Lambert et al., 2025). For pretraining, the maximal update parameterization (μP) provides a method for efficient learning rate search, by tuning on a narrower proxy and transferring the selected rate to a wider target model (Yang et al., 2021). However, this approach cannot be used with pretrained checkpoints since model width is fixed and cannot be changed during post-training. This motivates finding a transfer dimension for LLM post-training that leaves the checkpoint architecture fixed.

In this work, we propose TipSweep, a method that transfers learning rate across the number of trainable transformer blocks, which is a suitable transfer axis for post-training. Specifically, we search for the optimal learning rate by training only the last k_0 of the model's L transformer blocks. The selected rate is then scaled for a subsequent set of k trainable blocks. A particularly useful case is to do a full-depth run in which all transformer blocks are trainable, namely $k = L$. For $k_0 < L$, restricting backpropagation through transformer blocks to the last k_0 reduces the computation and GPU memory requirements relative to its full-depth counterpart. In this setting, we cache the frozen prefix's outputs for reuse across multiple trials to reduce computation. As shown in Figure 1a, searching with a last-block proxy ($k_0 = 1$) reduces the total FLOPs of a 16-trial learning rate sweep on Llama-2-7B by a factor of approximately 8 relative to running all 16 trials purely at full depth. This reduction includes the cache construction and the subsequent full-depth run.

Our approach hinges on one question: Does the optimal learning rate scale predictably with the number of trainable blocks k? As shown in Figures 1b and 1c, we observe that the optimal learning rate roughly follows a power law in k. Surprisingly, the exponent depends on the precision used to accumulate parameter updates. For the same model and dataset, the optimal learning rate scales as k^{-1} when updates are accumulated in float32 (FP32) master weights, while it scales as $k^{-1/2}$ when updates are applied directly to bfloat16 (BF16) weights without a float32 master copy.

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



(a) Analytical FLOPs (b) Float32 weights (c) BFloat16 weights

Figure 1: a) Analytical FLOPs per target token against sweep size R for a full-depth sweep and for learning rate (LR) transfer with caching. b) and c) Learning rate sweeps on Llama-3.2-1B (Tülu 3) with the last k blocks trainable under FP32 master-weight and direct BF16 updates, respectively.

To understand these patterns, we provide a theoretical analysis that takes parameter-update rounding into account. In this analysis, we define a stable loss-reduction criterion that characterizes how the optimal learning rate should behave as the number of trainable blocks increases. Under assumptions supported by profiling in the training regimes we study, we derive exact scaling rules that match the observations for both FP32 and BF16 updates.

We conduct experiments to verify these scaling rules across SFT and DPO, 1B–7B models, and multiple post-training datasets. Furthermore, we implement a prototype of the proposed learning rate search strategy and show that it can tune the learning rate of a 6.7B model on a consumer GPU with approximately 16 GiB of available memory in reasonable time. This makes it possible to delegate the search trials to widely available consumer GPUs and reserve scarce high-end GPUs only for the final full-depth run. Our contributions are as follows:

- **Efficient learning rate search.** We propose learning rate transfer across the number of trainable transformer blocks, combined with frozen-prefix feature caching, to reduce the computation and memory required to select a learning rate for LLM offline post-training.
- **Precision-dependent scaling.** We identify that the learning rate scaling with trainable block count k follows scaling rules k^{-1} under FP32 updates and $k^{-1/2}$ under BF16 updates, supported by SFT and DPO experiments.
- **Theoretical analysis.** We provide a theoretical analysis explaining these scaling rules, based on a stable loss reduction criterion.
- **Empirical results and case study.** We conduct extensive experiments to examine the observed scaling rules. We implement and evaluate a prototype that enables learning rate searches for a 6.7B model on a consumer GPU with approximately 16 GiB of available memory.

2 Related Work

LLM post-training. Post-training teaches a pretrained LLM to follow instructions and improves its reasoning, typically in three stages (Qwen Team, 2024; Lambert et al., 2025; Walsh et al., 2025; Guo et al., 2025; OLMo Team et al., 2025). Supervised finetuning (SFT) trains the model to imitate reference responses (Ouyang et al., 2022; Zhou et al., 2023; Chung et al., 2024; Chu et al., 2025). Direct preference optimization (DPO) trains it to favor chosen over rejected responses (Rafailov et al., 2023; Meng et al., 2024; Liu et al., 2024b). Reinforcement learning with verifiable rewards (RLVR) trains it on its own responses, rewarding those that pass programmatic checks (Shao et al., 2024; Yu et al., 2025; Liu et al., 2026). Each stage requires tuning the learning rate, typically through costly full-depth runs. We instead search on a proxy with only the last few blocks trainable and scale the selected rate for a subsequent full-depth run.

Mixed-precision training. Mixed-precision training uses 16-bit computation but accumulates updates in FP32 master weights (Micikevicius et al., 2018; Rajbhandari et al., 2020). BF16's FP32-like dynamic range (Burgess et al., 2019; Kalamkar et al., 2019) allows dropping the master copy to save

memory (Yu et al., 2024; Ozkara et al., 2025), but round-to-nearest can then discard small updates at small learning rates (Zamirai et al., 2020; Rae et al., 2021). In this work, we observe that this rounding also changes learning rate scaling with the number of trainable blocks.

Hyperparameter transfer. Hyperparameter transfer characterizes how optimal hyperparameters change along a dimension of scale, so they can be tuned on a small proxy and transferred to the target model. Early works scale across width in pretraining (Yang & Hu, 2021; Yang et al., 2021; 2023), including with FP8 arithmetic (Blake et al., 2025). Depth extensions follow the same recipe (Hayou et al., 2019; 2021; He et al., 2023; Hayou & Yang, 2023; Noci et al., 2023; Yang et al., 2024; Bordelon et al., 2024). Both lines of work derive their scaling rules from a prescribed random initialization, and are stated for models trained from it. Existing post-training works carry that assumption over to LoRA initialization and analyze a single adapted layer (Hayou et al., 2024a;b; Li et al., 2025; Chen et al., 2026), leaving depth out of scope. We analyze scaling across the number of trainable blocks when tuning the pretrained weights directly, whose distribution is set by a pretraining run we cannot observe. We therefore identify the rules empirically and provide a theoretical explanation for their dependence on update precision.

3 Methodology

3.1 Precision-aware learning rate transfer

For a pretrained checkpoint with L transformer blocks, we finetune the model with R distinct learning rates on a grid with only the last k_0 blocks trainable, where $1 \le k_0 < L$. We train all parameters in these blocks, including those of the normalization modules, and freeze the rest of the model. Let $\widehat{\eta}_{k_0}$ be the optimal learning rate, i.e. the learning rate with smallest training loss. To finetune k blocks, TipSweep gives precision-dependent scaling rules for the optimal learning rate $\widehat{\eta}_k$:

$$\widehat{\eta}_k \approx \begin{cases} (k_0/k) \cdot \widehat{\eta}_{k_0}, & \text{FP32 master-weight updates,} \\ \sqrt{k_0/k} \cdot \widehat{\eta}_{k_0}, & \text{direct BF16 updates without an FP32 master copy.} \end{cases}$$

We initialize from the same checkpoint and train the last k transformer blocks at $\widehat{\eta}_k$ under the same update precision, freezing all other parameters. Figure 2 shows a good match between these scaling rules and empirical observations, and additional results are provided in Section 5. Using TipSweep, we can tune the learning rate with small k_0, and transfer it to the full-depth run ($k = L$).

Feature caching. When searching with only the last k_0 blocks trainable, backpropagation passes through the frozen language model (LM) head to the trainable blocks but does not enter the frozen prefix. Without feature caching, however, each trial still recomputes the frozen prefix on every forward pass. Feature caching removes this repeated computation. For a fixed tokenized dataset with deterministic prefix evaluation, we first pass the dataset through blocks $1, \dots, L - k_0$ in inference mode and cache the outputs of block $L - k_0$ on disk, a stage we call *cache build*. Each proxy trial (a run on a single learning rate value from the grid) then loads these features and performs forward and backward propagation through the last k_0 blocks and LM head, which we call a *cached proxy trial*. The full pipeline is illustrated in Figure 20.

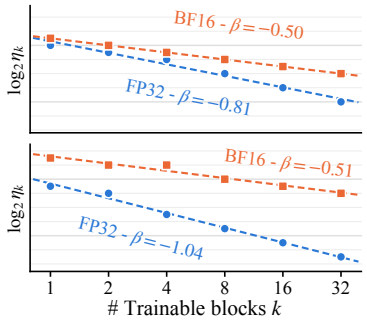
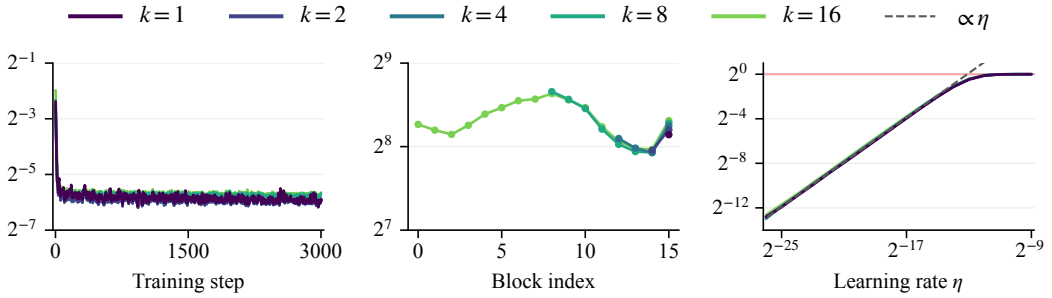


Figure 2: $\log_2 \widehat{\eta}_k$ vs. $\log_2 k$. Upper: DSCoder. Lower: OLMo-2.

3.2 Analytical FLOPs

To estimate computational cost, we calculate FLOPs over a single epoch, counting dense matrix products in forward and backward propagation at two FLOPs per multiply–add (Kaplan et al., 2020; Narayanan et al., 2021; Hoffmann et al., 2022). We omit non-matrix operations, optimizer updates, and any activation recomputation. Appendix D gives the detailed FLOP analysis.

Let $c_{\text{fwd,blk}}$ denote the forward FLOPs per token of one transformer block. A backward pass through a trainable block costs $2\, c_{\text{fwd,blk}}$, and the frozen LM head costs $\gamma\, c_{\text{fwd,blk}}$ per pass, forward or



(a) $\eta\,\widetilde{D}_k(\eta)$ (b) $d_l = \langle \nabla_{\boldsymbol{\theta}_l} \mathcal{L}, \boldsymbol{u}_l \rangle$ (c) $\phi_k(\eta) = \widetilde{D}_k(\eta)/D_k$

Figure 3: Profiling results for SFT of Llama-3.2-1B on Tülu 3 with FP32 weights. Each curve corresponds to a run with the last k blocks trainable at $\eta = 2^{-14}/k$. a) Realized first-order loss reduction against training step. b) Geometric mean over training steps of the per-block first-order loss contribution $\widetilde{d}_l(\eta)$, which equals d_l under the FP32 precision, against block index. c) BF16 update factor $\phi_k(\eta)$ against the probe learning rate η at step 3000. Complete results in Appendix A.

backward, for some $\gamma > 0$. Hence, the per-token FLOPs of each procedure are

Full-depth run:	$(3L + 2\gamma)\, c_{\text{fwd,blk}}$,	Uncached proxy trial:	$(L + 2k_0 + 2\gamma)\, c_{\text{fwd,blk}}$,
Cache build:	$(L - k_0)\, c_{\text{fwd,blk}}$,	Cached proxy trial:	$(3k_0 + 2\gamma)\, c_{\text{fwd,blk}}$.

Without caching, a proxy trial saves less than a factor of three, $(3L + 2\gamma)/(L + 2k_0 + 2\gamma) < 3$, because the frozen prefix still executes in every trial. We now compare the complete cost of using TipSweep to perform learning rate transfer with caching against a full-depth sweep. We account for the cache construction, R proxy trials, and the final full-depth run. For the full-depth sweep, since every trial trains the full model, the best of its R trials can be adopted directly without a final retraining. The resulting per-token FLOP ratio of the full-depth sweep to TipSweep is

$$\rho(R, k_0) = \frac{R(3L + 2\gamma)}{\underbrace{(L - k_0)}_{\text{cache build}} + \underbrace{R(3k_0 + 2\gamma)}_{\text{proxy trials}} + \underbrace{(3L + 2\gamma)}_{\text{full-depth run}}}. \tag{1}$$

The ratio in Equation 1 monotonically increases with R and decreases with k_0, but satisfies

$$\rho(R, k_0) < \min\left\{R, \frac{3L + 2\gamma}{3k_0 + 2\gamma}\right\} \leq \min\left\{R, \frac{L}{k_0}\right\}.$$

The two bounds correspond to two regimes of sweep grid size R, distinguished by which term dominates the denominator of Equation 1. For small sweeps, the one-time costs of cache build and the final full-depth run dominate. Each additional proxy trial then costs only a small fraction of a full-depth trial, so increasing R amortizes the one-time costs and raises ρ. For large sweeps, the proxy trials dominate and ρ approaches the per-trial ratio $(3L + 2\gamma)/(3k_0 + 2\gamma)$, the cost of a full-depth trial relative to a cached proxy trial. Further increasing R yields diminishing gains, whereas reducing k_0 raises this limit. We instantiate this cost comparison on Llama-2 (Touvron et al., 2023) with $k_0 = 1$. Figure 1a reports the Llama-2-7B configuration, and Figure 19 includes more cases.

4 Learning Rate Scaling under Update Rounding

In this section, we provide a theoretical justification for the observed scaling rules. The analysis is carried out at a physics level of rigor, making the modeling assumptions explicit and deferring other technical details to the appendix. The profiling during training supports all the assumptions, and representative results are shown in Figure 3. Due to space constraints, we defer the proofs, profiling setup details, and complete profiling results to Appendix A.

Consider a transformer with L blocks, where $\boldsymbol{\theta}_l$ denotes the parameters of block l. Write $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_L)$ for the collection of block parameters and $\mathcal{L}(\boldsymbol{\theta})$ for the loss. We finetune the last k

blocks, $\mathcal{A}_k = \{L-k+1, \ldots, L\}$, with all remaining parameters frozen. For $l \in \mathcal{A}_k$, let $\boldsymbol{u}_l$ denote the update direction produced by Adam. With learning rate η, the stored parameters are updated as

$$\boldsymbol{\theta}_l^+ = Q(\boldsymbol{\theta}_l - \eta \boldsymbol{u}_l), \qquad Q(\boldsymbol{\theta}_l) = \boldsymbol{\theta}_l, \qquad l \in \mathcal{A}_k,$$

where Q denotes the storage-precision rounding operator. Under FP32 master-weight accumulation, Q is the identity function. Under direct BF16 updates without an FP32 master copy, Q rounds each coordinate to the nearest representable BF16 value.

We define the *nominal* first-order loss contribution of block l, and its sum over trainable blocks, as

$$d_l = \langle \nabla_{\boldsymbol{\theta}_l} \mathcal{L}, \boldsymbol{u}_l \rangle, \qquad D_k = \sum\nolimits_{l \in \mathcal{A}_k} d_l.$$

With the realized direction $\widetilde{\boldsymbol{u}}_l(\eta) = (\boldsymbol{\theta}_l - \boldsymbol{\theta}_l^+)/\eta$, we define their *realized* counterparts as

$$\widetilde{d}_l(\eta) = \langle \nabla_{\boldsymbol{\theta}_l} \mathcal{L}, \widetilde{\boldsymbol{u}}_l(\eta) \rangle, \qquad \widetilde{D}_k(\eta) = \sum\nolimits_{l \in \mathcal{A}_k} \widetilde{d}_l(\eta).$$

For $D_k > 0$, we define the *update factor* $\phi_k(\eta) = \widetilde{D}_k(\eta)/D_k$. This factor is the ratio of the realized first-order loss reduction to its nominal value and equals one under the FP32 precision.

A first order Taylor gives the following approximation to loss reduction after one training step

$$\Delta\mathcal{L}_k(\eta) := \mathcal{L}(\boldsymbol{\theta}^+) - \mathcal{L}(\boldsymbol{\theta}) = -\eta \widetilde{D}_k(\eta) + \mathcal{O}(\eta^2) = -\eta D_k \phi_k(\eta) + \mathcal{O}(\eta^2).$$

For our analysis, we omit the term $\mathcal{O}(\eta^2)$ and focus on the first order term. The next assumption is fundamental to our analysis and assumes that optimal learning rates are associated with stable loss reduction. Throughout, we use the asymptotic notation $\Theta(\cdot)$ to denotes two-sided bounds with positive constants independent of k and L.[1]

Assumption 1 (Stable loss reduction)**.** When only the last k blocks are trainable, the optimal learning rate η_k satisfies

$$-\Delta\mathcal{L}_k(\eta_k) = \Theta(1). \tag{2}$$

Under the first-order approximation, Equation 2 is equivalent to

$$\eta_k D_k \, \phi_k(\eta_k) = \Theta(1). \tag{3}$$

Intuitively, the optimal learning rate will scale with k in a way that makes the loss reduction neither vanish nor explode. This assumption is the loss-level counterpart of a well-established criterion for learning rate transfer, which chooses the learning rate so that each update changes the model's features by $\Theta(1)$ as the model size scales up (Yang & Hu, 2021; Hayou et al., 2024a;b). Figure 3a shows that the per-step values of $\eta \widetilde{D}_k(\eta)$ at the empirically tuned learning rates remain on a comparable scale across k during training, supporting this assumption.

Assumption 2 (Comparable per-block contributions)**.** The nominal contributions satisfy

$$d_l = \Theta(1), \qquad l \in \mathcal{A}_k, \tag{4}$$

uniformly over trainable blocks in the regime considered.

Assumption 2 indicates that for each block, as we scale k, the contribution of block k to the loss reduction remains stable (not exploding and not vanishing). Figure 3b provides empirical support for this assumption, showing that the geometric mean of d_l over training steps is approximately invariant to k within each block and comparable across blocks.

Assumption 3 (Dependence of the update factor on the learning rate)**.** For a fixed $\alpha \geq 0$, the update factor satisfies $\phi_k(\eta) = \Theta(\eta^\alpha)$, uniformly over the learning rate regime considered.

Under FP32, $\phi_k(\eta) = 1$, corresponding to $\alpha = 0$. For BF16, we conjecture that a fraction proportional to η of the $\tilde{d}_l(\eta)$ updates are zero, while the others are equal to d_l. We justify this claim in Appendix A.3 and show it implies that $\phi_k(\eta) = \Theta(\eta)$, and therefore $\alpha = 1$. The empirical results in in Figure 3c, show that the curves for different k nearly coincide and grow approximately linearly in η before saturation to a constant. We conjecture that the optimal BF16 learning rates lie in this linear regime (in η) because post-training generally requires small adaptations to the pretrained weights, whereas larger learning rates can destabilize training.

[1]Precisely, we say for two quantities A and B that $B = \Theta(A)$ if there exists some constants $c_1, c_2 > 0$ independent of k and L such that $B/A \in [c_1, c_2]$. In other words, the notation $B = \Theta(A)$ means the two quantities have roughly the same order independently of k and L.

Proposition 4.1 (Learning rate scaling for general α (informal))**.** *Under the first-order approximation and Assumptions 1–3, the optimal learning rate η_k satisfies*

$$\eta_k = \Theta\Big(k^{-1/(1+\alpha)}\Big). \tag{5}$$

Corollary 4.2 (FP32 and BF16 scaling)**.** *Under the conditions of Proposition 4.1, FP32 master-weight updates have $\phi_k(\eta) = 1$, corresponding to $\alpha = 0$, whereas direct BF16 updates with $\phi_k(\eta) = \Theta(\eta)$ have $\alpha = 1$. Consequently,*

$$\eta_k = \begin{cases} \Theta(k^{-1}), & \textit{FP32 master-weight updates}, \\ \Theta(k^{-1/2}), & \textit{direct BF16 updates}. \end{cases} \tag{6}$$

Intuitively, for BF16 updates in the learning rate regime considered, reducing η not only shrinks the parameter updates per step but also causes more of them to be suppressed by rounding, making the realized update sparser. This suppression contributes a further factor of η in ϕ_k (see Appendix A.3), changing the realized first-order loss reduction from $\Theta(k\eta)$ to $\Theta(k\eta^2)$. Maintaining a stable loss reduction therefore changes the learning rate scaling from k^{-1} to $k^{-1/2}$.

5 Experiments

We test TipSweep learning rate scaling rules with the number of trainable blocks k and measure the GPU-memory and runtime costs of learning rate transfer with caching.[2] Our experiments span multiple model families and two post-training regimes: supervised finetuning (SFT) and direct preference optimization (DPO). Through these experiments, we aim to address three points:

1. Validate TipSweep scaling rules across post-training regimes and model families.
2. Study the impact of the placement of the trainable blocks on optimal learning rate scaling.
3. Quantify the GPU-memory and runtime costs associated with TipSweep.

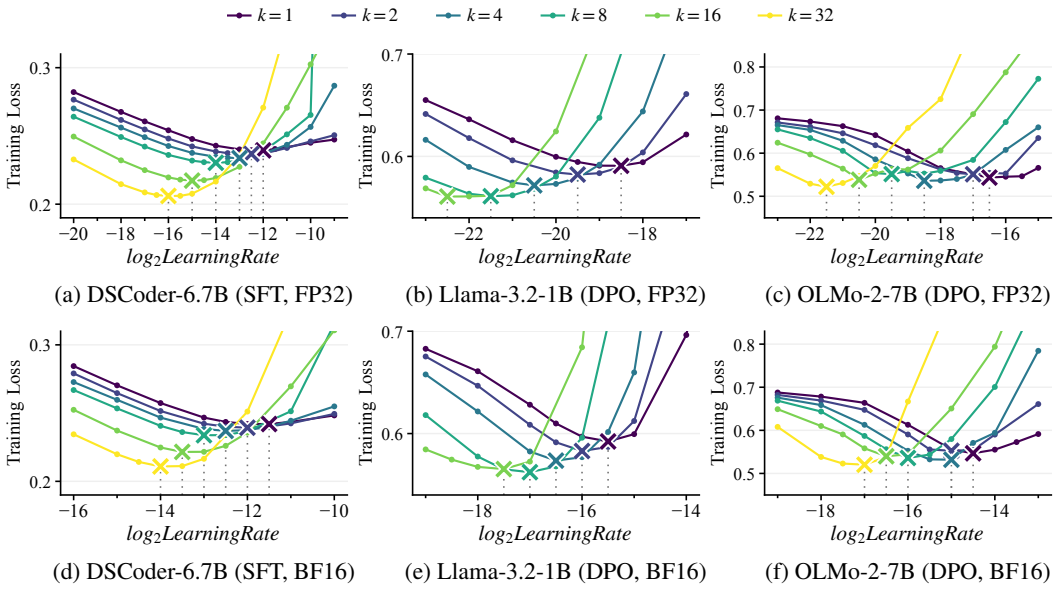
Shared post-training protocol. Unless otherwise specified, we train with a maximum sequence length of 1024 tokens, using the AdamW optimizer with zero weight decay and global-norm gradient clipping at 1.0. We vary the learning rate schedule across settings, using either a constant learning rate or cosine annealing with linear warmup, to show that the scaling rules hold across schedules. For each setting, we independently identify the optimal learning rate under each of two criteria: (i) the final training loss, smoothed with an exponential moving average (EMA), and (ii) a task-specific validation metric. We first sweep η on a coarse $\log_2$-spaced grid with multiplicative steps of $2\times$. We then refine around the coarse-grid rate with the lowest final EMA-smoothed training loss, evaluating two additional rates at offsets of ± 0.5 in $\log_2 \eta$. This local refinement improves resolution around the optimum without the cost of evaluating a uniformly finer grid. We report training-loss-based sweeps in the main text and defer the other results to Appendix C. Setup details are in Appendix B.

Interpreting the identified learning rates. The reported optimum for each metric is the best value among the evaluated grid points. Near a flat optimum, neighboring learning rates can perform similarly, so run-to-run noise may determine which one is selected. Finer grids could make selection more sensitive to these near-ties, and update precision might also affect the flatness of the performance curve. We therefore assess qualitative agreement with the prescribed scaling rules rather than require an exact numerical match.

5.1 Precision-Dependent Scaling across the Number of Trainable Blocks

Setup. For SFT tasks, we train Llama-3.2-1B (Grattafiori et al., 2024) on Tülu 3 (Lambert et al., 2025) and DeepSeek-Coder-6.7B-Base (Guo et al., 2024) on Magicoder-OSS-Instruct (Wei et al., 2024). For DPO tasks, we train Llama-3.2-1B-Instruct on UltraFeedback (Cui et al., 2023) using the standard DPO loss with $\beta = 0.1$. We also finetune OLMo-2-1124-7B-SFT on its official preference mixture (Walsh et al., 2025) using a length-normalized DPO loss with $\beta = 5.0$. This loss normalizes each response's policy–reference log-ratio by its length before applying the log-sigmoid. For this setting, we increase the maximum sequence length to 2048 tokens. Across the four settings, model depth is $L = 16$ for the Llama models and $L = 32$ for DeepSeek-Coder and the OLMo model.

[2] Source code is available in the supplementary materials.



(a) DSCoder-6.7B (SFT, FP32) (b) Llama-3.2-1B (DPO, FP32) (c) OLMo-2-7B (DPO, FP32)

(d) DSCoder-6.7B (SFT, BF16) (e) Llama-3.2-1B (DPO, BF16) (f) OLMo-2-7B (DPO, BF16)

Figure 4: Learning rate sweeps with FP32 master-weight (top) and direct BF16 updates (bottom). (a, d) SFT of DeepSeek-Coder-6.7B-Base on Magicoder-OSS-Instruct; (b, e) DPO of Llama-3.2-1B-Instruct on UltraFeedback; (c, f) DPO of OLMo-2-1124-7B-SFT on its official preference mixture. Each curve shows final training loss (EMA-smoothed) versus $\log_2$ learning rate with only the last k blocks trainable. Crosses and dotted lines mark each curve's optimal learning rate.

Results. The learning rate that minimizes the final training loss scales approximately as k^{-1} under FP32 master-weight updates (Figures 1b and 4a–4c) and as $k^{-1/2}$ under direct BF16 updates (Figures 1c and 4d–4f). Figure 14 quantifies both trends by plotting the optimal learning rate against k and reporting the scaling exponents from log–log fits. When finetuning more blocks, the loss also rises more sharply once the learning rate exceeds the optimum, supporting the need to rescale the learning rate rather than reuse the proxy rate unchanged. Update precision affects the results in two further ways. First, for a given setting and k, the optimal learning rate is lower under FP32 than under BF16, especially for DPO tasks. This is consistent with update rounding: BF16 suppresses small parameter increments, so larger learning rates are needed to produce effective weight changes. Second, the FP32 curves are flatter near their minima, particularly for the 7B-scale models at small k, so neighboring learning rates yield similar losses and the selected grid point becomes more sensitive to run-to-run noise. More results are in Appendix C.1, including ablations finetuning only self-attention or MLP modules in each trainable block.

5.2 Effect of Trainable-Block Placement

Setup. Our analysis extends to any subset of k trainable blocks under the same assumptions, since the scaling argument sums comparable per-block contributions and does not depend on block position. To test this prediction empirically, we compare three ways of choosing which k blocks to train: the last k blocks (*last-k*), the first k blocks (*first-k*), and a random subset of k blocks (*random-k*). For *random-k*, we sample 10 distinct subsets for each $k < L$. In this part, we finetune Mistral-7B-v0.1 ($L = 32$) (Jiang et al., 2023) on Magicoder-OSS-Instruct, under direct BF16 weight updates.

Results. Figure 5 shows that, for each k, the three placement strategies reach their minimum training loss at roughly the same learning rate. As k increases from 1 to 32, the optimal rate scales approximately as $k^{-1/2}$, consistent with the scaling rules for direct BF16 updates. Together, these observations suggest that the optimal learning rate depends on the number of trainable blocks but not on their placement. Although the optimal rate is nearly the same across placements, the final training loss varies substantially. Specifically, random-k generally yields the lowest loss while last-k gives the highest. The spread of final training loss across random subsets also tends to widen as k decreases. Figure 18 shows the positions of the trainable blocks within each sampled subset, with

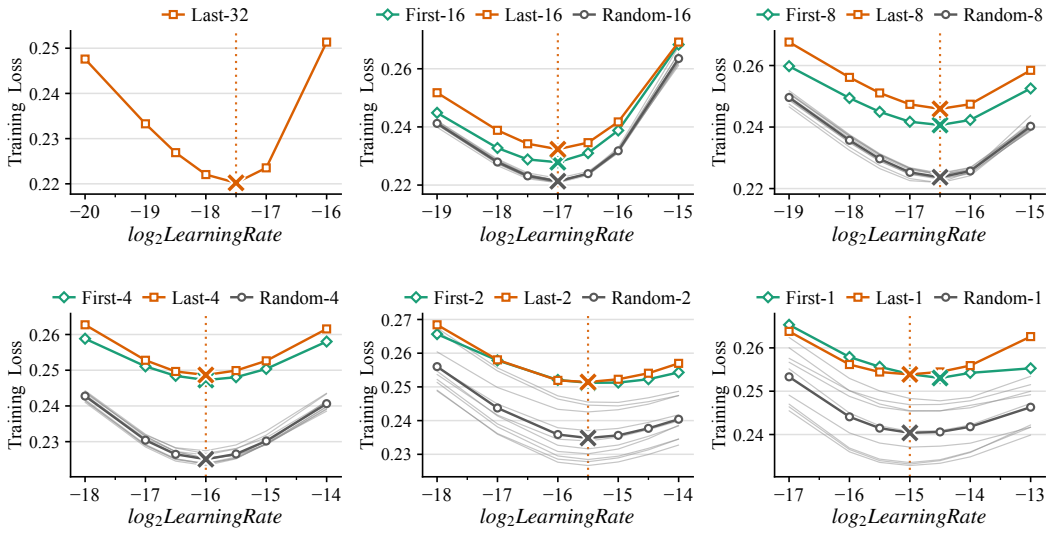


Figure 5: Learning rate sweeps across trainable-block placements. Each panel shows final training loss (EMA-smoothed) versus $\log_2$ learning rate for SFT of Mistral-7B-v0.1 under direct BF16 updates, with $k = 32, 16, \ldots, 1$ trainable blocks. For random-k, thin gray curves show the sampled subsets for each k, and the bold curve denotes their average. Crosses mark the optimal learning rate.

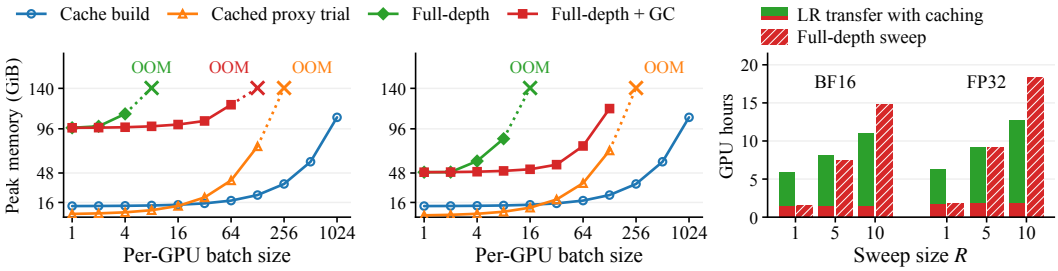
subsets ordered by final training loss. For $1 < k$, we find that subsets whose trainable blocks are spread more evenly across the model tend to obtain lower loss. For $k = 1$, training a block near the middle of the model tends to yield better performance than training one near the input or output.

5.3 Case Study

Setup. To measure the peak GPU memory and runtime costs, we implement a prototype of learning rate transfer with caching and evaluate it by finetuning DeepSeek-Coder-6.7B-Base on Magicoder-OSS-Instruct. Following Section 3.1, the prototype runs in two stages: a cache build, followed by cached proxy trials. We set $k_0 = 1$, so each trial trains only the last block, and compare against a full-depth sweep that trains all blocks at every candidate rate. Details are in Appendix E.

Peak allocated GPU memory. Figures 6a and 6b report the peak allocated GPU memory as the per-GPU batch size grows from 1 to 1024. To evaluate worst-case memory use, we construct training batches in which every sequence contains exactly 1024 tokens. At small batch sizes, both stages of our method use substantially less memory than full-depth training, particularly with FP32 master weights. This is because full-depth training stores gradients and optimizer states for all L blocks even at a batch size of 1, whereas the cache build requires neither and the proxy trials store them for one block. As the batch size grows, gradient checkpointing slows the growth of memory in full-depth training at the cost of extra recomputation. Our proxy trials do not use checkpointing, so their memory curve is steeper. Nonetheless, they use less memory than a checkpointed full-depth run up to a batch size of 128 under both precisions.

Running time on a consumer GPU. We deploy the prototype on a single NVIDIA RTX 5060 Ti with 15.45 GiB of available GPU memory and measure its runtime. Since full-depth training needs more memory than this GPU provides even at a batch size of 1, we run it on NVIDIA H200 GPUs, for the final full-depth run of our method and for every trial of the full-depth sweep. Figure 6c reports the resulting GPU hours by hardware tier against the sweep size R. With $R = 5$, LR transfer with caching uses roughly the same number of GPU hours as the full-depth sweep under both precisions. However, it spends about 80% of these hours on a consumer GPU with lower capacity, searching for the learning rate on a small proxy, and employs the H200 only for the final full-depth run, whereas the full-depth sweep occupies the high-capacity GPU throughout. As R



(a) Peak GPU Memory, FP32 (b) Peak GPU Memory, BF16 (c) GPU hours vs. sweep size

Figure 6: a) and b) depict peak allocated GPU memory against per-GPU batch size under FP32 and BF16 weights, respectively; OOM marks out-of-memory errors. c) GPU hours for a search over R candidate learning rates under each precision: cache build and R proxy trials on an RTX 5060 Ti (green) plus one full-depth run on H200 (red), versus R full-depth runs on H200 (red, hatched).

Table 1: Pass@1 (%, higher is better) on HumanEval and HumanEval+ for DeepSeek-Coder-6.7B-Base and its finetuned variants. Full-depth trains all 32 transformer blocks and Full-parameter also trains the embeddings and LM head. Both are trained at the learning rate transferred from a $k_0 = 1$ proxy search. The Base model and Magicoder-DS rows are reported by Wei et al. (2024).

Model / scope	Precision	Base LR	Transferred LR	HumanEval	HumanEval+
DeepSeek-Coder-6.7B-Base	—	—	—	47.6	39.6
Magicoder-DS	FP32	—	—	**66.5**	**60.4**
Full-depth (ours)	BF16	$2^{-11.5}$	2^{-14}	64.0	57.3
Full-parameter (ours)	BF16	$2^{-11.5}$	2^{-14}	64.6	59.1
Full-depth (ours)	FP32	2^{-12}	2^{-17}	64.0	57.9
Full-parameter (ours)	FP32	2^{-12}	2^{-17}	65.2	59.8

grows, this advantage widens, because the H200 hours of our method stay fixed at one full-depth run while those of the full-depth sweep grow linearly with R.

Quality of the transferred learning rate. We assess the quality of the learning rate transferred from the proxy search in full-depth training. We also include a *full-parameter* variant that additionally trains the embeddings and the LM head at the same rate. We train both variants under each precision and evaluate their final checkpoints on HumanEval and HumanEval+. Table 1 reports pass@1 alongside two reference values from Wei et al. (2024): the base model without finetuning, and Magicoder-DS, the released model finetuned on the same dataset. At the transferred rates, both variants perform competitively with Magicoder-DS under both precisions. Our training recipe differs from that of Magicoder-DS in epochs, optimizer, and batch size, so this comparison does not isolate the learning rate choice, but shows that the transferred rates yield competitive checkpoints under our training protocol. BF16 and FP32 also perform comparably for each variant and benchmark, which indicates that direct BF16 updates are a safe choice for saving memory with little loss in performance. In addition, full-parameter training outperforms full-depth training at the same rate. Although our analysis covers only the transformer blocks, this result shows that the transferred rates remain effective when all parameters are trained, without separate tuning.

6 Conclusion and Limitations

We propose learning rate transfer across the number of trainable blocks for LLM post-training and find that the optimal rate scales roughly as k^{-1} under FP32 master weights and $k^{-1/2}$ under direct BF16 updates. Whether other hyperparameters transfer similarly remains open, but our results establish a viable transfer axis. Our analysis relies on assumptions supported by profiling, and a more rigorous treatment is needed to establish when they hold. Nonetheless, our work shows update rounding as a key factor in learning rate scaling, providing directions for further theoretical study.

AI USE STATEMENT

In this work, we used general-purpose LLM assistants to discuss and refine theoretical arguments, assumptions, hypotheses, and proof drafts, and to polish mathematical notation. We also sought feedback on research methodology, experiments, and the interpretation of results. We did not use LLM assistants to implement our method. Generating synthetic datasets, translation, cleaning or reformatting datasets, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used LLM assistants to write plotting code for scientific figures, edit the paper for readability, and identify relevant literature. We reviewed all AI-assisted work and checked the mathematical claims and proof arguments. We verified that the plotted values match our experimental results and that the plotting code does not modify the underlying data. Every AI-suggested citation was checked against the original paper. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

All authors have read and adhere to the ICLR Code of Ethics. This work studies learning rate scaling for LLM post-training and involves no human subjects. We use only publicly available LLMs and public post-training datasets, in accordance with their respective licenses, and we do not collect or release any new data. We will release code to reproduce all of our experiments. By reducing the cost of hyperparameter tuning, our findings can make LLM post-training cheaper and more accessible, particularly to researchers with limited compute. We do not anticipate negative societal impacts beyond those generally associated with improved capabilities of large language models, and therefore do not discuss such impacts further. We have no conflicts of interest to disclose.

REPRODUCIBILITY STATEMENT

We have made several efforts to ensure that the results reported in this paper can be reproduced. Anonymized source code for our method, together with the scripts and configuration files needed to rerun every experiment, is included in the supplementary materials. The complete experimental setup—datasets and preprocessing steps, model architectures, hyperparameters, training schedules, evaluation protocol, and the hardware used—is documented in Appendix B, which expands on the summary in Section 5. Assumptions for all theoretical claims are stated in Section 4, and proofs are given in Appendix A.

REFERENCES

Charles Blake, Constantin Eichenberg, Josef Dean, Lukas Balles, Luke Prince, Björn Deiseroth, Andres Felipe Cruz Salinas, Carlo Luschi, Samuel Weinbach, and Douglas Orr. u-μP: The unit-scaled maximal update parametrization. In *International Conference on Learning Representations*, volume 2025, pp. 91057–91105, 2025.

Blake Bordelon, Lorenzo Noci, Mufan Bill Li, Boris Hanin, and Cengiz Pehlevan. Depthwise hyperparameter transfer in residual networks: Dynamics and scaling limit. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=KZJehvRKGD.

Neil Burgess, Jelena Milanovic, Nigel Stephens, Konstantinos Monachopoulos, and David Mansell. Bfloat16 processing for neural networks. In *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, pp. 88–91. IEEE, 2019.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Nan Chen, Soledad Villar, and Soufiane Hayou. Learning rate scaling across loRA ranks and transfer to full finetuning. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=vCAIwEJZQ6.

Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. SFT memorizes, RL generalizes: A comparative study of foundation model post-training. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=dYur3yabMj.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback, 2023.

Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. QLoRA: Efficient finetuning of quantized LLMs. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=OUIFPHEgJU.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate, 2022.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lIsCS8b6zj.

Soufiane Hayou and Greg Yang. Width and depth limits commute in residual networks. In *International Conference on Machine Learning*, pp. 12700–12723. PMLR, 2023.

Soufiane Hayou, Arnaud Doucet, and Judith Rousseau. On the impact of the activation function on deep neural networks training. In *International conference on machine learning*, pp. 2672–2680. PMLR, 2019.

Soufiane Hayou, Eugenio Clerico, Bobby He, George Deligiannidis, Arnaud Doucet, and Judith Rousseau. Stable resnet. In *International Conference on Artificial Intelligence and Statistics*, pp. 1324–1332. PMLR, 2021.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. The impact of initialization on lora finetuning dynamics. *Advances in Neural Information Processing Systems*, 37:117015–117040, 2024a.

Soufiane Hayou, Nikhil Ghosh, and Bin Yu. LoRA+: Efficient low rank adaptation of large models. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=NEv8YqBROO.

Bobby He, James Martens, Guodong Zhang, Aleksandar Botev, Andrew Brock, Samuel L Smith, and Yee Whye Teh. Deep transformers without shortcuts: Modifying self-attention for faithful signal propagation. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=NPrsUQgMjKK.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, Tom Hennigan, Eric Noland, Katie Millican, George van den Driessche, Bogdan Damoc, Aurelia Guy, Simon Osindero, Karen Simonyan, Erich Elsen, Oriol Vinyals, Jack W. Rae, and Laurent Sifre. Training compute-optimal large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mistral 7b, 2023. URL https://arxiv.org/abs/2310.06825.

Dhiraj Kalamkar, Dheevatsa Mudigere, Naveen Mellempudi, Dipankar Das, Kunal Banerjee, Sasikanth Avancha, Dharma Teja Vooturi, Nataraj Jammalamadaka, Jianyu Huang, Hector Yuen, et al. A study of bfloat16 for deep learning training. *arXiv preprint arXiv:1905.12322*, 2019.

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

Dawid Jan Kopiczko, Tijmen Blankevoort, and Yuki M Asano. VeRA: Vector-based random matrix adaptation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=NjNfLdxr3A.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Christopher Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=i1uGbfHHpH.

Quentin Lhoest, Albert Villanova del Moral, Yacine Jernite, Abhishek Thakur, Patrick von Platen, Suraj Patil, Julien Chaumond, Mariama Drame, Julien Plu, Lewis Tunstall, Joe Davison, Mario Šaško, Gunjan Chhablani, Bhavitvya Malik, Simon Brandeis, Teven Le Scao, Victor Sanh, Canwen Xu, Nicolas Patry, Angelina McMillan-Major, Philipp Schmid, Sylvain Gugger, Clément Delangue, Théo Matussière, Lysandre Debut, Stas Bekman, Pierric Cistac, Thibault Goehringer, Victor Mustar, François Lagunas, Alexander Rush, and Thomas Wolf. Datasets: A community library for natural language processing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 175–184, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.emnlp-demo.21.

Shiwei Li, Xiandi Luo, Xing Tang, Haozhao Wang, Hao Chen, weihongluo, Yuhua Li, xiuqiang He, and Ruixuan Li. Beyond zero initialization: Investigating the impact of non-zero initialization on loRA fine-tuning dynamics. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=8V6MEtSnlR.

Yixiao Li, Yifan Yu, Chen Liang, Nikos Karampatziakis, Pengcheng He, Weizhu Chen, and Tuo Zhao. LoftQ: LoRA-fine-tuning-aware quantization for large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=LzPWWPAdY4.

Haokun Liu, Derek Tam, Muqeeth Mohammed, Jay Mohta, Tenghao Huang, Mohit Bansal, and Colin Raffel. Few-shot parameter-efficient fine-tuning is better and cheaper than in-context learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=rBCvMG-JsPd.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=1qvx610Cu7.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. DoRA: Weight-decomposed low-rank adaptation. In *Forty-first International Conference on Machine Learning*, 2024a.

Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J Liu, and Jialu Liu. Statistical rejection sampling improves preference optimization. In *The Twelfth International Conference on Learning Representations*, 2024b. URL https://openreview.net/forum?id=xbjSwwrQOe.

Zihe Liu, Jiashun Liu, Yancheng He, Weixun Wang, Jiaheng Liu, Ling Pan, Xinyu Hu, Shaopan Xiong, Ju Huang, Jian Hu, Shengyi Huang, Siran Yang, Jiamang Wang, Wenbo Su, and Bo Zheng. Tricks or traps? a deep dive into RL for LLM reasoning. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=R0JM3BWP7W.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple preference optimization with a reference-free reward. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=3Tzcot1LKb.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=r1gs9JgRZ.

Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, et al. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*, pp. 1–15, 2021.

Lorenzo Noci, Chuning Li, Mufan Li, Bobby He, Thomas Hofmann, Chris J Maddison, and Dan Roy. The shaped transformer: Attention models in the infinite depth-and-width limit. *Advances in Neural Information Processing Systems*, 36:54250–54281, 2023.

OLMo Team, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, et al. OLMo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=TG8KACxEON.

Kaan Ozkara, Tao Yu, and Youngsuk Park. Stochastic rounding for LLM training: Theory and practice. In *The 28th International Conference on Artificial Intelligence and Statistics*, 2025. URL https://openreview.net/forum?id=3j3NtXcc95.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Houwen Peng, Kan Wu, Yixuan Wei, Guoshuai Zhao, Yuxiang Yang, Ze Liu, Yifan Xiong, Ziyue Yang, Bolin Ni, Jingcheng Hu, et al. FP8-LM: Training FP8 large language models. *arXiv preprint arXiv:2310.18313*, 2023.

Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

Jack W Rae, Sebastian Borgeaud, Trevor Cai, Katie Millican, Jordan Hoffmann, Francis Song, John Aslanides, Sarah Henderson, Roman Ring, Susannah Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=HPuSIXJaa9.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimizations toward training trillion parameter models. In *SC20: international conference for high performance computing, networking, storage and analysis*, pp. 1–16. IEEE, 2020.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020. URL https://github.com/huggingface/trl.

Evan Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, Allyson Ettinger, Michal Guerquin, David Heineman, Hamish Ivison, Pang Wei Koh, Jiacheng Liu, Saumya Malik, William Merrill, Lester James Validad Miranda, Jacob Morrison, Tyler Murray, Crystal Nam, Jake Poznanski, Valentina Pyatkin, Aman Rangapur, Michael Schmitz, Sam Skjonsberg, David Wadden, Christopher Wilhelm, Michael Wilson, Luke Zettlemoyer, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. 2 OLMo 2 furious. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=2ezugTT9kU.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with OSS-instruct. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 52632–52657. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/wei24h.html.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, Online, October 2020. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/2020.emnlp-demos.6.

Yuhui Xu, Lingxi Xie, Xiaotao Gu, Xin Chen, Heng Chang, Hengheng Zhang, Zhengsu Chen, Xiaopeng Zhang, and Qi Tian. QA-LoRA: Quantization-aware low-rank adaptation of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=WvFoJccpo8.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Ge Yang, Edward Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tuning large neural networks via zero-shot hyperparameter transfer. *Advances in Neural Information Processing Systems*, 34:17084–17097, 2021.

Greg Yang and Edward J Hu. Tensor programs iv: Feature learning in infinite-width neural networks. In *International Conference on Machine Learning*, pp. 11727–11737. PMLR, 2021.

Greg Yang, James B Simon, and Jeremy Bernstein. A spectral condition for feature learning. *arXiv preprint arXiv:2310.17813*, 2023.

Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: Feature learning in infinite depth neural networks. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=17pVDnpwwl.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Juncai Liu, Lingjun Liu, Xin Liu, Haibin Lin, Zhiqi Lin, Bole Ma, Guangming Sheng, Yuxuan Tong, Chi Zhang, Mofan Zhang, Ru Zhang, Wang Zhang, Hang Zhu, Jinhua Zhu, Jiaze Chen, Jiangjie Chen, Chengyi Wang, Hongli Yu, Yuxuan Song, Xiangpeng Wei, Hao Zhou, Jingjing Liu, Wei-Ying Ma, Ya-Qin Zhang, Lin Yan, Yonghui Wu, and Mingxuan Wang. DAPO: An open-source LLM reinforcement learning system at scale. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=2a36EMSSTp.

Tao Yu, Gaurav Gupta, Karthick Gopalswamy, Amith R Mamidala, Hao Zhou, Jeffrey Huynh, Youngsuk Park, Ron Diamant, Anoop Deoras, and Jun Huan. Collage: Light-weight low-precision strategy for LLM training. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=LkJ6qOMv77.

Pedram Zamirai, Jian Zhang, Christopher R Aberger, and Christopher De Sa. Revisiting bfloat16 training. *arXiv preprint arXiv:2010.06192*, 2020.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, LILI YU, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: Less is more for alignment. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=KBMOKmX2he.

A PROOFS AND PROFILING FOR LEARNING RATE SCALING

This appendix provides the profiling setup, complete profiling results, and proofs of the scaling results in Section 4. We use the same notation and assumptions introduced in the main text.

A.1 PROFILING SETUP

This subsection describes how we obtain the profiling results in Figure 3 and Appendix A.2, which provide empirical support for the analysis in Section 4. In each profiled run, we record at every optimizer step the gradient and the update direction of every trainable block, compute from them the first-order contributions $\widetilde{d}_l(\eta)$ and $\widetilde{D}_k(\eta)$, and measure the realized loss change $\Delta\mathcal{L}_k(\eta)$ of the step. Furthermore, at selected steps of the FP32 runs, we compute the BF16 update factor $\phi_k(\eta)$ and the fraction of nonzero realized updates $p_k(\eta)$ over a wide grid of learning rates. Below, we describe the profiled runs and their configurations, define each measured quantity, and state how the measurements are aggregated for visualization.

Profiled runs. We profile the two Llama-3.2-1B settings used in Section 5.1: Llama-3.2-1B on Tülu 3 for SFT and Llama-3.2-1B-Instruct on UltraFeedback for DPO. For each setting, we profile last-k runs with $k \in \{1, 2, 4, 8, 16\}$ under both update precisions, FP32 master-weight updates and direct BF16 updates, which gives a total of 20 profiled runs. Both models have $L = 16$ blocks, so the profiled runs range from the smallest proxy to the full-depth run. The runs follow the training protocol specified in Appendix B. We restate the settings that matter for interpreting the measurements: a constant learning rate, so that η is a single number for the whole run, AdamW with zero weight decay, so that the update direction contains no decay term, global-norm gradient clipping at 1.0, BF16 autocast for computation, and effective batch sizes of 32 examples for SFT and 8 preference pairs for DPO. To limit the cost of profiling, we profile the first 3000 optimizer steps of each run, which is about 30% of the $10{,}000$-step SFT epoch and 40% of the roughly $7{,}600$-step DPO epoch.

Table 2: Settings and learning rates used for profiling. For each setting, we profile runs with $k \in \{1, 2, 4, 8, 16\}$ trainable blocks using the listed learning rate scaling rule.

Task	Model	Dataset	Update precision	Learning rate η
SFT	Llama-3.2-1B	Tülu 3	FP32 master weights	$2^{-14}/k$
SFT	Llama-3.2-1B	Tülu 3	Direct BF16	$2^{-13}/\sqrt{k}$
DPO	Llama-3.2-1B-Instruct	UltraFeedback	FP32 master weights	$2^{-18.5}/k$
DPO	Llama-3.2-1B-Instruct	UltraFeedback	Direct BF16	$2^{-15.5}/\sqrt{k}$

Learning rates used for profiling. We profile runs at the learning rates specified in Table 2. These rates match those selected by the sweeps in Section 5.1 using the training-loss metric in 19 of 20 cases. The only exception is Llama-3.2-1B with FP32 master weights at $k = 16$: the sweep identifies $2^{-17.5}$, whereas we profile at the rule value $2^{-14}/16 = 2^{-18}$, a factor of $\sqrt{2}$ lower. We use the rule value to maintain exact learning rate scaling across k.

Prescribing the learning rates does not impose the scaling properties examined by the profiling. For the same parameters, minibatch, and Adam moment estimates, changing the learning rate of the current step leaves both the raw gradient $\nabla_{\boldsymbol{\theta}_l}\mathcal{L}$ and the clipped, Adam-normalized update direction $\boldsymbol{u}_l$ unchanged. Thus, $d_l = \langle \nabla_{\boldsymbol{\theta}_l}\mathcal{L}, \boldsymbol{u}_l \rangle$ has no direct dependence on the current step's learning rate, although the learning rate could still affect it indirectly through earlier training trajectories. We measure $\eta\widetilde{D}_k(\eta)$ to check whether the prescribed rates yield first-order loss reductions on a comparable scale as the number of trainable blocks k varies. With $\eta \propto 1/k$ under FP32 master-weight updates, this requires the measured sum $\widetilde{D}_k(\eta)$ to grow as k. With $\eta \propto 1/\sqrt{k}$ under direct BF16 updates, it requires this sum to grow as $\sqrt{k}$. These growth rates are checked from the recorded raw gradients and realized update directions, rather than assumed from the learning rate rules. We also examine the individual d_l in the FP32 runs to check whether their values are comparable across blocks and last-k runs, rather than checking only their sum. The dependence of the BF16 update factor $\phi_k(\eta)$ on η is tested separately. At selected steps of the FP32 runs, we hold the model parameters, raw gra-

dients, and update directions fixed and vary the probe learning rate over a wide grid. This directly tests whether $\phi_k(\eta)$ grows approximately linearly with η in the relevant regime, rather than inferring this dependence from comparisons across runs trained at different learning rates.

First-order contributions. At each profiled step, we record the raw gradient $\nabla_{\boldsymbol{\theta}_l}\mathcal{L}$ and the realized update direction $\widetilde{\boldsymbol{u}}_l(\eta)$ for every trainable block $l \in \mathcal{A}_k$. The gradient is computed on the complete minibatch before gradient clipping and recorded in BF16 under both update precisions. We compute the realized direction from the parameters before and after the optimizer step:

$$\widetilde{\boldsymbol{u}}_l(\eta) = \frac{\boldsymbol{\theta}_l - \boldsymbol{\theta}_l^+}{\eta},$$

using parameter values in the corresponding update accumulation precision.

Let $\boldsymbol{u}_l$ denote the nominal direction produced by Adam after gradient clipping, before multiplication by η. Under direct BF16 updates,

$$\boldsymbol{\theta}_l^+ = Q(\boldsymbol{\theta}_l - \eta \boldsymbol{u}_l),$$

where Q rounds each coordinate to the nearest representable BF16 value. Thus, the recorded direction $\widetilde{\boldsymbol{u}}_l(\eta)$ includes the effect of update rounding. Under FP32 master-weight updates, we neglect FP32 rounding and treat the recorded direction as the nominal direction, $\widetilde{\boldsymbol{u}}_l(\eta) = \boldsymbol{u}_l$. From the recorded gradients and realized directions, we compute the first-order contribution of each block and its sum over all trainable blocks:

$$\widetilde{d}_l(\eta) = \big\langle \nabla_{\boldsymbol{\theta}_l}\mathcal{L},\, \widetilde{\boldsymbol{u}}_l(\eta) \big\rangle, \qquad \widetilde{D}_k(\eta) = \sum_{l \in \mathcal{A}_k} \widetilde{d}_l(\eta).$$

Each inner product includes all parameters of the block and is computed in float64 precision. Under the FP32 idealization, these quantities equal the nominal d_l and D_k. Hence, $\eta\widetilde{D}_k(\eta)$ is the first-order approximation of the step's loss reduction. Figure 8 presents this approximation.

BF16 update factor. For fixed parameters, gradients, and nominal update directions, the update factor $\phi_k(\eta) = \widetilde{D}_k(\eta)/D_k$ depends on the learning rate. We therefore measure it over a grid of learning rates rather than only at an individual run's learning rate. At selected steps of each FP32 run, we save the parameters $\boldsymbol{\theta}_l$, raw gradients $\nabla_{\boldsymbol{\theta}_l}\mathcal{L}$, and realized update directions for every trainable block $l \in \mathcal{A}_k$. We use FP32 runs so that, under the FP32 idealization, the recorded update directions equal the nominal directions $\boldsymbol{u}_l$ needed for these probes. Using these saved quantities, we compute, for each probe learning rate $\eta \in \{2^{-8}, 2^{-9}, \ldots, 2^{-26}\}$,

$$\widetilde{\boldsymbol{u}}_l(\eta) = \frac{Q(\boldsymbol{\theta}_l) - Q\big(Q(\boldsymbol{\theta}_l) - \eta \boldsymbol{u}_l\big)}{\eta},$$

$$\widetilde{D}_k(\eta) = \sum_{l \in \mathcal{A}_k} \big\langle \nabla_{\boldsymbol{\theta}_l}\mathcal{L}, \widetilde{\boldsymbol{u}}_l(\eta) \big\rangle,$$

$$\phi_k(\eta) = \frac{\widetilde{D}_k(\eta)}{D_k}.$$

Here, Q rounds each coordinate to the nearest representable BF16 value and is implemented by casting to `torch.bfloat16` in PyTorch. The subtraction $Q(\boldsymbol{\theta}_l) - \eta \boldsymbol{u}_l$ is performed in FP32, and the denominator $D_k = \sum_{l \in \mathcal{A}_k} \langle \nabla_{\boldsymbol{\theta}_l}\mathcal{L}, \boldsymbol{u}_l \rangle$ uses the same saved gradients and directions and remains fixed across different probe learning rates. The inner Q rounds the starting parameters to BF16, and the outer Q rounds the updated parameters. The probe grid extends below and above the training rates in Table 2, without changing the run's training rate. All probes at a given step use the same saved parameters, gradients, and update directions, isolating the effect of η on update rounding from its effect on the training trajectory. From the same probes, we also record the fraction of the N_k trainable parameter elements whose realized update $\widetilde{u}_{l,i}(\eta)$ is nonzero.

Profiled steps and aggregation. We record $\widetilde{d}_l(\eta)$, $\widetilde{D}_k(\eta)$, and $\Delta\mathcal{L}_k(\eta)$ at every step of each run, whereas $\phi_k(\eta)$ and $p_k(\eta)$ are measured at certain steps $t \in \{500, 1000, 1500, 2000, 2500, 3000\}$.

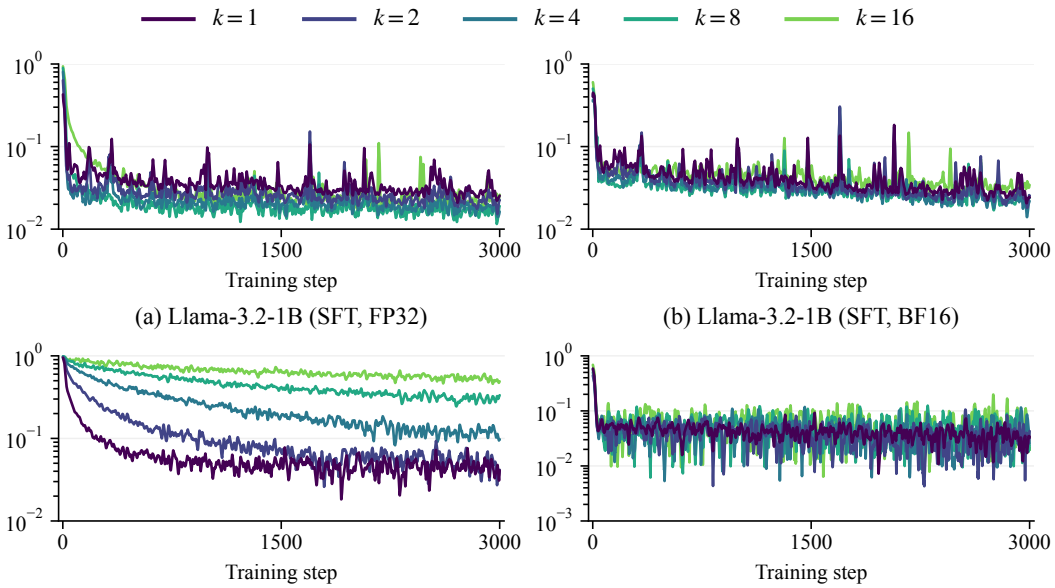
Figure 7 plots the per-step relative error $|\Delta\mathcal{L}_k(\eta) + \eta\widetilde{D}_k(\eta)|/|\eta\widetilde{D}_k(\eta)|$. Figure 8 plots $\eta\widetilde{D}_k(\eta)$ at each training step, smoothed by averaging over that step and the nine steps before it. Figure 9 plots

the geometric mean of $\widetilde{d}_l(\eta)$ over all 3000 steps against the block index l. We use the geometric rather than the arithmetic mean to reduce the influence of large, noisy values in the initial steps. Figures 10 and 11 plot $\phi_k(\eta)$, and Figures 12 and 13 plot $p_k(\eta)$, against the probe learning rate η at each of the six selected training steps.

A.2 Complete Profiling Results

The following profiling results complement the representative results in Figure 3.

Accuracy of the first-order approximation. Figure 7 reports the per-step relative error of the realized first-order loss approximation. The ratio $|\Delta\mathcal{L}_k(\eta)+\eta\widetilde{D}_k(\eta)|/|\eta\widetilde{D}_k(\eta)|$ compares the magnitude of the higher-order term with the realized first-order loss reduction. The small relative errors observed in most settings support the accuracy of the first-order approximation in those settings.



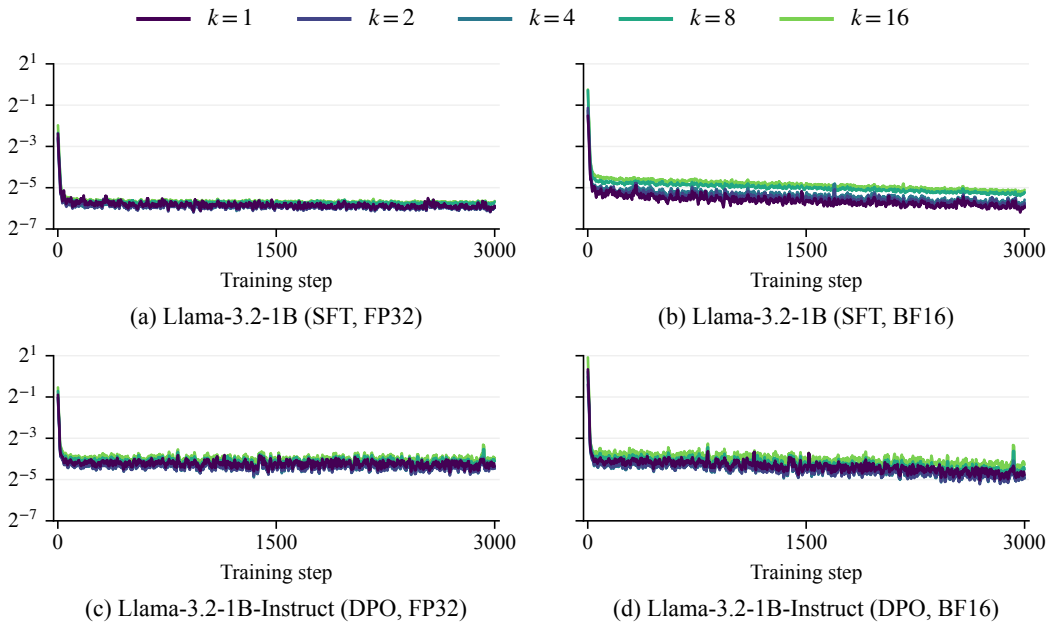
(a) Llama-3.2-1B (SFT, FP32)
(b) Llama-3.2-1B (SFT, BF16)
(c) Llama-3.2-1B-Instruct (DPO, FP32)
(d) Llama-3.2-1B-Instruct (DPO, BF16)

Figure 7: Relative error of the realized first-order loss approximation across training steps. We plot $|\Delta\mathcal{L}_k(\eta)+\eta\widetilde{D}_k(\eta)|/|\eta\widetilde{D}_k(\eta)|$ at each training step. Under the FP32 idealization, $\widetilde{D}_k(\eta)=D_k$. Each curve corresponds to a last-k run with $k\in\{1,2,4,8,16\}$ at the learning rate specified in Table 2. Panels (a, b) show SFT on Llama-3.2-1B with FP32 and BF16 update accumulation, respectively. Panels (c, d) show DPO on Llama-3.2-1B-Instruct with the same precision settings.

Stable loss reduction. Figure 8 shows the realized first-order loss reduction, $\eta\widetilde{D}_k(\eta)$, across training steps. We observe that the per-step values of $\eta\widetilde{D}_k(\eta)$ at the learning rates specified in Table 2 remain on a comparable scale across k, supporting Assumption 1.

Per-block contributions. Figure 9 plots the realized per-block contribution, $\widetilde{d}_l(\eta)$, as a function of block position l, taking the geometric mean over training steps. Under the FP32 idealization, we have $\widetilde{d}_l(\eta)=d_l$. In the FP32 panels (a, c), these values are approximately invariant to k within each block and comparable across blocks, which supports Assumption 2. The BF16 panels (b, d) are also included for comparison and reflect the effects of update rounding.

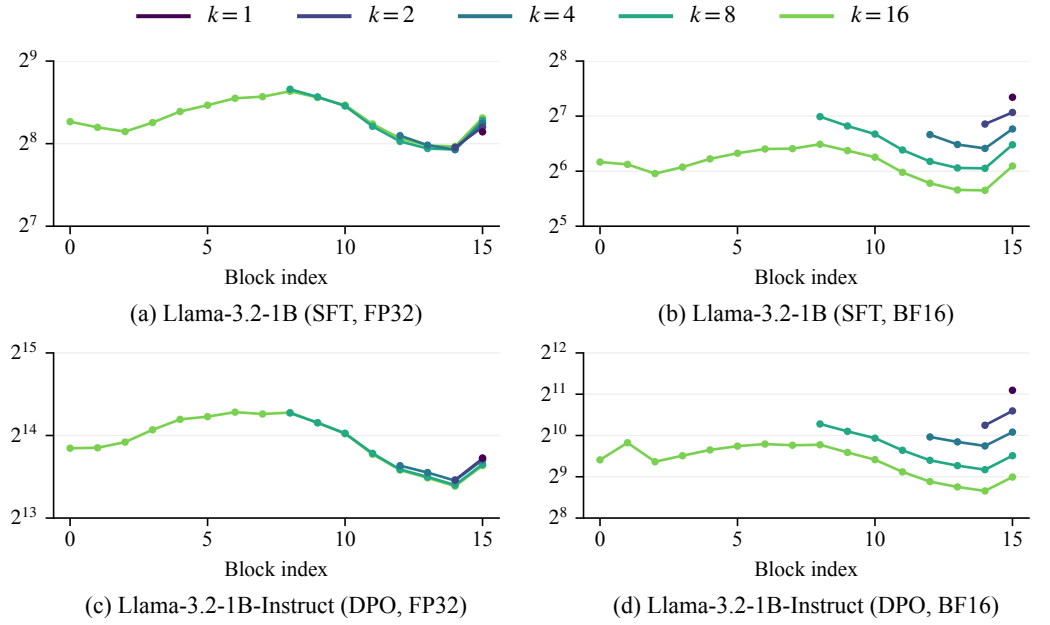
BF16 update factor. Figures 10 and 11 report the BF16 update factor as a function of learning rate for SFT and DPO, respectively. Across the displayed training steps, the curves for different k nearly coincide and grow approximately linearly with η before saturation. These profiles support the case $\alpha=1$ of Assumption 3 over the corresponding learning rate regime.



(a) Llama-3.2-1B (SFT, FP32) (b) Llama-3.2-1B (SFT, BF16)

(c) Llama-3.2-1B-Instruct (DPO, FP32) (d) Llama-3.2-1B-Instruct (DPO, BF16)

Figure 8: Realized first-order loss reduction across training steps. We plot $\eta\widetilde{D}_k(\eta)$ averaged over each step and the preceding nine steps. Under the FP32 idealization, this quantity equals ηD_k. Each curve corresponds to a last-k run with $k \in \{1, 2, 4, 8, 16\}$ at the learning rate specified in Table 2. Panels (a, b) show SFT on Llama-3.2-1B with FP32 and BF16 update accumulation, respectively. Panels (c, d) show DPO on Llama-3.2-1B-Instruct with the same respective precision settings.



(a) Llama-3.2-1B (SFT, FP32) (b) Llama-3.2-1B (SFT, BF16)

(c) Llama-3.2-1B-Instruct (DPO, FP32) (d) Llama-3.2-1B-Instruct (DPO, BF16)

Figure 9: Realized per-block first-order loss contributions. For each trainable block l, we plot the geometric mean over training steps of $\widetilde{d}_l(\eta) = \langle \nabla_{\boldsymbol{\theta}_l}\mathcal{L}, \widetilde{\boldsymbol{u}}_l(\eta)\rangle$. Under the FP32 idealization, we have $\widetilde{d}_l(\eta) = d_l$. Each curve corresponds to a last-k run with $k \in \{1, 2, 4, 8, 16\}$ at the learning rate specified in Table 2. The horizontal axis uses block indices $0, \ldots, 15$ from bottom to top. The vertical axes are logarithmic with panel-specific ranges. Panels (a, b) show SFT on Llama-3.2-1B with FP32 and BF16 update accumulation, respectively. Panels (c, d) show DPO on Llama-3.2-1B-Instruct with the same respective precision settings.

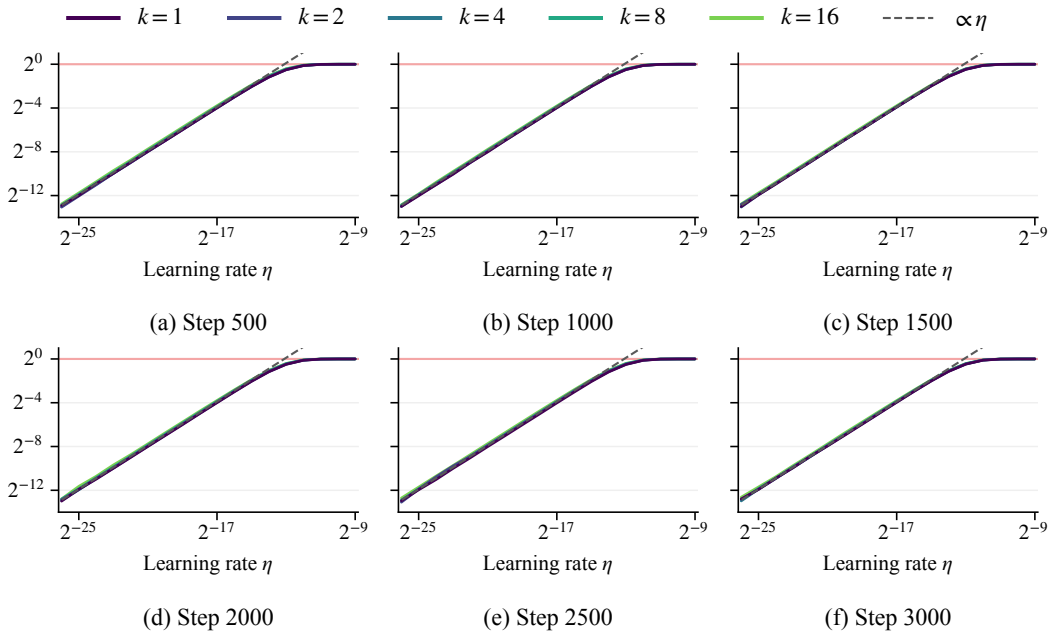


Figure 10: BF16 update factor for SFT on Llama-3.2-1B. We plot $\phi_k(\eta) = \widetilde{D}_k(\eta)/D_k$ against the learning rate η at six steps during training. The colored curves correspond to $k \in \{1, 2, 4, 8, 16\}$. Panels (a)–(f) correspond to steps 500, 1000, 1500, 2000, 2500, and 3000, respectively. Both axes are logarithmic, and the dashed line is a slope-1 reference ($\propto \eta$).

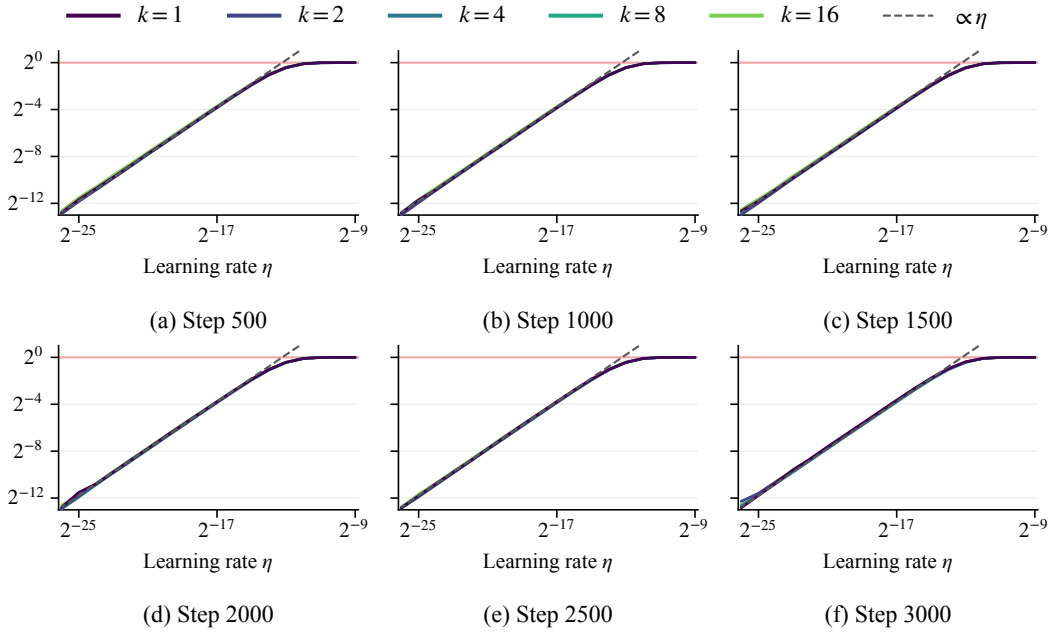
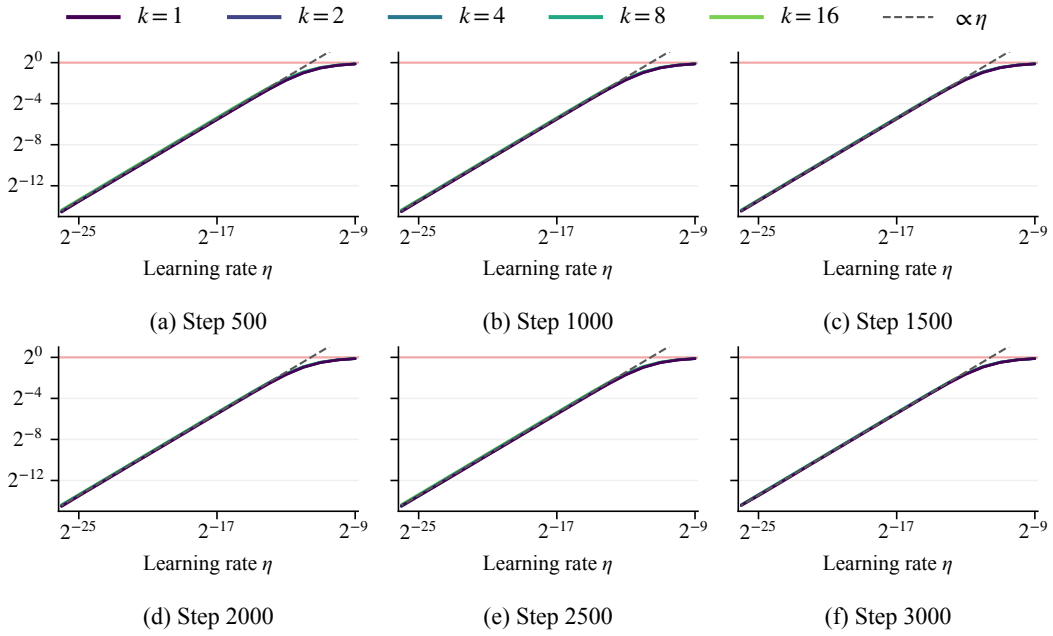


Figure 11: BF16 update factor for DPO on Llama-3.2-1B-Instruct. We plot $\phi_k(\eta) = \widetilde{D}_k(\eta)/D_k$ against the learning rate η at six steps during training. The colored curves correspond to $k \in \{1, 2, 4, 8, 16\}$. Panels (a)–(f) correspond to steps 500, 1000, 1500, 2000, 2500, and 3000, respectively. Both axes are logarithmic, and the dashed line is a slope-1 reference ($\propto \eta$).

A.3 Update Sparsity and the BF16 Update Factor



(a) Step 500 (b) Step 1000 (c) Step 1500

(d) Step 2000 (e) Step 2500 (f) Step 3000

Figure 12: Fraction of nonzero realized updates for SFT on Llama-3.2-1B. We plot $p_k(\eta)$, the fraction of trainable parameter elements whose realized update $\widetilde{u}_{l,i}(\eta)$ under direct BF16 updates is nonzero, against the learning rate η at six steps during training. The colored curves correspond to $k \in \{1, 2, 4, 8, 16\}$. Panels (a)–(f) correspond to steps 500, 1000, 1500, 2000, 2500, and 3000, respectively. Both axes are logarithmic; the dashed line is a slope-1 reference ($\propto \eta$), and the red line marks the upper bound $p_k(\eta) = 1$.

Write $g_{l,i}$, $u_{l,i}$, and $\widetilde{u}_{l,i}(\eta)$ for the coordinates of $\nabla_{\boldsymbol{\theta}_l}\mathcal{L}$, $\boldsymbol{u}_l$, and $\widetilde{\boldsymbol{u}}_l(\eta)$, respectively. Let n_l denote the number of parameters in block l and $N_k = \sum_{l \in \mathcal{A}_k} n_l$ the total number of trainable parameter elements. Write $\mathbb{E}_k$ for expectation over a coordinate $J = (l, i)$ drawn uniformly from these trainable elements. Define

$$p_k(\eta) = \frac{1}{N_k} \sum_{l \in \mathcal{A}_k} \sum_{i=1}^{n_l} \mathbf{1}_{\{\widetilde{u}_{l,i}(\eta) \neq 0\}}, \qquad \mu_k = \mathbb{E}_k[g_J u_J] = \frac{D_k}{N_k} > 0.$$

Here, $p_k(\eta)$ is the fraction of nonzero realized updates, and μ_k is the mean nominal first-order contribution per trainable parameter element. For $p_k(\eta) > 0$, the update factor factorizes exactly as

$$\phi_k(\eta) = p_k(\eta)\, \frac{\mathbb{E}_k[g_J \widetilde{u}_J(\eta) \mid \widetilde{u}_J(\eta) \neq 0]}{\mu_k}, \tag{7}$$

because coordinates with zero realized updates contribute nothing. The second factor compares the mean realized first-order contribution among coordinates with nonzero updates to the mean nominal contribution over all trainable coordinates.

Proposition A.1 (Sufficient conditions for a linear BF16 update factor (informal))**.** *Under direct BF16 updates, suppose that*

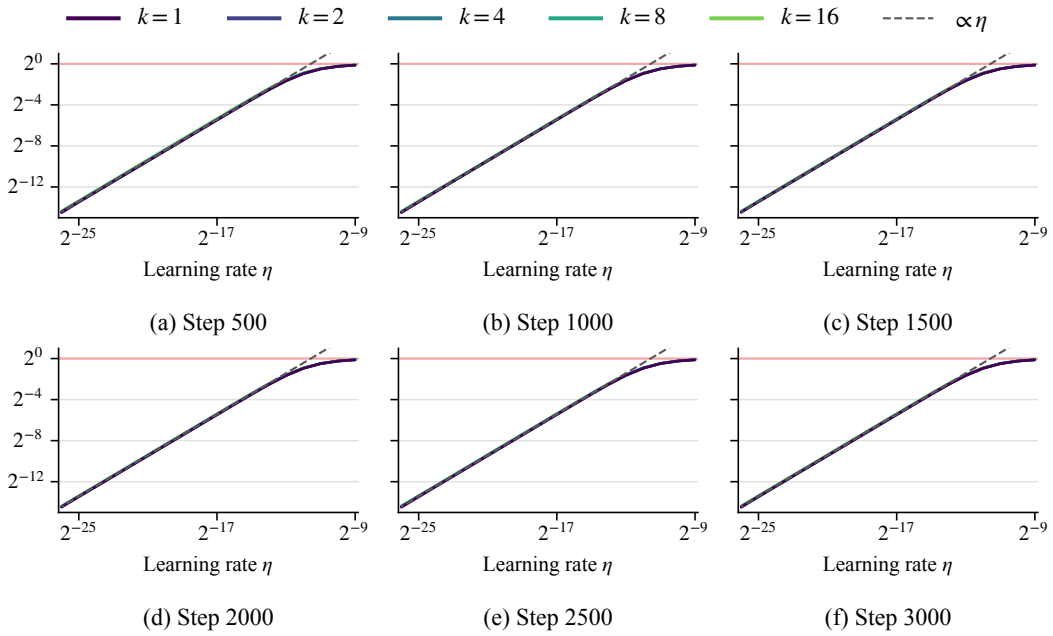
$$p_k(\eta) = \Theta(\eta) \qquad \textit{and} \qquad \frac{\mathbb{E}_k[g_J \widetilde{u}_J(\eta) \mid \widetilde{u}_J(\eta) \neq 0]}{\mu_k} = \Theta(1) \tag{8}$$

uniformly over the learning rate regime considered. Then

$$\phi_k(\eta) = \Theta(\eta) \tag{9}$$

over this regime; that is, Assumption 3 holds with $\alpha = 1$.

Figures 12 and 13 show that the fraction of nonzero realized updates $p_k(\eta)$ grows approximately linearly with η before saturation across the displayed values of k and training steps, supporting the first condition in Equation 8.



(a) Step 500 (b) Step 1000 (c) Step 1500

(d) Step 2000 (e) Step 2500 (f) Step 3000

Figure 13: Fraction of nonzero realized updates for DPO on Llama-3.2-1B-Instruct. We plot $p_k(\eta)$, the fraction of trainable parameter elements whose realized update $\widetilde{u}_{l,i}(\eta)$ under direct BF16 updates is nonzero, against the learning rate η at six steps during training. The colored curves correspond to $k \in \{1, 2, 4, 8, 16\}$. Panels (a)–(f) correspond to steps 500, 1000, 1500, 2000, 2500, and 3000, respectively. Both axes are logarithmic; the dashed line is a slope-1 reference ($\propto \eta$), and the red line marks the upper bound $p_k(\eta) = 1$.

Relevance of the linear regime. Our derivation of the $k^{-1/2}$ rule for direct BF16 updates assumes that the optimal learning rates lie in the linear regime, before $\phi_k(\eta)$ saturates. We conjecture that this holds because the realized updates at learning rates near saturation ($\phi_k(\eta) = 1$) are too large for effective post-training. Since post-training starts from useful pretrained representations, adaptation need not involve large changes throughout the model (Hayou et al., 2024a). FP32 master-weight updates can change every coordinate, so the optimal FP32 learning rates, which are much smaller than the saturation point, provide empirical evidence of the useful update scale. Direct BF16 updates require larger learning rates to compensate for increments suppressed by rounding (Section 5.1). However, this compensation does not require every coordinate to change. Under the conditions of Proposition A.1, the realized first-order loss reduction is roughly proportional to η times the fraction of nonzero updates. Because this fraction also grows with η, a larger learning rate allows BF16 to reach the loss reduction obtained at the optimal FP32 learning rate before $\phi_k(\eta)$ saturates. Raising η further toward saturation changes more coordinates by larger increments, which can make the realized update too large for effective adaptation. Therefore, as long as the optimal FP32 learning rate lies well below the saturation point, the optimal BF16 learning rate lies between the two, within the linear regime.

A.4 Proofs

Throughout this subsection, $\Theta(\cdot)$ denotes positive two-sided bounds with constants independent of k and L. In Assumption 2, the same bounds on d_l apply to every trainable block in the transformer. In Assumption 3, the ratio $\phi_k(\eta)/\eta^\alpha$ remains between fixed positive constants throughout the learning rate regime considered. In the following, we present a proof for Proposition 4.1 and Corollary 4.2.

Proposition 4.1 (restated). *Let* $1 \le k \le L$, $\mathcal{A}_k = \{L - k + 1, \dots, L\}$, *and* $D_k = \sum_{l \in \mathcal{A}_k} d_l$. *Suppose that, for a fixed* $\alpha \ge 0$,

$$d_l = \Theta(1), \qquad l \in \mathcal{A}_k, \qquad \phi_k(\eta) = \Theta(\eta^\alpha)$$

throughout the learning rate regime considered. Assume also that the higher-order term satisfies

$$|\Delta\mathcal{L}_k(\eta) + \eta D_k \phi_k(\eta)| \le \epsilon\, \eta D_k \phi_k(\eta), \qquad 0 \le \epsilon < 1,$$

where $\Delta\mathcal{L}_k(\eta) = \mathcal{L}(\boldsymbol{\theta}^+) - \mathcal{L}(\boldsymbol{\theta})$ and ϵ is fixed independently of k, L, and η. Then every optimal learning rate η_k within this regime, satisfying $-\Delta\mathcal{L}_k(\eta_k) = \Theta(1)$, obeys

$$\eta_k = \Theta\Big(k^{-1/(1+\alpha)}\Big).$$

Proof. Assumption 2 gives

$$D_k = \sum_{l \in \mathcal{A}_k} d_l = \Theta(k).$$

Combining this with $\phi_k(\eta) = \Theta(\eta^\alpha)$ yields

$$\eta D_k \phi_k(\eta) = \Theta\big(k\eta^{1+\alpha}\big).$$

The first-order approximation implies

$$(1-\epsilon)\eta D_k \phi_k(\eta) \le -\Delta\mathcal{L}_k(\eta) \le (1+\epsilon)\eta D_k \phi_k(\eta).$$

Since $\epsilon < 1$ is fixed, it follows that $-\Delta\mathcal{L}_k(\eta) = \Theta(k\eta^{1+\alpha})$. Therefore, by assumption 1,

$$k(\eta_k)^{1+\alpha} = \Theta(1).$$

Dividing by k and taking the positive $(1+\alpha)$-th root gives

$$\eta_k = \Theta\Big(k^{-1/(1+\alpha)}\Big).$$

□

Corollary 4.2 (restated). *Under the conditions of Proposition 4.1, FP32 master-weight updates have $\phi_k(\eta) = 1$, corresponding to $\alpha = 0$, whereas direct BF16 updates with $\phi_k(\eta) = \Theta(\eta)$ have $\alpha = 1$. Consequently,*

$$\eta_k = \begin{cases} \Theta(k^{-1}), & \text{FP32 master-weight updates,} \\ \Theta(k^{-1/2}), & \text{direct BF16 updates.} \end{cases}$$

Proof. Under the FP32 idealization, Q is the identity, so $\widetilde{\boldsymbol{u}}_l(\eta) = \boldsymbol{u}_l$ and $\widetilde{D}_k(\eta) = D_k$. Thus $\phi_k(\eta) = 1$, and the proposition with $\alpha = 0$ gives $\eta_k = \Theta(k^{-1})$. For direct BF16 updates without an FP32 master copy, the assumed relation $\phi_k(\eta) = \Theta(\eta)$ corresponds to $\alpha = 1$. The proposition then gives $\eta_k = \Theta(k^{-1/2})$. □

Proposition A.1 (restated). *Under direct BF16 updates, suppose that $D_k > 0$ and*

$$p_k(\eta) = \Theta(\eta), \qquad \frac{\mathbb{E}_k[g_J \widetilde{u}_J(\eta) \,|\, \widetilde{u}_J(\eta) \neq 0]}{\mu_k} = \Theta(1),$$

uniformly over the learning rate regime considered, with positive constants independent of k, L, and η. Then $\phi_k(\eta) = \Theta(\eta)$ throughout this regime, so Assumption 3 holds with $\alpha = 1$.

Proof. The two conditions give constants $0 < a \le b < \infty$ and $0 < c \le C < \infty$, independent of k, L, and η, such that

$$a\eta \le p_k(\eta) \le b\eta, \qquad c \le \frac{\mathbb{E}_k[g_J \widetilde{u}_J(\eta) \,|\, \widetilde{u}_J(\eta) \neq 0]}{\mu_k} \le C$$

throughout the regime considered. In particular, $p_k(\eta) > 0$ for $\eta > 0$, so the conditional expectation is well-defined. Since $\mu_k = D_k/N_k > 0$, the exact factorization in Equation 7 gives

$$ac\,\eta \le p_k(\eta)\, \frac{\mathbb{E}_k[g_J \widetilde{u}_J(\eta) \,|\, \widetilde{u}_J(\eta) \neq 0]}{\mu_k} = \phi_k(\eta) \le bC\,\eta.$$

The constants ac and bC are positive and independent of k, L, and η. Therefore, $\phi_k(\eta) = \Theta(\eta)$ uniformly over this regime, which is Assumption 3 with $\alpha = 1$. □

Table 3: Training settings shared by all experiments, unless stated otherwise.

Setting	Value
Training duration	1 epoch
Optimizer	Fused PyTorch AdamW
Adam coefficients	$(\beta_1, \beta_2) = (0.9, 0.999)$
Adam epsilon	10^{-8}
Weight decay	0
Gradient clipping	Maximum gradient norm 1.0
Mixed-precision compute	BF16
TF32 matrix multiplication	On
Padding multiple	8 tokens
Random seed	42

B Experimental Setup

B.1 Shared training protocol

Trainable parameters and learning rate. Unless stated otherwise, we finetune the last k blocks of a model with L transformer blocks,

$$\mathcal{A}_k = \{L - k + 1, \ldots, L\}.$$

All parameters of these blocks are trainable: the self-attention modules, the gated MLPs, and the RMSNorm parameters. The input embeddings, the LM head, the final normalization layer, and all unselected blocks stay frozen. Full-depth runs set $k = L$. Weights are updated in place, and no adapters such as LoRA (Hu et al., 2022) are used. Learning rates are swept on a base-2 grid, $\eta = 2^{-a}$, with integer exponents a and half-integer refinements near the optimum.

Optimization and batching. Table 3 lists the settings shared by all experiments. Batch sizes are effective batch sizes: with N data-parallel processes, a per-device microbatch of b, and g accumulated microbatches, one optimizer step uses Nbg examples. For DPO, an example is a preference pair, that is, a prompt with one chosen and one rejected response. We smooth the training loss ℓ_t at step t with an exponential moving average (EMA), following $\bar{\ell}_t = 0.1\,\ell_t + 0.9\,\bar{\ell}_{t-1}$.

Parameter-update precision. FP32 and BF16 denote the precision in which parameter updates are accumulated, not necessarily the precision used for forward and backward computation.

For runs without DeepSpeed, we load and store the model weights in the specified dtype. PyTorch AdamW stores its first- and second-moment estimates in the same dtype as the parameters it updates. Thus, FP32 runs store the weights and optimizer moments in FP32, whereas BF16 runs store all states in BF16 and apply updates directly to the BF16 stored weights, without an FP32 master copy. Both configurations use PyTorch BF16 autocast, which selects the precision of individual operations without changing the dtype of the stored weights. Matrix multiplications in linear and attention layers use BF16, with inputs and weights cast as needed, while numerically sensitive operations such as cross-entropy loss use FP32. Backward operations follow the precision selected for their corresponding forward operations, and each weight's gradient is stored in that weight's dtype. Autocast therefore controls computation precision but does not change the precision in which parameter updates are accumulated.

For larger models, FP32 runs are instead trained on multiple GPUs with DeepSpeed ZeRO stage 3 (Rajbhandari et al., 2020), as marked in the ZeRO-3 column of Tables 4 and 5. In our ZeRO-3 configuration, forward and backward computation uses BF16 weights, while parameter updates are accumulated in a separate FP32 master copy. Specifically, gradients are accumulated in FP32 with gradient accumulation, and AdamW updates the master weights using FP32 moments. After each optimizer step, the updated master weights are cast and copied back to the BF16 weights for subsequent computation. These runs thus retain FP32 update accumulation despite using BF16 weights for computation. ZeRO-3 also partitions the weights, gradients, master copy, and optimizer moments across GPUs to reduce per-GPU memory requirements. During forward computation, ZeRO-3 uses

Table 4: SFT experiment details. Train and Valid give the numbers of examples after preprocessing. Length gives the maximum sequence length in tokens, and Batch the effective batch size. Cosine denotes linear warmup over 5% of the steps followed by cosine decay to 0.1η. Constant denotes zero warmup followed by a constant learning rate. ZeRO-3 indicates whether the FP32 runs use DeepSpeed ZeRO stage 3. BF16 runs do not use DeepSpeed.

Model	Dataset	Train	Valid	Length	Batch	Schedule	ZeRO-3
Llama-3.2-1B	Tülu 3	320,000	32,000	1,024	32	Constant	No
DeepSeek-Coder-6.7B	Magicoder-OSS-Instruct	72,198	—	1,024	32	Cosine	Yes
Mistral-7B-v0.1	Magicoder-OSS-Instruct	72,360	—	1,024	32	Cosine	—

Table 5: DPO experiments. Train and Valid give the numbers of preference pairs after preprocessing. Loss gives the DPO variant, and β is the corresponding DPO coefficient. Length gives the maximum sequence length in tokens, and Batch the effective batch size. Constant denotes zero warmup followed by a constant learning rate. ZeRO-3 indicates whether the FP32 runs use DeepSpeed ZeRO stage 3. BF16 runs do not use DeepSpeed.

Model	Dataset	Train	Valid	Length	Batch	Schedule	Loss	β	ZeRO-3
Llama-3.2-1B-Instruct	UltraFeedback	61,141	987	1,024	8	Constant	Sigmoid	0.1	No
OLMo-2-1124-7B-SFT	OLMo preference mix	24,000	2,400	2,048	8	Constant	Sigmoid-norm	5.0	Yes

all-gather communication to reconstruct each layer's full weight matrices from the parameter shards distributed across GPUs, which incurs additional communication overhead. The precision scheme above corresponds to `bf16.enabled=auto` and `data_types.grad_accum_dtype=fp32` in our DeepSpeed configuration. We additionally set `overlap_comm=true` to overlap communication with computation and `contiguous_gradients=true` to reduce memory fragmentation, and we use no parameter or optimizer offloading, so all shards remain in GPU memory.

B.2 Supervised finetuning (SFT)

Table 4 lists the SFT experiments in this work. The loss is the negative log-likelihood (NLL) averaged over supervised tokens, and sequences are not packed.

Llama-3.2-1B on Tülu 3. In this part, we finetune `Llama-3.2-1B` (Grattafiori et al., 2024)[3] on `tulu-3-sft-mixture` (Lambert et al., 2025)[4], drawing disjoint random subsets of the dataset for training and for held-out evaluation. Conversations are rendered in the Llama 3 chat format with the fixed system message "You are a helpful assistant", and the loss covers the tokens of assistant turns only. We evaluate the held-out NLL every $1{,}000$ optimizer steps.

DeepSeek-Coder-6.7B on Magicoder-OSS-Instruct. In this part, we finetune a pretrained code model `deepseek-coder-6.7b-base` (Guo et al., 2024)[5] on a public coding dataset `Magicoder-OSS-Instruct-75K` (Wei et al., 2024)[6]. Each example pairs a coding problem with a solution, formatted with the prompt given by the dataset authors[7], and the loss covers solution tokens only. We evaluate the final checkpoint on HumanEval (Chen et al., 2021) and HumanEval+ with the EvalPlus harness (Liu et al., 2023), using the training prompt format and greedy decoding with at most 768 new tokens, and report pass@1. We leverage the vLLM engine (Kwon et al., 2023) for generating the model response.

[3] `https://huggingface.co/meta-llama/Llama-3.2-1B`
[4] `https://huggingface.co/datasets/allenai/tulu-3-sft-mixture`
[5] `https://huggingface.co/deepseek-ai/deepseek-coder-6.7b-base`
[6] `https://huggingface.co/datasets/ise-uiuc/Magicoder-OSS-Instruct-75K`
[7] `https://github.com/ise-uiuc/magicoder`

Table 6: Software versions (top) and hardware of the training node (bottom).

Component	Version or specification
PyTorch (Paszke et al., 2019)	2.11.0, CUDA 13.0 build
Transformers (Wolf et al., 2020)	5.15.0
TRL (von Werra et al., 2020)	1.10.0
Datasets (Lhoest et al., 2021)	5.0.1
Accelerate (Gugger et al., 2022)	1.14.0
DeepSpeed	0.19.5
Kernels	0.16.0
FlashAttention (Dao, 2024)	2.8.3
vLLM (Kwon et al., 2023)	0.26.0
GPU	4× NVIDIA H200, 140 GiB each
CPU	2× Intel Xeon Gold 6448Y, 32 cores each
Host memory	1,007 GiB
Operating system	Ubuntu 24.04.3 LTS, Linux kernel 6.8.0
NVIDIA driver	590.44.01

Mistral-7B-v0.1 on Magicoder-OSS-Instruct. To examine how the position of trainable layers affects the results, we adapt `Mistral-7B-v0.1` (Jiang et al., 2023)[8] with 32 layers to `Magicoder-OSS-Instruct-75K`, using the same prompt format described above. We adopt direct BF16 updates without an FP32 master copy. For *random-k*, we construct each subset by sampling k blocks uniformly at random without replacement. This sampling uses a block-selection seed that is separate from the training seed listed in Table 3. We discard duplicate subsets and keep 10 distinct subsets of layer indices for each k. Figure 18 depicts the layer indices of the 10 sampled subsets for each k, where each row is one subset and dark grey cells denote sampled trainable layers.

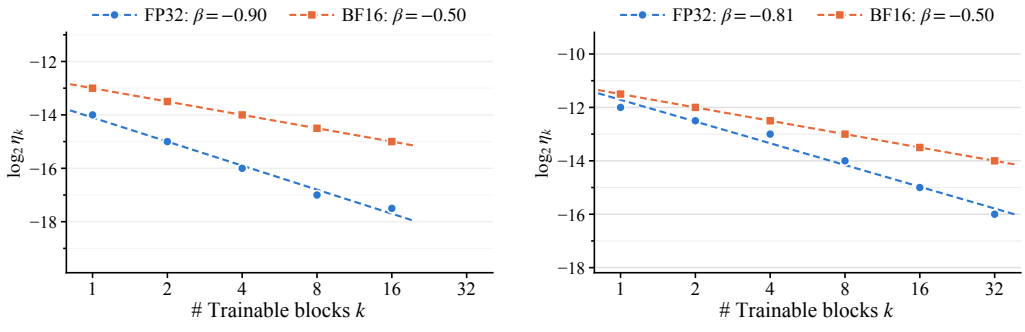
B.3 Direct preference optimization (DPO)

Table 5 lists the DPO experiments. For the standard sigmoid DPO setting, response log probabilities are summed over response tokens and the loss is averaged over the pairs in a batch, without length normalization. The OLMo setting uses the length-normalized sigmoid variant: each response's policy–reference log-ratio is divided by its response length before applying the log-sigmoid. Reference log probabilities come from the initial policy: they are computed once for all training and held-out pairs before the first update, so no separate reference model is kept during training.

Llama-3.2-1B-Instruct on UltraFeedback. `Llama-3.2-1B-Instruct` (Grattafiori et al., 2024)[9] is finetuned on `UltraFeedback` (Cui et al., 2023)[10]. The official chat template of this model inserts the current date into the system message, which is inappropriate for our experiments because the rendered prompts would then depend on the date of the run; we therefore replace the system message with the fixed text "You are a helpful assistant" and otherwise follow the Llama 3 chat format. Prompt–response sequences longer than the maximum sequence length are truncated from the right. We evaluate the DPO loss on the held-out pairs every 1,000 optimizer steps.

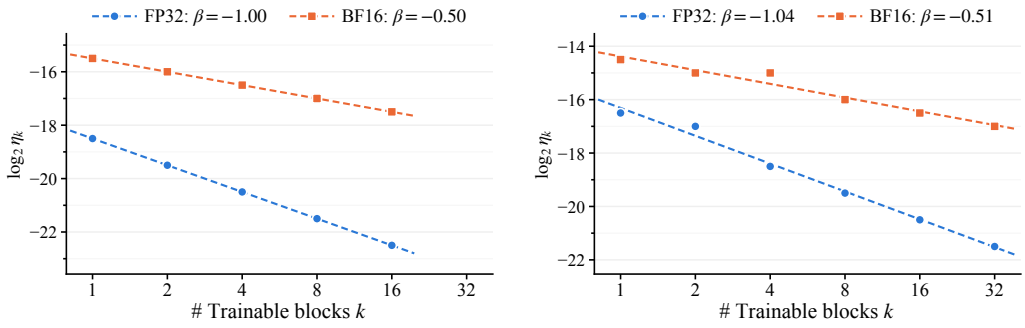
OLMo-2-1124-7B-SFT on its official mixture. We finetune `OLMo-2-1124-7B-SFT`[11] on `olmo-2-1124-7b-preference-mix` (Walsh et al., 2025)[12], formatting each example with the model's official chat template. We set the maximum sequence length to 2,048 tokens, retain only pairs whose chosen and rejected sequences are both shorter than this limit, and subsample 24,000 training and 2,400 validation pairs from the retained pairs. We train with the length-normalized sigmoid DPO loss ($\beta = 5.0$) and an effective batch size of 8 pairs, evaluating on the validation subset every 2,048 optimizer steps.

[8] https://huggingface.co/mistralai/Mistral-7B-v0.1
[9] https://huggingface.co/meta-llama/Llama-3.2-1B-Instruct
[10] https://huggingface.co/datasets/HuggingFaceH4/ultrafeedback_binarized
[11] https://huggingface.co/allenai/OLMo-2-1124-7B-SFT
[12] https://huggingface.co/datasets/allenai/olmo-2-1124-7b-preference-mix



(a) Llama-3.2-1B (SFT)

(b) DeepSeek-Coder-6.7B-Base (SFT)



(c) Llama-3.2-1B-Instruct (DPO)

(d) OLMo-2-1124-7B-SFT

Figure 14: Optimal learning rate against the number of trainable blocks k under FP32 master-weight (blue, circles) and direct BF16 (orange, squares) updates. (a) SFT of Llama-3.2-1B on Tülu 3; (b) SFT of DeepSeek-Coder-6.7B-Base on Magicoder-OSS-Instruct; (c) DPO of Llama-3.2-1B-Instruct on UltraFeedback; (d) DPO of OLMo-2-1124-7B-SFT on its official preference mixture. Each marker is the learning rate η_k that minimizes the final training loss (EMA-smoothed) with only the last k blocks trainable, taken from the sweeps of Figure 1 and Figure 4. Dashed lines are least-squares fits of $\log_2 \eta_k$ against $\log_2 k$, and each legend reports the fitted slope β, i.e., $\eta_k \propto k^{\beta}$, to be compared with $\beta = -1$ under FP32 and $\beta = -1/2$ under direct BF16 updates (Equation (6)).

B.4 Software and hardware

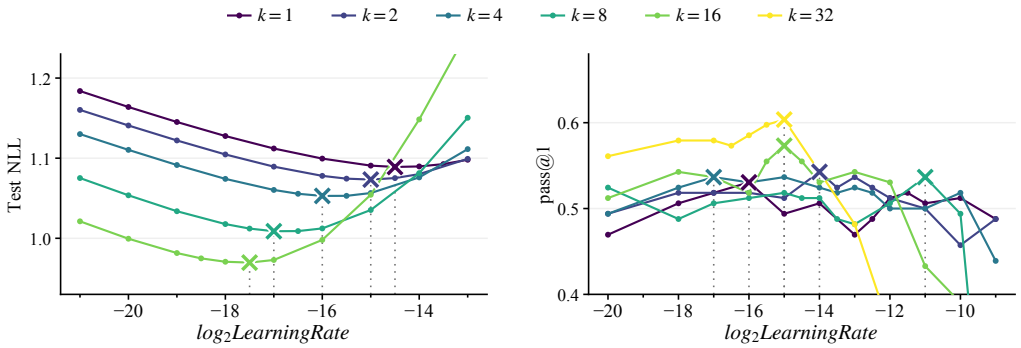
Table 6 lists the software and hardware environment of our experiment.

C Additional Experimental Results

C.1 Precision-Dependent Scaling across the Number of Trainable Blocks

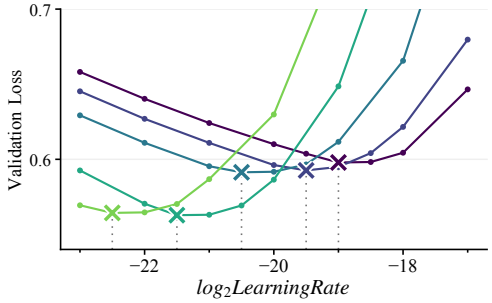
Optimal learning rate against the number of trainable blocks. Figure 14 summarizes the learning rate sweeps of Section 5.1: for each of setting, it plots the optimal learning rate η_k against the number of trainable blocks k under both update precisions, with a least-squares fit of $\log_2 \eta_k$ against $\log_2 k$ per precision.

LR sweep with regard to validation metrics. Figures 15 and 16 show the same runs as Section 5.1, with the learning rate selected by the task-specific validation metric instead of the EMA-smoothed training loss, under FP32 master-weight and direct BF16 weight updates, respectively. For the three settings evaluated by validation loss, the results agree with the training-loss sweeps: the optimal learning rate decreases with k at roughly k^{-1} under FP32 and $k^{-1/2}$ under BF16. In addition, the FP32 curves are flatter around their minima, so the selected rate is more sensitive to run-to-run noise. For DeepSeek-Coder-6.7B-Base, where the validation metric is HumanEval+ pass@1, the picture is different. For $k < 16$ the curves are nearly flat: learning rates spanning several powers of

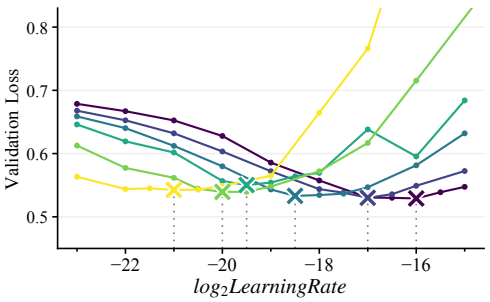


(a) Llama-3.2-1B (SFT)

(b) DeepSeek-Coder-6.7B-Base (SFT)



(c) Llama-3.2-1B-Instruct (DPO)



(d) OLMo-2-1124-7B-SFT (DPO)

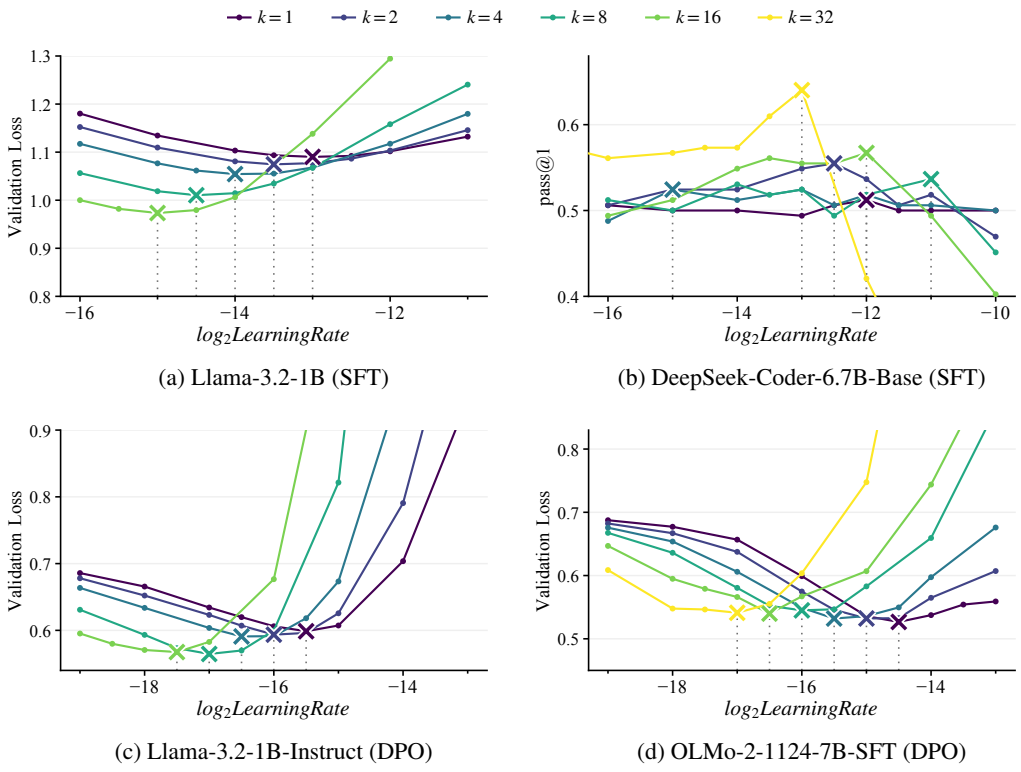
Figure 15: Learning rate sweeps under FP32 master-weight updates, selected by the task-specific validation metric. (a) SFT of Llama-3.2-1B on Tülu 3, validation NLL; (b) SFT of DeepSeek-Coder-6.7B-Base on Magicoder-OSS-Instruct, HumanEval+ pass@1 (higher is better); (c) DPO of Llama-3.2-1B-Instruct on UltraFeedback, validation DPO loss; (d) DPO of OLMo-2-1124-7B-SFT on its official preference mixture, validation DPO loss. Each curve shows the validation metric of the final checkpoint versus $\log_2$ learning rate with only the last k blocks trainable. Crosses and dotted lines mark each curve's optimal learning rate: the lowest loss in (a), (c) and (d), and the highest pass@1 in (b).

two give almost the same pass@1, so no single rate stands out. Only $k = 16$ and $k = 32$ show a clear peak. By contrast, the training loss has a distinct minimum at every k and changes visibly from one grid point to the next, so it could reveal small differences between learning rates that pass@1 cannot. We therefore identify the learning rate for transfer from the training loss.

Ablation study on finetuning only self-attention or MLP modules. We test whether the scaling rule changes when only the self-attention or only the MLP modules of the last k blocks are trainable. We finetune Mistral-7B-v0.1 on Magicoder-OSS-Instruct under direct BF16 updates. Results in Figure 17 show that the optimal learning rate scales as $k^{-1/2}$ in both cases. At every k, the optimal learning rate for the MLP modules is typically lower than that for the self-attention modules.

C.2 Effect of Trainable-Block Placement

Figure 18 shows the positions of the trainable blocks within each sampled subset, with subsets ordered by final training loss.



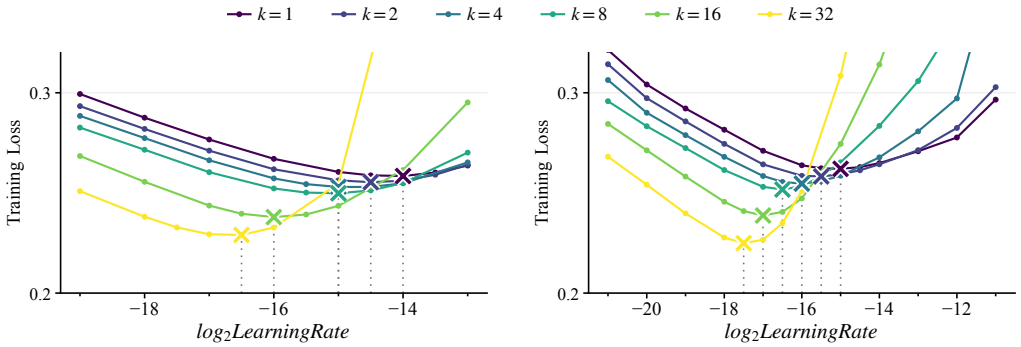
(a) Llama-3.2-1B (SFT)

(b) DeepSeek-Coder-6.7B-Base (SFT)

(c) Llama-3.2-1B-Instruct (DPO)

(d) OLMo-2-1124-7B-SFT (DPO)

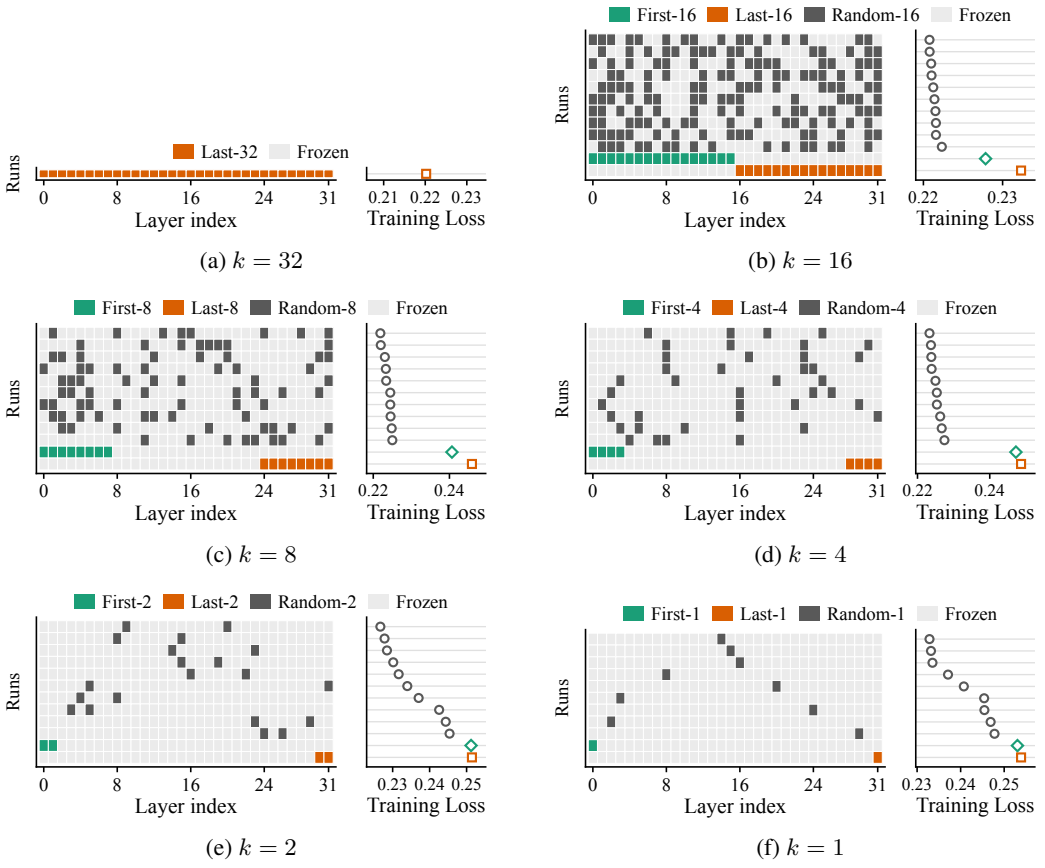
Figure 16: Learning rate sweeps under direct BF16 updates, selected by the task-specific validation metric. (a) SFT of Llama-3.2-1B on Tülu 3, validation NLL; (b) SFT of DeepSeek-Coder-6.7B-Base on Magicoder-OSS-Instruct, HumanEval+ pass@1 (higher is better); (c) DPO of Llama-3.2-1B-Instruct on UltraFeedback, validation DPO loss; (d) DPO of OLMo-2-1124-7B-SFT on its official preference mixture, validation DPO loss. Each curve shows the validation metric of the final checkpoint versus $\log_2$ learning rate with only the last k blocks trainable. Crosses and dotted lines mark each curve's optimal learning rate: the lowest loss in (a), (c) and (d), and the highest pass@1 in (b).



(a) Self-attention modules only

(b) MLP only

Figure 17: Learning rate sweeps when only the (a) self-attention modules or (b) MLP modules of the last k blocks are trainable. Both panels show SFT of Mistral-7B-v0.1 on Magicoder-OSS-Instruct under direct BF16 updates, for $k \in \{1, 2, 4, 8, 16, 32\}$. Each curve shows final training loss (EMA-smoothed) versus $\log_2$ learning rate. Crosses and dotted lines mark each curve's optimal rate.



(a) $k = 32$ (b) $k = 16$

(c) $k = 8$ (d) $k = 4$

(e) $k = 2$ (f) $k = 1$

Figure 18: Trainable-layer placements for the runs in Figure 5. Each subfigure corresponds to a fixed number of trainable blocks k. Within each subfigure, each row shows one placement: the first k layers (green), the last k layers (orange), or k randomly selected layers (grey; one row per subset). Filled and pale cells indicate trainable and frozen layers, respectively. The right axis reports each placement's final training loss at its best learning rate. Within each subfigure, rows are ordered from lowest to highest loss.

Table 7: Notation used in the cost model.

Symbol	Meaning
B	batch size, in sequences
T	sequence length, in tokens
L	number of transformer blocks
d	hidden size
d_{ffn}	feed-forward hidden size
n_h, d_h	number of attention heads and dimension per head
V	vocabulary size
k_0	number of trainable top blocks, $k_0 \in \{1, \dots, L\}$
R	number of learning rate trials in the sweep
$N = 12Ld^2$	non-embedding parameter count
γ	head cost relative to one block

D FLOP Cost Analysis

D.1 Setup and Counting Conventions

The learning rate search procedure. Given a single pretrained checkpoint, we run a sweep of R independent learning rate trials. In each trial the top k_0 transformer blocks are trainable, including their normalization parameters, and every other parameter stays frozen. The searched learning rate is then scaled to the value used for a finetuning run in which all transformer blocks are trainable. The baseline is a vanilla sweep of R full-depth trials over the same learning rate grid. The proxy pipeline must pay for the final full-depth run as well, while the vanilla sweep pays for nothing beyond the sweep.

The finetuning setting here is offline LLM post-training on a fixed tokenized dataset, such as SFT and DPO. We assume that each example has fixed token sequences, attention masks, and position IDs, and that the frozen prefix is evaluated deterministically. Its outputs can therefore be cached once and reused across trials. Cached examples may still be shuffled, provided the sequence context that determines each cached feature remains unchanged. This caching characteristic does not extend to on-policy reinforcement learning, whose training tokens are sampled from the current policy rather than from a fixed dataset.

Notation. Table 7 summarizes the symbols used in the analysis. The nominal architecture is a dense decoder-only transformer with full multi-head self-attention and a two-layer feed-forward network (FFN), where $n_h d_h = d$ and $d_{\text{ffn}} = 4d$. Its language-model head is untied from the input embedding and frozen in every run, including the full-depth runs.

Counting convention and omitted terms. We report *analytical FLOPs*, counting the floating-point operations in the dense matrix products of the forward and backward passes, with each multiply–add contributing two FLOPs (Kaplan et al., 2020; Narayanan et al., 2021; Hoffmann et al., 2022). The count is hardware-independent, so it does not change with kernel efficiency or device utilization. It is also not the arithmetic a run actually performs, which includes every term excluded below. Counts are per token unless stated otherwise. For DPO, the analysis counts only policy-model computation and assumes that reference log probabilities are already available.

Considered matrix products are the attention projections, the two attention matrix products, the feed-forward network, and the LM head. Everything else is omitted, including the embedding lookup; bias and residual additions; LayerNorm or RMSNorm normalization; elementwise nonlinearities; rotary and other positional encodings; attention scaling, masking and dropout; the attention softmax; and the vocabulary softmax with its cross-entropy. Each omitted term is lower order in the model dimension. The largest is the attention softmax, at $O(BT^2 n_h)$ per layer against the attention products at $4BT^2 d = 4BT^2 n_h d_h$. It is therefore smaller by a factor of order d_h wherever those products appear. The vocabulary softmax costs $O(V)$ per token against the counted head at $2dV$.

Two further terms, activation recomputation (or gradient checkpointing) and optimizer updates, are omitted. Under standard blockwise gradient checkpointing, recomputation adds approximately one additional forward pass of each checkpointed block per training step. Checkpointing all trainable blocks therefore adds approximately $L\,C_{\mathrm{fwd,blk}}$ to a full-depth step, whereas a proxy step incurs only $k_0\,C_{\mathrm{fwd,blk}}$ and, for small k_0, may not need checkpointing at all. Optimizer-update cost is approximately proportional to the number of trainable parameters, so training k_0 of the L blocks reduces this cost by a factor of about L/k_0. These omitted costs can increase the compute savings relative to the analytical counts reported here, depending on the checkpointing policy.

D.2 Proxy-Trial Costs and Cache Amortization

Forward propagation. Consider a batch of B sequences of length T. Within a single transformer block, the query, key and value projections cost $6BTd^2$, the attention scores $2BT^2d$, the attention–value product $2BT^2d$, and the output projection $2BTd^2$. The feed-forward network applies two linear layers at $2BTd\,d_{\mathrm{ffn}}$ each, or $16BTd^2$ together when $d_{\mathrm{ffn}} = 4d$. Summing,

$$\begin{aligned} C_{\mathrm{fwd,blk}} &= 6BTd^2 + 2BT^2d + 2BT^2d + 2BTd^2 + 16BTd^2 \\ &= 24BTd^2 + 4BT^2d. \end{aligned}$$

A forward pass comprises L such blocks and the language-model head, which projects the final hidden states onto the vocabulary at

$$C_{\mathrm{fwd,head}} = 2BTdV.$$

Summing over the blocks and the head,

$$\begin{aligned} C_{\mathrm{fwd,full}} &= L\,C_{\mathrm{fwd,blk}} + C_{\mathrm{fwd,head}} \\ &= L(24BTd^2 + 4BT^2d) + 2BTdV. \end{aligned} \tag{10}$$

Backward propagation. For a matrix product $Y = XW$, the backward pass computes the input gradient $\nabla_X = \nabla_Y W^\top$ and the weight gradient $\nabla_W = X^\top \nabla_Y$, each at the same cost as the forward product. A trainable block therefore costs $2\,C_{\mathrm{fwd,blk}}$. The head is frozen, so its backward pass computes only the gradient with respect to the final hidden states, at $C_{\mathrm{fwd,head}}$. In a full-depth run every block is trainable, giving

$$C_{\mathrm{bwd,full}} = 2L\,C_{\mathrm{fwd,blk}} + C_{\mathrm{fwd,head}}.$$

With Equation 10, one full-depth run costs

$$\begin{aligned} C_{\mathrm{full}} &= C_{\mathrm{fwd,full}} + C_{\mathrm{bwd,full}} \\ &= 3L\,C_{\mathrm{fwd,blk}} + 2\,C_{\mathrm{fwd,head}} \\ &= 3L\left(24BTd^2 + 4BT^2d\right) + 4BTdV. \end{aligned} \tag{11}$$

Attention convention. The attention products above are counted as dense $T \times T$ matrix products, at $4BT^2d$ per block. Counting only the causal pairs $j \le i$ gives $2BdT(T+1)$, close to half. The factor applies to the attention term alone, which reaches the scale of the $24BTd^2$ from the projections and the feed-forward network only when $T \gtrsim 6d$.

Consistency with the $6N$ rule. Dropping the attention products and the LM head from Equation 11 leaves $72BTLd^2$, or $6N$ per token with $N = 12Ld^2$, the standard estimate of training cost in Kaplan et al. (2020); Hoffmann et al. (2022). Equation 11 is that rule with two explicit corrections, one for the attention products and one for the frozen head,

$$C_{\mathrm{full}} = BT\left(6N + 12LTd + 4dV\right).$$

D.3 Cost of one proxy run

Trainability assumptions. In a proxy trial, the top k_0 blocks, numbered $L-k_0+1$ through L, are trainable, while blocks 1 through $L-k_0$, the input embedding, and the LM head are frozen. Each trainable block costs $2\,C_{\mathrm{fwd,blk}}$ in the backward pass, including block $L-k_0+1$, whose trainable input normalization requires the input gradient of its first matrix product.

Without feature caching. Without caching, every trial recomputes the outputs of blocks 1 through $L - k_0$, so its forward pass comprises all L blocks and the LM head. The backward pass reaches only the LM head and the k_0 trainable blocks, since nothing below block $L - k_0 + 1$ receives a gradient. A trial therefore costs

$$\begin{aligned} C_{\text{uncached}} &= \underbrace{L\, C_{\text{fwd,blk}} + C_{\text{fwd,head}}}_{\text{forward}} + \underbrace{2k_0\, C_{\text{fwd,blk}} + C_{\text{fwd,head}}}_{\text{backward}} \\ &= (L + 2k_0)\, C_{\text{fwd,blk}} + 2\, C_{\text{fwd,head}}. \end{aligned} \tag{12}$$

Let

$$\gamma = \frac{C_{\text{fwd,head}}}{C_{\text{fwd,blk}}}$$

denote the forward cost of the LM head relative to one transformer block. For the nominal architecture above, $\gamma = V/(12d + 2T)$. Dividing Equation 11 by Equation 12 gives the compute-reduction factor of a trial against one full-depth run,

$$\rho_{\text{uncached}} = \frac{C_{\text{full}}}{C_{\text{uncached}}} = \frac{3L + 2\gamma}{L + 2k_0 + 2\gamma}. \tag{13}$$

The bound $\rho_{\text{uncached}} < 3$ is strict for every L, every $k_0 \geq 1$ and every $\gamma \geq 0$, and it is tight. In situations where $k_0 = 1$ with $V \ll 12d + 2T$,

$$\rho_{\text{uncached}} \approx \frac{3L}{L + 2},$$

which approaches 3 as L grows. Freezing the lower blocks removes their backward pass and nothing else. Their forward pass is repeated in every trial, while a full run costs at most three times a single forward pass, so without caching no choice of k_0 can save more than a factor of three.

With feature caching. Under the fixed-input and deterministic-prefix assumptions above, the outputs of the frozen blocks can be computed once and stored for reuse, at the cost of disk storage that is cheap relative to accelerator memory. The cache point is the output of block $L - k_0$, which is the input of block $L - k_0 + 1$. The cache is built with one forward pass of the dataset through blocks 1 through $L - k_0$,

$$C_{\text{build}} = (L - k_0)\, C_{\text{fwd,blk}}, \tag{14}$$

paid once for the whole sweep. A cached trial makes one pass over the dataset, reading the stored features and passing them forward and backward through the k_0 trainable blocks and the LM head,

$$C_{\text{cached}} = 3k_0\, C_{\text{fwd,blk}} + 2\, C_{\text{fwd,head}} = (3k_0 + 2\gamma)\, C_{\text{fwd,blk}}.$$

The head contributes little to the full-depth and uncached costs when $\gamma \ll L$, but it can contribute substantially to a cached proxy when γ is comparable to k_0. In particular, its fraction of the cached-trial cost is $2\gamma/(3k_0 + 2\gamma)$. A sweep of R cached trials costs $C_{\text{build}} + R\, C_{\text{cached}}$, against $R\, C_{\text{full}}$ for R full-depth trials, so

$$\rho_{\text{cached}}(R, k_0) = \frac{R\,(3L + 2\gamma)}{(L - k_0) + R\,(3k_0 + 2\gamma)}, \tag{15}$$

with limit

$$\rho_{\text{cached}}(\infty, k_0) = \lim_{R \to \infty} \rho_{\text{cached}}(R, k_0) = \frac{3L + 2\gamma}{3k_0 + 2\gamma}. \tag{16}$$

For $k_0 < L$ the cache term $(L - k_0)/R$ falls as R grows, so ρ_{cached} increases with R and approaches equation 16 from below. At $R = 1$ the denominator of Equation 15 is $(L - k_0) + 3k_0 + 2\gamma = L + 2k_0 + 2\gamma$, so $\rho_{\text{cached}}(1, k_0) = \rho_{\text{uncached}}$. Building the cache and reading it once is a full forward pass, so a single cached pass saves nothing. With E epochs per trial, every R in Equation 15 becomes RE and Equation 16 is unchanged. The R-dependence enters only through amortization, which is why Equation 13 carries no R at all. Any saving beyond ρ_{uncached} comes from spreading Equation 14 over more proxy trials.

Table 8: Llama-2 configurations used.

Model	L	d	n_h	d_{ffn}	V
Llama-2 7B	32	4096	32	11008	32000
Llama-2 13B	40	5120	40	13824	32000
Llama-2 70B	80	8192	64	28672	32000

D.4 End-to-End Cost of Learning Rate Transfer

The two sweep procedures. The proxy pipeline builds the cache and runs R cached trials. The learning rate selected by the sweep is then scaled for full-depth training, and the target is finetuned once at the scaled value. The full-depth sweep, in contrast, runs R trials and then keeps the best one. The final run of the proxy pipeline costs the same as one full-depth trial.

Proposition D.1. *For $1 \le k_0 \le L$ and $R \ge 1$, the ratio of the analytical FLOPs required by the full-depth sweep to those required by the learning rate transfer with cache is*

$$\rho(R, k_0) = \frac{R\,(3L + 2\gamma)}{\underbrace{(L - k_0)}_{\textit{cache build}} + \underbrace{R\,(3k_0 + 2\gamma)}_{\textit{proxy trials}} + \underbrace{(3L + 2\gamma)}_{\textit{full-depth run}}}. \tag{17}$$

Proof. The full-depth sweep costs $R\,C_{\text{full}}$ and the pipeline $C_{\text{build}} + R\,C_{\text{cached}} + C_{\text{full}}$. Taking the ratio and dividing through by $C_{\text{fwd,blk}}$ gives Equation 17. □

The ratio in Equation 17 increases in R and decreases in k_0, so a larger sweep or a smaller k_0 raises it. However, neither raises it without limit.

Corollary D.2. *For $1 \le k_0 \le L$ and $R \ge 1$,*

$$\rho(R, k_0) < \min\left\{R,\ \frac{3L + 2\gamma}{3k_0 + 2\gamma}\right\} \ \le\ \min\left\{R,\ \frac{L}{k_0}\right\}. \tag{18}$$

Proof. In Equation 17, the cache term is nonnegative, while the proxy-trial and final-run terms are strictly positive. Consequently, discarding any of these terms raises the ratio $\rho(R, k_0)$. Discarding the cache and proxy-trial terms leaves R, and discarding the cache and final-run terms leaves $(3L + 2\gamma)/(3k_0 + 2\gamma)$. For the second inequality, $k_0(3L + 2\gamma) \le L(3k_0 + 2\gamma)$ reduces to $\gamma k_0 \le \gamma L$, which holds because $\gamma \ge 0$ and $k_0 \le L$. □

Two separate limits are at work, the sweep size R and the network depth L. The final run is common to both procedures and costs as much as one full-depth trial, so the proxy pipeline always pays at least a $1/R$ fraction of what the full-depth sweep pays, whatever k_0 is. A proxy trial trains k_0 blocks against L for a full-depth trial, so no proxy trial saves more than a factor of L/k_0, which is at most L, attained at $k_0 = 1$.

D.5 FLOP Estimates for Llama-2 Configurations

We instantiate Proposition D.1 using the nominal architecture with the depths, hidden sizes, and vocabulary sizes of the three Llama-2 models (Touvron et al., 2023), listed in Table 8. We fix $k_0 = 1$ and $T = 2048$, and vary the number of trials R. Let

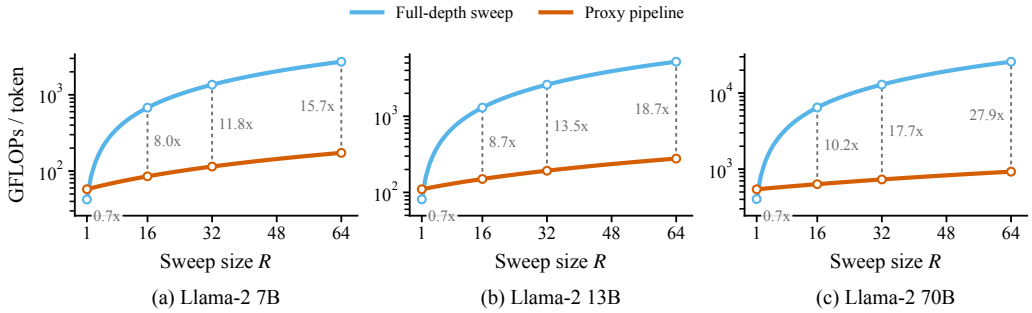
$$c_{\text{fwd,blk}} = C_{\text{fwd,blk}}/(BT) = 24d^2 + 4Td$$

denote the forward cost of one block per token. A full-depth sweep of R trials then costs

$$R\,(3L + 2\gamma)\,c_{\text{fwd,blk}}$$

per target token, and the proxy pipeline costs

$$\big[(L - 1) + R\,(3 + 2\gamma) + (3L + 2\gamma)\big]\,c_{\text{fwd,blk}}.$$



(a) Llama-2 7B (b) Llama-2 13B (c) Llama-2 70B

Figure 19: Analytical FLOPs per token against sweep size R, at $k_0 = 1$ and $T = 2048$. The proxy pipeline includes the cache build, R cached trials and one final full-depth run, so it is more expensive than a single full-depth trial at $R = 1$. Grey vertical dashed lines mark the gap between the two curves at $R \in \{1, 16, 32, 48, 64\}$, labeled with the ratio $\rho(R, 1)$ of Equation 17.

Figure 19 presents analytical FLOPs per token against sweep size for the three Llama-2 models. At $R = 1$ the ratio is below one in every model. In a single-trial sweep, the full-depth baseline requires exactly one full-depth run. By contrast, the proxy pipeline must perform that same run in addition to building the feature cache and executing one proxy trial, so it cannot be cheaper. The ratio increases with R as the cache build and final full-depth run are amortized over more proxy trials, but the gains diminish as repeated proxy trials account for a larger share of the total cost. For small sweeps, the cost of building the cache and performing the final full-depth run limits the compute reduction, since both costs are incurred once regardless of the number of trials. For sufficiently large sweeps, repeated proxy trials dominate the pipeline cost, and the reduction approaches the cached per-trial factor $\rho_{\text{cached}}(\infty, k_0) = (3L + 2\gamma)/(3k_0 + 2\gamma)$. For $k_0 = 1$, this limit remains below L because every cached trial still incurs the forward and backward costs of the frozen LM head.

Departures from the nominal architecture. Llama-2 utilizes a gated FFN with three matrices, and the 70B variant additionally uses grouped-query attention with eight key–value heads. These differences change the forward cost of a block but affect the compute-reduction factors only through γ. At $T = 2048$, accounting for the actual architectures increases the full-depth training FLOPs by approximately 0.5%, 0.8%, and 6.0% relative to the nominal estimates for 7B, 13B, and 70B, respectively. Nonetheless, with $k_0 = 1$, the relative change in $\rho(R, 1)$ remains below 0.5% for all models over the plotted range $1 \le R \le 64$. Thus, the nominal architecture closely approximates the compute-reduction factors reported here, despite a modest discrepancy in absolute FLOPs for 70B.

E Cached Learning Rate Search: Prototype Evaluation

To evaluate the practical efficiency of our methods, we build a prototype consisting of two stages: cache build (Stage 1) and cached proxy trials (Stage 2). The prototype architecture is shown in Figure 20. In Stage 1, we pass the input tokens through the frozen prefix up to block $L - k_0$ and cache the resulting features on a local disk. This stage runs in inference mode and requires only forward propagation. In Stage 2, we load the cached features and train blocks $L - k_0 + 1$ through L for each candidate learning rate. Each trial performs forward and backward propagation through these k_0 blocks and the frozen LM head. The k_0 trainable blocks serve as a proxy for the full-depth model.

We evaluate the prototype using the 32-block `deepseek-coder-6.7b-base` (Guo et al., 2024) and the Magicoder-OSS-Instruct dataset (Wei et al., 2024), which contains 75k coding question and solution pairs. We use an effective batch size of 32 and a maximum sequence length of 1024 tokens. We restrict this case study to $k_0 = 1$, where only the last block is trainable during cached proxy trials. After measuring memory and runtime, we verify in Section E.3 that the learning rate transferred from this proxy yields a competitive full-depth model.

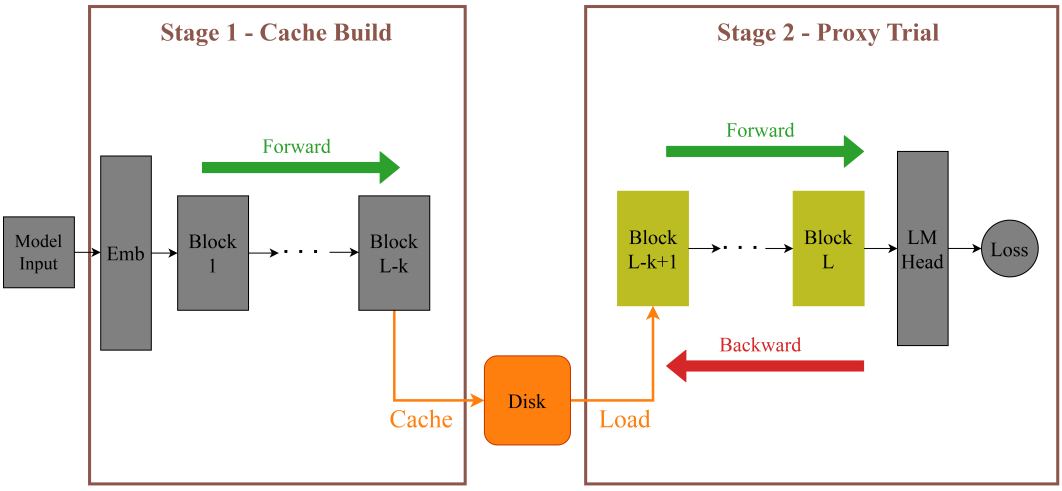


Figure 20: Two-stage prototype for learning rate search. Stage 1 (cache build) stores the output of block $L - k_0$ on a local disk after a forward pass through the frozen prefix. Stage 2 (cached proxy trials) reuses these features to train blocks $L - k_0 + 1$ through L for each candidate learning rate. The LM head remains frozen, but gradients propagate through it to the trainable blocks.

Table 9: Hardware configuration for the case study on a consumer GPU.

Component	Specification
GPU	1× NVIDIA GeForce RTX 5060 Ti, 15.9 GiB
CPU	1× AMD Ryzen 9 9900X, 12 cores, 24 threads
Host memory	30.5 GiB
Operating system	Ubuntu 24.04.4 LTS, Linux kernel 7.0.0-31-generic
NVIDIA driver	595.84

E.1 Peak Allocated GPU Memory

We measure the peak allocated GPU memory of each stage as the per-GPU batch size increases from 1 to 1024 sequences. To evaluate worst-case memory use, we construct training batches in which every sequence contains exactly 1024 tokens. For comparison, we also measure peak memory for full-depth training with and without gradient checkpointing. We run all experiments on a single H200 GPU with 140GiB of memory. The results are shown in Figures 6a and 6b.

At small batch sizes, both stages use substantially less memory than full-depth training. This difference reflects the memory required by full-depth training to store model weights, optimizer states, and gradients, even at a batch size of 1. The resulting initial memory requirement is particularly high with `float32` master weights, as shown in Figure 6a. Gradient checkpointing reduces the rate at which memory usage grows with batch size in full-depth training, at the cost of additional recomputation time. Stage 2 does not use gradient checkpointing, and its memory curve is steeper than that of a checkpointed full-depth training. Applying gradient checkpointing to Stage 2 could similarly reduce this growth rate and allow larger batch sizes during training.

E.2 Learning rate search on a consumer GPU

The preceding memory analysis suggests that our prototype can run on a consumer GPU, but its runtime on such hardware remains to be assessed. We therefore benchmark the prototype on a single NVIDIA GeForce RTX 5060 Ti with 15.45 GiB of GPU memory available to PyTorch. The specific hardware configurations are in Table 9. With this system, we evaluate on the same 6.7B model with a maximum sequence length of 1024 tokens. Stage 2 utilizes gradient accumulation to maintain an effective batch size of 32.

Table 10: Runtime (in hours) and feature-cache storage (GiB) for learning rate search with the $k_0 = 1$ prototype on a single RTX 5060 Ti with 15.45 GiB of GPU memory available to PyTorch. In this case, only the last transformer block is trainable during proxy trials. We report the fastest benchmarked configuration for each stage. For Stage 2, BF16 denotes `bfloat16` weights and optimizer states, and FP32 denotes `float32` master weights with `bfloat16` autocast. Each proxy trial makes one pass over the dataset. Sweep runtimes include one cache build and the specified number of proxy trials, but exclude the final full-depth training run.

	BF16	FP32
Stage 1 batch size	1	1
Stage 2 micro-batch × accumulation	1×32	2×16
Stage 1, run once	3.78 h	3.78 h
Stage 2, per learning rate	0.58 h	0.71 h
Cache build + one proxy trial	4.36 h	4.49 h
Cache build + five proxy trials	6.68 h	7.34 h
Cache build + ten proxy trials	9.58 h	10.89 h
Feature-cache size	336.8 GiB	336.8 GiB

Table 11: Effect of batch size on one-pass runtime. B is the Stage 1 batch size and the Stage 2 micro-batch size. Stage 2 uses $32/B$ gradient accumulation steps. All runtimes are in hours. The final column reports the total number of tokens processed over one dataset pass, including padding.

B	Stage 1 (BF16)	Stage 2 (BF16)	Stage 2 (FP32)	Tokens after padding (millions)
1	**3.78**	**0.58**	0.82	44.35
2	4.44	0.68	**0.71**	51.02
4	5.07	0.74	0.81	56.87
8	5.02	0.71	0.74	61.79
16	5.87	0.85	0.89	65.79

Table 10 summarizes the fastest benchmarked configurations for each precision, the per-stage runtime, and the sweep runtime for different learning rate grid sizes. Cache build accounts for most of the runtime when evaluating a single learning rate, but this cost is paid only once across a sweep. Each additional learning rate requires only a Stage 2 trial, making it substantially less expensive to extend the search after the cache has been built. The precision choice has a modest effect on the per-trial runtime of Stage 2, whereas cache-build runtime is comparable across the two configurations. In Stage 1, both configurations use `bfloat16` for the main forward computations, with autocast applied when weights are stored in `float32`. Since this stage performs only forward propagation without weight updates, the two precisions have comparable cache-build runtime.

On the consumer GPU tested here, the largest feasible batch does not minimize runtime. As shown in Table 11, the fastest benchmarked configurations use small batches in both stages. Because stage 1 is already compute-bound on this GPU at a batch size of 1, larger batches yield no throughput gain. Moreover, since each batch is padded to its longest sequence, larger batches introduce additional padding tokens, which incur extra computation and further increase the runtime.

The prototype's low GPU memory consumption comes with a disk-storage requirement, reported in Table 10. Both precision configurations use the same cache of `bfloat16` features, so reducing the precision of weights and optimizer states in Stage 2 does not reduce the cache size. Nonetheless, disk storage is generally less expensive per unit of capacity than GPU memory, and existing cache compression techniques could further reduce the storage requirement.

Under the tested configurations, the final full-depth run still requires more GPU memory than the RTX 5060 Ti provides. Nonetheless, the prototype allows learning rate sweeps to run on more accessible consumer hardware, so that high-end GPUs can be reserved solely for the final full-depth

Table 12: Full-depth training on the H200 computing node, using the fastest benchmarked configuration for each precision and GPU setting. Each run trains all 32 blocks for one epoch at an effective batch size of 32. BF16 uses `bfloat16` weights and optimizer states, and FP32 uses `float32` master weights with `bfloat16` autocast. Micro-batch sizes are per GPU. Single-GPU runs do not use DeepSpeed. Peak memory is the maximum allocated memory per GPU during training.

Precision	H200 GPUs	ZeRO-3	Micro-batch × accumulation	Gradient Checkpointing	Peak memory (GiB/GPU)	Runtime (h)
BF16	1	No	4×8	No	73.0	1.48
FP32	1	No	2×16	No	123.2	1.93
FP32	2	Yes	8×2	No	104.9	0.91

Table 13: GPU hours by hardware tier for a search over R candidate learning rates. The prototype includes one cache build and R proxy trials on the RTX 5060 Ti with 15.45 of available memory, followed by one final full-depth run on the H200 with 140 GiB of memory. A full-depth sweep uses the H200 node for all R trials. BF16 employs one H200 and FP32 uses two with ZeRO-3 (Table 12), so FP32 H200 hours are twice the wall-clock time.

		Prototype		Full-depth sweep
Precision	R	RTX 5060 Ti	H200	H200
BF16	1	4.36	1.48	1.48
	5	6.68	1.48	7.38
	10	9.58	1.48	14.76
FP32	1	4.49	1.83	1.83
	5	7.34	1.83	9.14
	10	10.89	1.83	18.29

run. Decoupling the two phases in this way may reduce the cost of hyperparameter search and make larger search grids practical within a fixed compute budget.

Full-depth training on H200 GPUs. Table 12 reports full-depth runtime for the same model and dataset on the H200 computing node used for our main experiments. For each precision and GPU setting, we benchmark the runtime with different configurations and adopt the fastest benchmarked configuration for reporting. With FP32 update precision, ten trials on the RTX 5060 Ti take 10.89 h, including cache construction but excluding the final full-depth run. This is comparable to the approximately 9.1 h required for ten full-depth trials on two H200s with DeepSpeed ZeRO-3.

Table 13 reports GPU hours separately by hardware tier, including the final full-depth run for the prototype. For the tested sweep sizes, the prototype uses 4.36–10.89 h on the consumer GPU and requires one full-depth run on the H200 rather than R. Although full-depth training exceeds the RTX 5060 Ti's memory capacity under the tested configurations, the search uses no high-end GPUs.

E.3 Quality of the Transferred Learning Rate

We next evaluate learning rate transfer from the $k_0 = 1$ proxy to full-depth training on the same model and dataset. The proxy sweeps select $\hat{\eta}_1 = 2^{-11.5}$ under direct BF16 updates and $\hat{\eta}_1 = 2^{-12}$ under FP32 master-weight updates. For $L = 32$, the precision-dependent scaling rules give $\hat{\eta}_{32} = 2^{-11.5}/\sqrt{32} = 2^{-14}$ under BF16 and $\hat{\eta}_{32} = 2^{-12}/32 = 2^{-17}$ under FP32. For each precision, we train two configurations at the transferred rate. The *full-depth* configuration trains all $L = 32$ transformer blocks while keeping the input embeddings, final normalization layer, and LM head frozen, matching the target of our transfer rule. The *full-parameter* configuration trains every parameter, as in Magicoder-DS (Wei et al., 2024). All runs follow the training protocol of the main experiments. We evaluate the final checkpoints on HumanEval and HumanEval+ with the EvalPlus harness and the vLLM engine, using greedy decoding, as described in Appendix B.

Table 14: Pass@1 (%) for `deepseek-coder-6.7b-base` on HumanEval and HumanEval+ after being finetuned on Magicoder-OSS-Instruct at the transferred learning rate. $\hat{\eta}_1$ is the rate selected by the $k_0 = 1$ proxy sweep, and $\hat{\eta}_{32}$ is the rate obtained by scaling it for $L = 32$ with the scaling rule of the corresponding update precision. Full-depth trains all 32 transformer blocks and keeps the input embeddings, final normalization layer, and LM head frozen. Full-parameter trains every parameter. Our runs train for one epoch at an effective batch size of 32 with a maximum sequence length of 1024 tokens, and the final checkpoint is evaluated with EvalPlus using greedy decoding. The first two rows are reference values reported by Wei et al. (2024): the base model without finetuning, and Magicoder-DS, which the dataset authors obtained by finetuning all parameters for two epochs with Adafactor at a learning rate of 5×10^{-5}, a batch size of 512, and a sequence length of 1216, keeping the weights and optimizer states in `float32` under `bfloat16` autocast, that is, FP32 master-weight updates in our terminology. HumanEval and HumanEval+ contain 164 problems each, so one problem corresponds to around 0.6 percentage points.

Model / training scope	Precision	Learning rate		Pass@1 (%)	
		Proxy $\hat{\eta}_1$	Training	HumanEval	HumanEval+
DeepSeek-Coder-6.7B-Base (no finetuning)	—	—	—	47.6	39.6
Magicoder-DS-6.7B	FP32	—	—	66.5	60.4
Full-depth (ours)	BF16	$2^{-11.5}$	2^{-14}	64.0	57.3
Full-parameter (ours)				64.6	59.1
Full-depth (ours)	FP32	2^{-12}	2^{-17}	64.0	57.9
Full-parameter (ours)				65.2	59.8

Table 14 reports the pass@1 metric alongside two reference values from Wei et al. (2024): the base model without finetuning, and Magicoder-DS, the released model finetuned on the same dataset. At the transferred rates, all four runs raise pass@1 from 47.6% to 64.0–65.2% on HumanEval and from 39.6% to 57.3–59.8% on HumanEval+. Their scores are within 2.5 percentage points of Magicoder-DS on HumanEval and 3.1 on HumanEval+, corresponding to at most four and five of the 164 coding problems, respectively. However, Magicoder-DS uses two training epochs with the Adafactor optimizer at a batch size of 512, whereas our runs use one epoch with the AdamW optimizer at a batch size of 32. The comparison therefore does not isolate the learning rate choice, but shows that the transferred rates yield competitive checkpoints under our own training protocol.

Two further observations follow. First, BF16 and FP32 differ by at most one solved problem for each configuration and benchmark, despite an eightfold difference in their transferred learning rates. This shows that directly using BF16 for accumulating updates can save the GPU memory with little performance degradation. Second, full-parameter training outperforms full-depth training at the same rate by one or two problems on HumanEval and three on HumanEval+. Although our analysis covers only transformer blocks, the transferred rates remain effective in this setting when all parameters are trained, without separate learning rate tuning conducted.

F Additional Related Work on PEFT

Another line of work lowers the cost of post-training: parameter-efficient finetuning (PEFT) (Liu et al., 2022; Han et al., 2024), such as low-rank adaptation (LoRA) and its variants (Hu et al., 2022; Kopiczko et al., 2024; Liu et al., 2024a), reduces the number of trainable parameters, while quantization trades numerical precision for throughput or memory (Peng et al., 2023; Dettmers et al., 2023; Li et al., 2024; Xu et al., 2024). Our work instead reduces the cost of learning rate search while updating pretrained weights in-place, without adapters.