THE RENAISSANCE OF LINE SEARCH FOR DEEP LEARNING?

Anonymous authors
Paper under double-blind review

ABSTRACT

The learning rate (LR) is a critical hyperparameter in deep neural network (DNN) training. In practice, predefined LR schedulers must be tuned through repeated trial runs, which incurs substantial human effort and hidden computational cost. Classical optimization determines the LR in a more principled way. One such way is line search, which adaptively selects the LR by checking whether a candidate step sufficiently decreases the objective. Line search is considered impractical for DNN training because it relies on accurate, and therefore expensive, evaluations of the gradient and the objective. However, line search need not be performed at every step. Established DNN schedulers provide a useful prior on the LR trajectory: the LR follows a warmup–decay shape and varies slowly within each stage. Based on this insight, we propose a line-search-based LR scheduler that runs searches only periodically on accumulated batches and reuses each selected LR until the next search. During warmup, a search may only increase the LR. During decay, a search may only decrease it, by at most one step. The resulting scheduler has a single tuned hyperparameter, the Armijo constant, and adds only about 19% wall-clock overhead on average. We evaluate it on LLM post-training with Qwen2.5-0.5B and 1.5B, covering SFT and DPO with both LoRA and full fine-tuning. With only four trials, it comes within 5% of the best of sixteen tuned (base LR, scheduler) runs on every task, substantially reducing the need for manual LR-scheduler tuning.

1 INTRODUCTION

Training deep neural networks (DNNs) boils down to non-convex and stochastic optimization. The optimization dynamics are highly sensitive to many hyperparameters. Among them, the learning rate (LR) is one of the most important hyperparameters to set. A small LR slows training down or causes optimization to stall in an unfavorable region, whereas a large one can lead to divergence or outright training failure (Kosson et al., 2024). Moreover, the optimizer normally needs to navigate very different local landscapes over the training process, so the LR that is appropriate in the early stage is different from those appropriate later.

Practitioners use predefined LR schedulers that set the LR as a function of the iteration index t. These predefined LR schedulers require two key hyperparameters: **(i) the base (peak) LR** and **(ii) the shape of the LR trajectory** (Loshchilov & Hutter, 2017; Bengio, 2012; Vaswani et al., 2017). As the appropriate choices of these two hyperparameters cannot be determined in a principled way, practitioners must launch multiple runs with different combinations of these two key hyperparameters and compare the results before selecting the most appropriate scheduler, causing a huge hidden cost. This matters in two settings. (1) Under a fixed compute budget, LR tuning consumes a large portion of the budget that could otherwise be spent on training or exploring other design choices. (2) In method comparisons, the high cost of LR tuning makes it difficult to search equally carefully for every method, including the baselines. As a result, differences in reported performance may be due to unequal tuning effort rather than differences between the methods themselves, which has already happened in some academic papers (Sivaprasad et al., 2020; Lee et al., 2026).

Classic deterministic optimization problems, especially convex problems, do not face this issue of extensive manual LR tuning, because the LR is selected by a principled procedure rather than prescribed in advance. One popular procedure is **line search** (Nocedal & Wright, 2006), which adapts the LR to the loss landscape along the update direction, rather than prescribing the LR for

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.

each step. It is implemented in most numerical optimization packages (SciPy Developers, 2024; JAXopt Developers, 2024; JuliaNLSolvers Developers, 2024b;a; R Core Team, 2024; Coppola, 2024; Okazaki, 2010; Qiu, 2024; PETSc Development Team, 2025). Applying line search has the potential to reduce the burden of carefully tuning the LR schedulers.

Practical line search evaluates candidate LRs until it finds one that satisfies an acceptance criterion, such as sufficient decrease in the objective. It often relies on accurate evaluations of the gradient and objective. In large-scale deep learning, the complexity of **accurate evaluation** of the gradient and objective grows linearly with respect to the number of data points, which can induce **huge computational cost** at modern scale. One can in principle turn to stochastic variants of line search that replace exact evaluations with noisy but cheaper minibatch estimates, but this introduces an **accuracy-cost trade-off**: performing a sufficiently accurate evaluation may require substantial computational cost. Evaluations on a few minibatches remain inexpensive, but the results will be less reliable. Severe as this trade-off may appear, we argue that **both ends** can be alleviated in modern deep learning practice:

On the accuracy end, accurate estimation is not scary. Advances in computing hardware and distributed training have enabled substantially larger batches (Ben-Nun & Hoefler, 2019; He et al., 2021; Shen et al., 2024). Take ImageNet training as an example: the batch size has increased from 128 images in AlexNet (Krizhevsky et al., 2012) to 32,768 in large-batch ResNet-50 training (You et al., 2017), representing a $256\times$ increase. Assuming i.i.d. sampling, let ξ_B denote the minibatch gradient noise. It satisfies $\sqrt{\mathbb{E}\|\xi_B\|_2^2} \propto B^{-1/2}$, where B is the batch size (McCandlish et al., 2018). Thus this increase would correspond to an approximately $16\times$ reduction in minibatch-sampling noise. A similar growth in batch size is evident in language-model pretraining: the global token batch increased from 32,768 tokens per update in GPT-1 (Radford et al., 2018) to as many as 16 million in Llama 3 (Dubey et al., 2024), an approximately $488\times$ increase. The post-training setting we study in Section 4 is low-noise, as a typical run draws batches of order 128 sequences from a narrow and homogeneous dataset of order 10^4 examples. A single batch thus already covers on the order of 1% of the full objective, which is accurate enough for a line search to be informative.

On the cost end, per-step line search is unnecessary. Established engineering practice in deep learning has produced a widely applied prior on the shape of the LR trajectory. The LR trajectory can often be divided into different stages. **(1) Within each stage, the LR changes slowly** (Loshchilov & Hutter, 2017; Smith, 2017; Naganuma et al., 2026; Hu et al., 2024; Dremov et al., 2025) or does not change (Krizhevsky et al., 2012; He et al., 2016; Goyal et al., 2017). This motivates a reconsideration of the role of line search in deep learning: rather than selecting a new LR at every step, line search should be used to periodically select an LR that can be reused for an interval, which requires less tuning budget. **(2) For the overall shape, the common LR trajectory often includes two parts: the warmup stage and the gradual decay stage.** What remains unknown, and must be tuned for each task, is the scale of the LR. Line search can complement this prior by providing specific LRs that are adaptive to the local loss landscape. Leveraging the LR's gradual or constant behavior within each stage and the overall warmup–decay pattern, together with the adaptivity of line search, we propose our method with the following **contributions**:

- **We introduce a two-phased LR scheduler based on line search**, which combines the shape prior already encoded in standard schedulers with the adaptivity of line search, at marginal extra computational overhead. Unlike predefined schedulers, which require tuning both a base LR and an LR trajectory shape, our method tunes only one hyperparameter, the Armijo constant c_1, greatly reducing the burden of hyperparameter tuning.
- **Systematic evaluation across post-training tasks.** In this paper, we choose to focus on LLM post-training. We compare our method with strong baselines: a random search over scheduler and base LR and SALSA, a per-step stochastic line search method. Our method achieves performance close to the best baseline, with a limited tuning budget.

2 Background and Related Work

2.1 Line Search and Stochastic Line Search

Let $f\colon \mathbb{R}^d \to \mathbb{R}$ be differentiable. At iteration k, line search selects a step size, or learning rate (LR), along an update direction d_k by examining the one-dimensional function $\phi(\alpha) = f(x_k + \alpha d_k)$ for

$\alpha > 0$. A common acceptance criterion is the Armijo condition (Armijo, 1966):

$$\phi(\alpha) \;\le\; \phi(0) + c_1\,\alpha\,\phi'(0), \qquad c_1 \in (0,1), \qquad \phi'(0) = \nabla f(x_k)^\top d_k. \tag{1}$$

Here, c_1 controls the amount of decrease required for a candidate LR to be accepted; a larger c_1 yields a stricter condition. The most common way to find an LR satisfying (1) is *backtracking line search*: starting from an initial candidate α_0, the LR is repeatedly shrunk as $\alpha \leftarrow \rho\,\alpha$ with $\rho \in (0,1)$ until the condition holds. Another option is *forward search*: if the initial candidate already satisfies (1), the LR is repeatedly enlarged as $\alpha \leftarrow \gamma\,\alpha$ with $\gamma > 1$ (typically $\gamma = 1/\rho$), and the search stops at the first candidate that violates the condition, returning the largest LR that satisfied it.

Stochastic line search (SLS) replaces full-batch quantities with minibatch estimates. Let B denote a minibatch consisting of b samples drawn from the dataset. At iteration k, the minibatch objective and its gradient satisfy $\mathbb{E}_{B_k}[f_{B_k}(x_k)] = f(x_k)$, $\mathbb{E}_{B_k}[\nabla f_{B_k}(x_k)] = \nabla f(x_k)$. These are unbiased estimators of the objective and the gradient. However, any finite minibatch approximation introduces approximation error. A step accepted by line search on a minibatch may not decrease the full objective. Convergence of SLS has been studied under a range of assumptions and scenarios (Paquette & Scheinberg, 2020; di Serafino et al., 2023; Lapucci & Pucci, 2025; Vaswani & Babanezhad, 2025; Collins-Woodfin et al., 2024; Vaswani et al., 2019).

On the algorithmic side, prior work has addressed stochastic noise in SLS through several complementary strategies. One line of work modifies the search criterion itself: Mahsereci & Hennig (2017) replace the deterministic Wolfe conditions with a probabilistic test based on a Gaussian-process surrogate. Galli et al. (2023) use nonmonotone acceptance tests that permit occasional increases in the minibatch loss. A second line of work reduces the uncertainty in function and gradient estimates. This includes coupling progressive batching with a stochastic line search in L-BFGS (Bollapragada et al., 2018), incorporating classical variance-reduction techniques (Jiang & Stich, 2023), and using adaptive importance sampling to reduce the variance encountered by the Armijo test (Camellini et al., 2026). Beyond standard descent methods, Vaswani et al. (2019) combine stochastic extragradient updates with a Lipschitz line search. Other frameworks extend SLS to momentum-based and more general non-descent directions (Fan et al., 2023; Lapucci & Pucci, 2026; Lapucci et al., 2026).

However, much of the existing empirical evidence remains concentrated on small-to-medium-scale benchmarks, such as matrix factorization and MNIST/CIFAR classification, leaving a gap between empirical evidence and practical performance in modern DNN training. The parabolic-approximation line search (Mutschler & Zell, 2020) was evaluated on larger-scale DNN models and datasets, including ResNet-50 and DenseNet-121 on ImageNet, but the authors later reported, in a retrospective note added to the arXiv version, that it generally does not work well with PyTorch's default initialization. Its performance in LLM post-training remains unexamined. More recently, Kenneweg et al. (2024) proposed SALSA, which applies exponential moving averages (EMA) to both sides of the Armijo condition to reduce the noise level and was evaluated on modern DNN training tasks. SALSA is the most closely related competing method to our approach. Our approach takes a complementary direction: we leverage the structural priors of common deep-learning LR trajectories and perform line searches periodically to identify an appropriate LR range, rather than determining LR at every step.

2.2 LR SCHEDULERS

Most schedulers predetermine the LR trajectory before training, independently of the observed objective. **Decaying** schedulers anneal monotonically from a base LR using exponential decay (Bengio, 2012), linear decay to zero (Defazio et al., 2023; Bergsma et al., 2025), or cosine annealing, which has become a standard recipe for large-scale language-model training (Brown et al., 2020; Touvron et al., 2023). **Piecewise-constant schedulers** such as StepLR reduce the LR at prescribed points and have been widely used in CNN training (Krizhevsky et al., 2012; He et al., 2016; Goyal et al., 2017). **Metric-driven scheduling.** `ReduceLROnPlateau` (Paszke et al., 2019) produces a piecewise-constant LR trajectory, but uses a monitored metric to determine when to reduce the LR. Although closer in spirit to our method, it still requires a base LR and a plateau-detection rule.

These schedulers have something in common: they often gradually decay from the peak LR, and change slowly or remain constant over intervals within each phase. Combined with the commonly used warmup, they encode a strong prior. They do not, however, determine the task-specific LR scale. We use this warmup–decay prior to constrain the overall trajectory, while periodic line searches select its scale from the observed loss landscape.

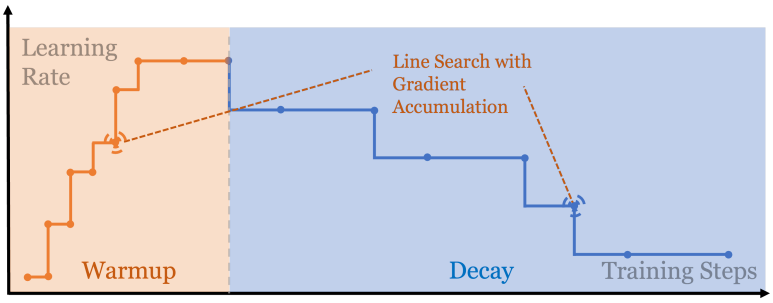


Figure 1: Overview of the typical shape of the proposed line search scheduler. The scheduler is divided into the warmup stage and decay stage. At each prescribed anchor point, line search is performed on an accumulated batch and the resulting LR is reused until the next anchor point.

2.3 Tuning-Free Optimizers

A related line of work seeks to avoid task-specific LR sweeps. Methods such as D-Adaptation (Defazio & Mishchenko, 2023), DoG and DoWG (Ivgi et al., 2023; Khaled et al., 2023), and Prodigy (Mishchenko & Defazio, 2024) estimate the unknown distance D, which determines the optimal step size or LR $\frac{D}{G\sqrt{T}}$ for nonsmooth convex optimization. These methods avoid an explicit LR sweep in deep learning. However, they do not guarantee consistently near-optimal empirical performance across tasks relative to a separately tuned Adam baseline (Pang et al., 2026). We observe the same in our experiments in Appendix H. This gap is especially problematic for tuning-free methods: task-specific tuning is undefined, leaving no clear way to improve it.

3 Methods

3.1 An LR Trajectory Shape Prior for Line Search

Line search selects an LR that is favorable in the short term, but this choice can be suboptimal over the full training run (Roulet et al., 2024; Wu et al., 2018). We argue that this suboptimality, caused by the preference for **local** gain, can be mitigated by equipping it with a prior on the **global** shape of the LR trajectory, which, as we argue below, reflects the overall structure of the loss landscape that the optimizer traverses during training.

Widely adopted LR schedulers typically consist of two essential components: an initial warmup phase and a final decay phase (Naganuma et al., 2026). The popularity of both components stems from the geometry of the DNN loss landscape: **Warmup.** Warmup increases the LR linearly from zero to the base LR at the beginning of training. At initialization, the network often lies in a sharp, high-loss region where a large LR can easily cause divergence. This instability arises partly from unreliable moment estimates in adaptive optimizers such as Adam during the early steps (Liu et al., 2020), and partly from excessively large update magnitudes (Ma & Yarats, 2021). Warmup gradually steers the network toward flatter regions that can tolerate larger LRs (Kalra & Barkeshli, 2024). This transition is beneficial: it accelerates optimization and, in some settings, improves generalization (Lewkowycz et al., 2020). **Decay.** In the later stages of training, the loss landscape resembles a deep valley with a river running along its floor (Wen et al., 2025). A large LR makes rapid progress along the river direction but causes the iterate to oscillate between the valley walls. Decaying the LR damps these oscillations, allowing the iterate to settle toward the river bottom and further reducing the loss.

Taken together, these observations indicate that the warmup–decay shape is more than a heuristic: it mirrors the overall landscape of DNN training, in which the optimizer starts in a sharp, high-loss hill, moves into a river valley, and finally descends to the valley floor. We therefore adopt the warmup–decay shape as a prior on the LR trajectory. Guided by this prior, we modify classical line search so that its local LR selections follow the global shape of the trajectory, rather than being driven solely by limited local landscape information.

3.2 TWO-PHASED PERIODIC LINE SEARCH

Line search is performed only at pre-scheduled iterations; each search yields an **anchor LR** that is adopted immediately and used for every step until the next line search, producing a piecewise constant LR trajectory. We also need to specify T_{wmp} that splits training into a **warmup phase** ($t < T_{\text{wmp}}$), in which a search may only raise the LR, and a **decay phase** ($t \geq T_{\text{wmp}}$), in which a search may only lower it, and by at most one step to avoid sudden drops in the LR due to a single inaccurate search. The trajectory is therefore non-decreasing in the first phase and non-increasing in the second, which creates the warmup–decay shape of Section 3.1. Figure 1 shows the resulting overall shape; Algorithm 1 gives the full loop and Algorithm 2 specifies the rule applied at an anchor.

Notation Let θ_t be the parameters at iteration t and let d_t be the search direction. An ordinary step is $\theta_{t+1} = \theta_t + \alpha d_t$, where $d_t = -\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} - \lambda\theta_t$. Here $\hat{m}_t, \hat{v}_t$ are the bias-corrected first and second moments in Adam respectively. Searches are evaluated on a **search batch** $\hat{V}$ formed by accumulating N_{accum} minibatches, which lowers the noise of both the loss $f_{\hat{V}}$ and the directional derivative $\phi'_0 = \nabla f_{\hat{V}}(\theta_t)^\top d_t$. Since d_t need not be a descent direction, whenever $\phi'_0 \geq 0$ at an anchor we fall back to $d_t = -\nabla f_{\hat{V}}(\theta_t)$ (Lapucci & Pucci, 2026). All candidate LRs lie on the grid $\{\rho^k\alpha : k \in \mathbb{Z}\}$ with the backtracking factor $\rho \in (0, 1)$ and growth factor $\gamma = 1/\rho$; moving to $\gamma\alpha$ or $\rho\alpha$ is one step up or down. We use $\rho = 0.5$, the standard choice for backtracking line search (Nocedal & Wright, 2006). The user specifies the number of line searches for the warmup and decay stages, giving N_{wmp} and N_{dec}. The anchors are spread evenly within each phase; the intervals

$$\Delta^{\text{ls}}_{\text{wmp}} = \left\lceil \frac{T_{\text{wmp}}}{N_{\text{wmp}}} \right\rceil, \qquad \Delta^{\text{ls}}_{\text{dec}} = \left\lfloor \frac{T - T_{\text{wmp}}}{N_{\text{dec}}} \right\rfloor \tag{2}$$

are derived from the budget. The number of searches, and hence the extra forward passes they cost, is fixed in advance and does not grow with T, so the relative overhead of the scheduler decreases as runs get longer. Section 4 reports the measured wall-clock overhead.

Algorithm 1: Two-phased periodic line-search scheduler (full training loop)

Input: initial parameters θ_0; initial LR α_0; total iterations T; warmup length T_{wmp}; search budget $N_{\text{wmp}}, N_{\text{dec}}$; accumulation factor N_{accum}; Armijo constant c_1; backtracking factor $\rho \in (0, 1)$; growth factor $\gamma = 1/\rho$

Output: trained parameters θ_T

$\Delta^{\text{ls}}_{\text{wmp}} \leftarrow \lceil T_{\text{wmp}}/N_{\text{wmp}} \rceil$; $\Delta^{\text{ls}}_{\text{dec}} \leftarrow \lfloor (T - T_{\text{wmp}})/N_{\text{dec}} \rfloor$
$\alpha \leftarrow \alpha_0$
for $t = 0, \ldots, T-1$ **do**
  if $t < T_{\text{wmp}}$ **then**
    phase $\leftarrow$ WARMUP; $\Delta^{\text{ls}} \leftarrow \Delta^{\text{ls}}_{\text{wmp}}$; $s \leftarrow t$
  else
    phase $\leftarrow$ DECAY; $\Delta^{\text{ls}} \leftarrow \Delta^{\text{ls}}_{\text{dec}}$; $s \leftarrow t - T_{\text{wmp}}$
  end
  if $t > 0$ and $s \bmod \Delta^{\text{ls}} = 0$ **then**
    $\hat{V} \leftarrow$ accumulated N_{accum} minibatches
    $g \leftarrow \nabla f_{\hat{V}}(\theta_t)$; $d_t \leftarrow$ optimizer direction from g
    $\phi'_0 \leftarrow g^\top d_t$
    $\alpha \leftarrow$ `PhaseSearch` $(\theta_t, d_t, \hat{V}, \phi'_0, \alpha, c_1, \rho, \gamma, \text{phase})$
  else
    / Ordinary iteration: the LR from the last anchor is reused. */*
    sample minibatch B_t; $g \leftarrow \nabla f_{B_t}(\theta_t)$; $d_t \leftarrow$ optimizer direction from g
  end
  $\theta_{t+1} \leftarrow \theta_t + \alpha\, d_t$
end
return θ_T

Initialization The scale of the LR is set by a single search at $t = 0$: starting from the ceiling $\alpha_{\max}$, we perform a backtracking line search to find the largest LR that satisfies the Armijo condition (1) on $\hat{V}$. Every later search moves the LR from its current value, so no search restarts from a fixed value. We use $\alpha_{\max} = 0.1$ and $\alpha_{\min} = 10^{-8}$ on every task.

Algorithm 2: PHASESEARCH: the phase-restricted rule applied at one anchor

Input: iterate θ; direction d; search batch $\hat{V}$; directional derivative $\phi'_0 = \nabla f_{\hat{V}}(\theta)^\top d$; current LR α; Armijo constant c_1; backtracking factor ρ and growth factor $\gamma = 1/\rho$; phase $\in \{\text{WARMUP}, \text{DECAY}\}$
Output: updated LR α

$f_0 \leftarrow f_{\hat{V}}(\theta)$
if phase $=$ WARMUP **then**
  / Forward Search to increase the LR */*
  while $f_{\hat{V}}(\theta + \gamma\alpha d) \le f_0 + c_1\gamma\alpha\phi'_0$ **do**
    $\alpha \leftarrow \gamma\alpha$
  end
else
  / capped at one backtracking step. Here, $\hat{V}$ may optionally be replaced by a held-out batch V_{val}. */*
  if $f_{\hat{V}}(\theta + \rho\alpha d) < f_{\hat{V}}(\theta + \alpha d)$ **then**
    $\alpha \leftarrow \rho\alpha$
  end
end
return α

Warmup phase: forward-only search. At a warmup anchor we allow **only forward search**. Starting from the current LR α, we **iteratively** test the enlarged candidate $\gamma\alpha$ against the Armijo condition (1). Concretely, the search returns $\gamma^k\alpha$ where $\gamma^{k+1}\alpha$ is the first rejected candidate. If the first candidate $\gamma\alpha$ is already rejected, the search returns α unchanged, so the LR can only increase or stay the same, which is a deliberate relaxation of the classic line search, as the warmup stage encourages large LRs.

Gradual decay phase: one backtracking step per anchor. At a decay anchor we allow **only backtracking**, and we cap the reduction at **a single grid step** to enforce a gradual decay. We evaluate the loss at the current LR α and at the reduced candidate $\rho\alpha$, and keep whichever is lower. In contrast to the warmup rule, this comparison is **not iterated**, so a decay anchor costs exactly two evaluations and the LR decreases by at most a factor of ρ. The cap guards against overly rapid decay caused by noise in gradient and objective evaluation. Without it, a backtracking loop could apply several reductions at one anchor based on one unreliable line search result, collapsing the LR long before training ends and stalling progress. If a larger decay is genuinely needed, it takes place over several anchors, which keeps the decay gradual. Since this comparison does not involve the Armijo condition, the constant c_1 acts only during warmup.

Search Scheduling and Evaluation Batches The loss landscape and optimizer state change rapidly in the early stage of training and more slowly later. Even splitting the anchors so as to set $\Delta^{ls}_{\text{wmp}} = \Delta^{ls}_{\text{dec}}$ cannot capture the rapidly changing early-training dynamics. Therefore, we place anchors more densely during warmup, so that $\Delta^{ls}_{\text{wmp}} \ll \Delta^{ls}_{\text{dec}}$. Because the warmup phase only takes a small fraction of training, this is not computationally expensive. In our experiments, we use $N_{\text{wmp}} = 5$ and $N_{\text{dec}} = 20$, which yield a smaller Δ^{ls}_{wmp} under Eq. (2). At each anchor, we accumulate N_{accum} minibatches into $\hat{V}$ for line search, further reducing the noise level in gradient and objective evaluation. As an optional extension to prevent overfitting, in the decay phase, line search can be easily modified to evaluate the objective on a held-out batch from the instead of the training set. We provide an ablation study of these design choices in Appendix D.

3.3 ANALYSIS OF HYPERPARAMETER SENSITIVITY

The procedure introduces several design parameters, including the warmup and decay search numbers, N_{wmp} and N_{dec}; the warmup duration, T_{wmp}; and the accumulation factor, N_{accum}. But conceptually, we expect limited sensitivity within suitable ranges for the following reasons.

N_{wmp} and N_{dec} determine when searches occur rather than the LR values they select. Their choice is therefore largely one-sided: too few searches can miss changes in the optimization dynamics, whereas additional searches mostly re-confirm the current LR. This makes a suitable value easier to find than a base LR, which must be neither too small nor too large.

For N_{accum}, let $B = |\hat{V}|$ denote the number of examples in the accumulated search batch. As discussed in Section 1, under independent sampling, the noise level of a batch estimate decreases at a rate of $O(B^{-1/2})$. We therefore expect diminishing returns as B grows: once the estimates are stable enough that additional samples rarely change the line-search decision, further accumulation increases computation with little effect on the selected learning rate. We expect the minibatches used in many modern deep-learning settings to be sufficiently large that only modest accumulation is needed.

We have also measured the sensitivity to these hyperparameters empirically. The results in Section 4 show limited sensitivity to these choices over the tested ranges. The main parameter we tune is the **Armijo constant** c_1, yielding a smaller tuning space than predefined schedulers.

4 EXPERIMENTS

We evaluate our scheduler on LLM post-training, for two main reasons. First, **post-training shows highly heterogeneous LR ranges.** The objective may be supervised fine-tuning (SFT), preference optimization (DPO), or an RL objective, and the parameterization may be full fine-tuning (FT) or low-rank adaptation (LoRA). Each combination requires a different range of LR, making post-training a demanding scenario for testing an adaptive LR scheduler. Second, **post-training is more practical than training a DNN model from scratch**. Since current foundation models are already strong, the common workflow is to adapt an existing foundation model to a target domain through post-training.

Experimental Protocol The question this experiment asks is how much tuning budget a method would consume, on a problem it has not been tuned for, before the model is trained well. We therefore compare LR schedulers under a constrained tuning budget. For each method, we perform random search over its own hyperparameter space and track the best performance achieved as a function of cumulative wall-clock time. We allow four values for each hyperparameter a method tunes, so a method tuning two of them gets 16 trials and a method tuning one gets 4; the drawn values are listed in Appendix A. To rule out obviously divergent trials, we monitor every trial during training and terminate it at the first step where either (i) the running median of the training loss over a 10-step window reaches three times its initial value, or (ii) the evaluation loss fails to improve on its initial value for six consecutive evaluations.

For the baselines, we jointly tune the base LR, sampled log-uniformly from $(10^{-7}, 10^{-2})$, and the scheduler type, chosen from $\{\text{cosine}, \text{linear}, \text{inverse-square-root}\}$, which gives $4 \times 4 = 16$ trials. For line search, the only tunable hyperparameter is the Armijo constant $c_1 \in (0, 1)$, drawn uniformly, which gives four trials. As a third method we include SALSA (Kenneweg et al., 2024), the stochastic line search most closely related to ours (Section 2.1). We tune its Armijo constant c_1 on the same 4-trial budget; its remaining hyperparameter, the EMA coefficient β_3, is reported by the authors to be insensitive within a broad range, and every other setting is left at their defaults.

We take the base LR and the scheduler type to be the two hyperparameters that matter most for a predefined scheduler, and we search their joint space thoroughly. Although there are arguments that the performance may well be less sensitive to the scheduler type than to the base LR (Naganuma et al., 2026), it does no harm to the baseline as we perform random search rather than grid search. Every one of the 16 trials still draws a base LR from the full log-uniform range, so no budget is spent enumerating combinations of a hyperparameter the objective is insensitive to (Bergstra & Bengio, 2012). The range itself is wide because the setting we study is one the method has not been tuned for, where no engineering experience narrows the usable LR band in advance. Appendix F repeats the comparison for a practitioner who does have such experience, with the baseline restricted to one decade either side of a per-case reference LR. Even so, most of our conclusions still hold.

We use the Qwen2.5 base models for SFT and the Qwen2.5 Instruct models for DPO, evaluating both 0.5B and 1.5B variants to assess performance across model sizes. All trials use AdamW. Within each dataset, the baseline and line-search methods share the same experimental settings and differ only in how the learning rate is determined. We split the datasets into a training set and an evaluation set. For the decay stage in our method, we split a held-out set from the training set as the search batch. **SFT datasets: Alpaca** (Taori et al., 2023) is a general-purpose dataset of 52K instruction-following demonstrations generated by `text-davinci-003`. **OpenMathInstruct-2 (OM2)** (Toshniwal et al., 2025) is a mathematical problem-solving dataset of problem–solution pairs synthesized from the GSM8K and MATH training sets. **DPO datasets: UltraFeedback (UF)** (Cui et al., 2024) provides

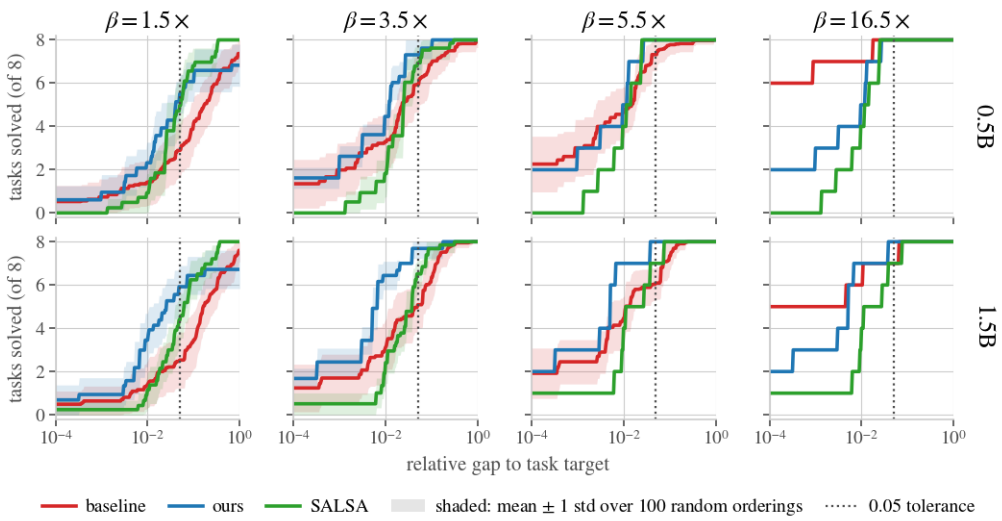


Figure 2: β-relative minimization profiles over the $n = 8$ post-training tasks, for budgets $\beta \in \{1.5, 3.5, 5.5, 16.5\}$ measured in units of one baseline trial. Higher is better. The dotted line marks a 5% tolerance to the per-task target. Shaded bands span random orderings of each method's trials; every trial is a single training run.

61K general instruction-following preference pairs ranked by GPT-4. **Math-Step-DPO-10K** (Lai et al., 2024) provides 10K mathematical-reasoning pairs.

β-Relative Minimization Profiles (β-RMPs) We report β-RMPs (Curtis et al., 2017) to compare methods across a heterogeneous set of problems. For each task, we define the target as the lowest evaluation loss attained by any of the compared methods. For a fixed β, each profile shows the fraction of tasks on which a method finds a solution that matches the target of that task, subject to a computational budget of β times the corresponding reference budget. For each task, we define the reference budget as the wall-clock time required for one full baseline trial on that task. Comparing profiles across values of β shows how performance changes as the available budget increases. A point (a, b) on a curve means that the method finds a solution within a tolerance a of the target, i.e. with a relative gap of at most a, on b of the $n = 8$ tasks. Higher curves indicate better performance. The dotted vertical line marks a tolerance of 5%, which we use throughout as a practical criterion for "matching the target". We report evaluation loss rather than downstream benchmark accuracy, following the established protocol for evaluating training algorithms (Shallue et al., 2019; Choi et al., 2019; Kasimbeg et al., 2025). Lower evaluation loss does not necessarily imply better model quality, particularly for DPO (Pal et al., 2024; Razin et al., 2025). Appendix G examines the relationship between DPO loss and preference accuracy on the test set across our trials. Within a run, selecting the checkpoint by loss rather than by accuracy costs at most 0.04 in preference accuracy.

Our method can match the best-tuned performance with less tuning. At $\beta = 16.5$, the baseline has fully explored its search space and therefore attains very strong performance. Measured at the 5% tolerance, however, our method already solves all 8 tasks in every one of the 100 orderings at $\beta = 5.5$ on 0.5B and at $\beta = 3.5$ on 1.5B, while the baseline solves 7.3 and 5.1 of the 8 tasks on average at the same budgets. **When the tuning budget is low, our method is the better choice.** On 1.5B, from $\beta = 1.5$ to $\beta = 5.5$ our method covers more tasks at every tolerance up to 5%. On 0.5B it holds at the two smallest budgets, $\beta = 1.5$ and $\beta = 3.5$. At $\beta = 5.5$ on 0.5B, the baseline is ahead for tolerances below $\approx 10^{-2}$. Our method is better for tolerances above 10^{-2} and reaches 100% at a stricter tolerance than the baseline. SALSA is competitive once the tolerance is loose, but falls below the baseline in the high-precision region.

Because the warmup search can only raise the LR, our method favors large LRs. For the same c_1, its peak LR is 5.8× SALSA's at the median. Large LRs can speed up DNN training, but they also risk divergence: 9 of our 32 trials diverge at 0.5B and 6 of 32 at 1.5B, whereas SALSA never diverges.

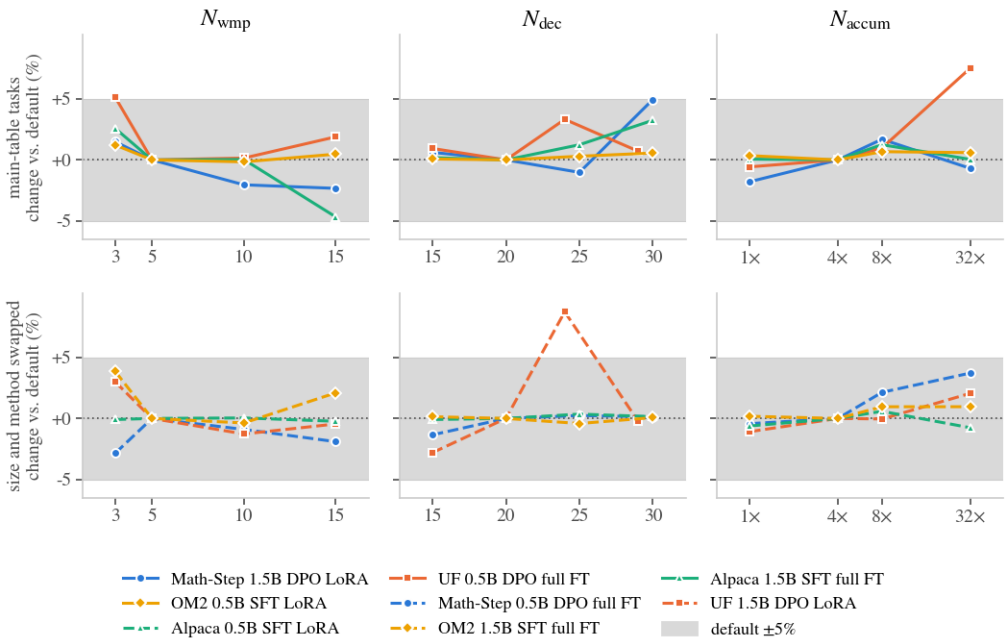


Figure 3: The line search is insensitive to its own cadence. Each panel varies one integer while the other two stay at the default; the vertical axis is the change in evaluation loss relative to each task's default run, so the eight tasks share one scale. The gray band is $\pm 5\%$, a fixed reference rather than an error bar: every trial uses one seed. 69 of the 72 non-default configurations fall inside it.

Counting all trials, $17/32$ of ours at 0.5B and $21/32$ at 1.5B fall within the 5% tolerance, against $20/32$ and $18/32$ for SALSA, i.e. 38 of 64 for both methods. Divergent trials, however, are cheap: the stopping rule of our protocol terminates them early. Among the trials that run to completion, $17/23$ of ours at 0.5B and $21/26$ at 1.5B fall within the tolerance, against SALSA's $20/32$ and $18/32$. A trial that misses the tolerance, under either method, consumes its full training budget.

Cost of the searches. Averaged across the sixteen tasks, the two line searches have nearly identical costs relative to a fixed-scheduler trial: $1.19\times$ for ours and $1.20\times$ for SALSA, with worst cases of $1.37\times$ and $1.26\times$. Our highest-overhead cases are all DPO tasks, which reflects their shorter trials rather than more expensive searches. Appendix C reports the wall-clock time of every task.

Sensitivity Analysis The scheduler in Section 3.2 requires three integers to be set before training: N_{wmp}, N_{dec}, and N_{accum}. We vary one at a time across eight experimental settings spanning SFT and DPO, 0.5B and 1.5B models, and LoRA and full fine-tuning. Specifically, we consider $N_{\text{wmp}} \in \{3, 5, 10, 15\}$, $N_{\text{dec}} \in \{15, 20, 25, 30\}$, and $N_{\text{accum}} \in \{1, 4, 8, 32\}$. Figure 3 shows the percentage change relative to each setting's default. Most changes fall within 5%, suggesting that the default is reasonably robust. For c_1, because our method encourages large LRs, a stricter line-search condition helps constrain the acceptable range.

5 Conclusion

In this paper, we combine line search with well-established heuristics from modern deep learning to develop an adaptive learning-rate scheduler that generalizes across diverse training settings. Our method substantially reduces the need for hyperparameter tuning. Across a range of LLM post-training tasks, the proposed scheduler achieves performance close to that of carefully tuned predefined schedulers while requiring a much smaller tuning space. These results suggest that line search, combined with deep learning LR trajectory priors, can be a practical option when the budget available for LR tuning is limited. It is a technique that deserves serious consideration as a way to cut the cost of hyperparameter tuning.

REFERENCES

Larry Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of Mathematics*, 16(1):1–3, 1966.

Tal Ben-Nun and Torsten Hoefler. Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys*, 52(4):1–43, 2019. doi: 10.1145/3320060. URL https://dl.acm.org/doi/10.1145/3320060.

Yoshua Bengio. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*, pp. 437–478. Springer, 2012.

Shane Bergsma, Nolan Dey, Gurpreet Gosal, Gavia Gray, Daria Soboleva, and Joel Hestness. Straight to zero: Why linearly decaying the learning rate to zero works best for LLMs. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=hrOlBgHsMI.

James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13(1):281–305, 2012.

Raghu Bollapragada, Jorge Nocedal, Dheevatsa Mudigere, Hao-Jun Michael Shi, and Ping Tak Peter Tang. A progressive batching L-BFGS method for machine learning. In *International Conference on Machine Learning*, pp. 620–629. PMLR, 2018. URL https://proceedings.mlr.press/v80/bollapragada18a.html.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, 2020. URL https://papers.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Filippo Camellini, Serena Crisci, Anna De Magistris, and Giorgia Franchini. A line-search based SGD algorithm with adaptive importance sampling. *Journal of Computational and Applied Mathematics*, 477:117120, 2026. doi: 10.1016/j.cam.2025.117120.

Dami Choi, Christopher J. Shallue, Zachary Nado, Jaehoon Lee, Chris J. Maddison, and George E. Dahl. On empirical comparisons of optimizers for deep learning. *arXiv preprint arXiv:1910.05446*, 2019. URL https://arxiv.org/abs/1910.05446.

Elizabeth Collins-Woodfin, Inbar Seroussi, Begoña García Malaxechebarría, Andrew W. Mackenzie, Elliot Paquette, and Courtney Paquette. The high line: Exact risk and learning rate curves of stochastic adaptive learning rate algorithms. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/0c7ca207a051228f978971447a56464a-Abstract-Conference.html.

Antonio Coppola. lbfgs: Optimize function using libLBFGS library. R package documentation, 2024. URL https://www.rdocumentation.org/packages/lbfgs/versions/1.2.1.2/topics/lbfgs.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting language models with scaled AI feedback. In *International Conference on Machine Learning*, 2024.

Frank E Curtis, Tim Mitchell, and Michael L Overton. A BFGS-SQP method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles. *Optimization Methods and Software*, 32(1):148–181, 2017.

Aaron Defazio and Konstantin Mishchenko. Learning-rate-free learning by D-adaptation. In *International Conference on Machine Learning*, pp. 7449–7479, 2023. URL https://proceedings.mlr.press/v202/defazio23a.html.

Aaron Defazio, Ashok Cutkosky, Harsh Mehta, and Konstantin Mishchenko. Optimal linear decay learning rate schedules and further refinements. *arXiv preprint arXiv:2310.07831*, 2023. URL https://arxiv.org/abs/2310.07831.

Daniela di Serafino, Nataša Krejić, Nataša Krklec Jerinkić, and Marco Viola. LSOS: Line-search second-order stochastic optimization methods for nonconvex finite sums. *Mathematics of Computation*, 92(341):1273–1299, 2023. URL https://arxiv.org/abs/2007.15966.

Aleksandr Dremov, Alexander Hägele, Atli Kosson, and Martin Jaggi. Training dynamics of the cooldown stage in warmup-stable-decay learning rate scheduler. *Transactions on Machine Learning Research*, 2025. URL https://arxiv.org/abs/2508.01483.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024. URL https://arxiv.org/abs/2407.21783.

Chen Fan, Sharan Vaswani, Christos Thrampoulidis, and Mark Schmidt. MSL: An adaptive momentem-based stochastic line-search framework. In *OPT 2023: Optimization for Machine Learning*, 2023.

Leonardo Galli, Holger Rauhut, and Mark Schmidt. Don't be so monotone: Relaxing stochastic line search in over-parameterized models. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/6d0bf1265ea9635fb4f9d56f16d7efb2-Abstract-Conference.html.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch SGD: Training ImageNet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017. URL https://arxiv.org/abs/1706.02677.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Xiaoxin He, Fuzhao Xue, Xiaozhe Ren, and Yang You. Large-scale deep learning optimizations: A comprehensive survey. *arXiv preprint arXiv:2111.00856*, 2021. URL https://arxiv.org/abs/2111.00856.

Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, et al. MiniCPM: Unveiling the potential of small language models with scalable training strategies. *arXiv preprint arXiv:2404.06395*, 2024.

Maor Ivgi, Oliver Hinder, and Yair Carmon. DoG is SGD's best friend: A parameter-free dynamic step size schedule. In *International Conference on Machine Learning*, pp. 14465–14499, 2023. URL https://proceedings.mlr.press/v202/ivgi23a.html.

JAXopt Developers. Line search — JAXopt documentation. JAXopt documentation, 2024. URL https://jaxopt.github.io/stable/line_search.html.

Xiaowen Jiang and Sebastian U. Stich. Adaptive SGD with Polyak stepsize and line-search: Robust convergence and variance reduction. In *Advances in Neural Information Processing Systems*, volume 36, pp. 26396–26424, 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/540eb9e0ee35d525231c3fd22d1dcbf2-Abstract-Conference.html.

JuliaNLSolvers Developers. Line search routines — LineSearches.jl documentation. LineSearches.jl documentation, 2024a. URL https://julianlsolvers.github.io/LineSearches.jl/latest/reference/linesearch.html.

JuliaNLSolvers Developers. Linesearch — Optim.jl documentation. Optim.jl documentation, 2024b. URL https://julianlsolvers.github.io/Optim.jl/v0.13/algo/linesearch/.

Dayal Singh Kalra and Maissam Barkeshli. Why warmup the learning rate? Underlying mechanisms and improvements. In *Advances in Neural Information Processing Systems*, 2024. URL https://arxiv.org/abs/2406.09405.

Priya Kasimbeg, Frank Schneider, Runa Eschenhagen, Juhan Bae, Chandramouli Shama Sastry, Mark Saroufim, Boyuan Feng, Less Wright, Edward Z Yang, Zachary Nado, et al. Accelerating neural network training: An analysis of the AlgoPerf competition. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2502.15015.

Philip Kenneweg, Tristan Kenneweg, Fabian Fumagalli, and Barbara Hammer. No learning rates needed: Introducing SALSA – Stable Armijo Line Search Adaptation. In *International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2024.

Ahmed Khaled, Konstantin Mishchenko, and Chi Jin. DoWG unleashed: An efficient universal parameter-free gradient descent method. In *Advances in Neural Information Processing Systems*, volume 36, pp. 6748–6769, 2023.

Atli Kosson, Bettina Messmer, and Martin Jaggi. Analyzing & reducing the need for learning rate warmup in GPT training. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/059445c2d5b3ef918079851628fef1d6-Abstract-Conference.html.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, volume 25, 2012.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-DPO: Step-wise preference optimization for long-chain reasoning of LLMs. *arXiv preprint arXiv:2406.18629*, 2024.

Matteo Lapucci and Davide Pucci. Convergence conditions for stochastic line search based optimization of over-parametrized models. *Optimization*, pp. 1–20, 2025.

Matteo Lapucci and Davide Pucci. Effectively leveraging momentum terms in stochastic line search frameworks for fast optimization of finite-sum problems. *Computational Optimization and Applications*, 2026. doi: 10.1007/s10589-026-00776-2. arXiv:2411.07102.

Matteo Lapucci, Giampaolo Liuzzi, Stefano Lucidi, Davide Pucci, and Marco Sciandrone. A globally convergent gradient method with momentum. *Computational Optimization and Applications*, 93 (2):795–820, 2026.

Yu-Ang Lee, Ching-Yun Ko, Pin-Yu Chen, and Mi-Yen Yeh. Learning rate matters: Vanilla LoRA may suffice for LLM fine-tuning. *arXiv preprint arXiv:2602.04998*, 2026.

Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: The catapult mechanism. *arXiv preprint arXiv:2003.02218*, 2020. URL https://arxiv.org/abs/2003.02218.

Liyuan Liu, Haoming Jiang, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Jiawei Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020. URL https://arxiv.org/abs/1908.03265.

Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://arxiv.org/abs/1608.03983.

Jerry Ma and Denis Yarats. On the adequacy of untuned warmup for adaptive optimization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021. URL https://arxiv.org/abs/1910.04209.

Maren Mahsereci and Philipp Hennig. Probabilistic line searches for stochastic optimization. *Journal of Machine Learning Research*, 18(119):1–59, 2017.

Sam McCandlish, Jared Kaplan, Dario Amodei, and OpenAI Dota Team. An empirical model of large-batch training. *arXiv preprint arXiv:1812.06162*, 2018. URL https://arxiv.org/abs/1812.06162.

Konstantin Mishchenko and Aaron Defazio. Prodigy: An expeditiously adaptive parameter-free learner. In *International Conference on Machine Learning*, pp. 35779–35804. PMLR, 2024.

Maximus Mutschler and Andreas Zell. Parabolic approximation line search for DNNs. In *Advances in Neural Information Processing Systems*, volume 33, pp. 5405–5416, 2020.

Hiroki Naganuma, Atish Agarwala, Priya Kasimbeg, and George E Dahl. What do near-optimal learning rate schedules look like? *arXiv preprint arXiv:2603.10301*, 2026.

Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

Naoaki Okazaki. libLBFGS: A library of limited-memory BFGS optimization. Project documentation, 2010. URL https://www.chokkan.org/software/liblbfgs/.

Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddartha Naidu, and Colin White. Smaug: Fixing failure modes of preference optimisation with DPO-positive. *arXiv preprint arXiv:2402.13228*, 2024.

Yijiang Pang, Shuyang Yu, Bao Hoang, and Jiayu Zhou. Towards reliability of parameter-free optimization. In *Proceedings of the 2026 SIAM International Conference on Data Mining (SDM)*, 2026. arXiv:2405.04376.

Courtney Paquette and Katya Scheinberg. A stochastic line search method with expected complexity analysis. *SIAM Journal on Optimization*, 30(1):349–376, 2020.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, volume 32, 2019. URL https://arxiv.org/abs/1912.01703.

PETSc Development Team. Optimization line search (TaoLineSearch) — PETSc documentation. PETSc documentation, 2025. URL https://petsc.org/release/manualpages/TaoLineSearch/.

Yixuan Qiu. LBFGS++: A header-only C++ library for L-BFGS and L-BFGS-B algorithms. Project documentation, 2024. URL https://lbfgspp.statr.me/.

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

R Core Team. optim {stats}: General-purpose optimization — R documentation. CRAN R documentation, 2024. URL https://search.r-project.org/CRAN/refmans/stats/html/optim.html.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. OpenAI technical report, 2018. URL https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

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 *Advances in Neural Information Processing Systems*, volume 36, 2023.

Noam Razin, Sadhika Malladi, Adithya Bhaskar, Danqi Chen, Sanjeev Arora, and Boris Hanin. Unintentional unalignment: Likelihood displacement in direct preference optimization. In *International Conference on Learning Representations*, 2025.

Vincent Roulet, Atish Agarwala, Jean-Bastien Grill, Grzegorz Swirszcz, Mathieu Blondel, and Fabian Pedregosa. Stepping on the edge: Curvature aware learning rate tuners. In *Advances in Neural Information Processing Systems*, volume 37, pp. 47708–47740, 2024.

SciPy Developers. scipy.optimize.line_search — SciPy manual. SciPy documentation, 2024. URL https://docs.scipy.org/doc/scipy-1.12.0/reference/generated/scipy.optimize.line_search.html.

Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, and George E. Dahl. Measuring the effects of data parallelism on neural network training. *Journal of Machine Learning Research*, 20(112):1–49, 2019. URL https://jmlr.org/papers/v20/18-789.html.

Li Shen, Yan Sun, Zhiyuan Yu, Liang Ding, Xinmei Tian, and Dacheng Tao. On efficient training of large-scale deep learning models. *ACM Computing Surveys*, 57(3):1–36, 2024. doi: 10.1145/3700439. URL https://dl.acm.org/doi/10.1145/3700439.

Prabhu Teja Sivaprasad, Florian Mai, Thijs Vogels, Martin Jaggi, and François Fleuret. Optimizer benchmarking needs to account for hyperparameter tuning. In *International Conference on Machine Learning*, pp. 9036–9045. PMLR, 2020.

Leslie N Smith. Cyclical learning rates for training neural networks. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 464–472. IEEE, 2017.

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

Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. OpenMathInstruct-2: Accelerating AI for math with massive open-source instruction data. In *International Conference on Learning Representations*, 2025.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023. URL https://arxiv.org/abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://papers.nips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Sharan Vaswani and Reza Babanezhad. Armijo line-search can make (stochastic) gradient descent provably faster. In *International Conference on Machine Learning*, pp. 61035–61070. PMLR, 2025. URL https://arxiv.org/abs/2503.00229.

Sharan Vaswani, Aaron Mishkin, Issam Laradji, Mark Schmidt, Gauthier Gidel, and Simon Lacoste-Julien. Painless stochastic gradient: Interpolation, line-search, and convergence rates. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Kaiyue Wen, Zhiyuan Li, Jason S. Wang, David Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape view. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2410.05192.

Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. In *International Conference on Learning Representations*, 2018. arXiv:1803.02021.

Yang You, Igor Gitman, and Boris Ginsburg. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017. URL https://arxiv.org/abs/1708.03888.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, 2024.

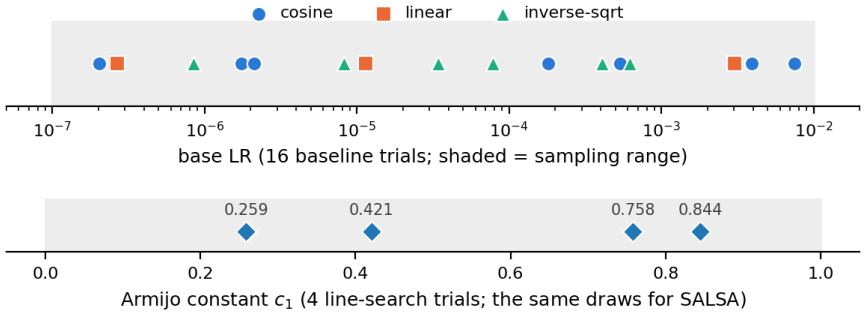


Figure 4: The hyperparameter values tried by each method, identical on every task. Top: the 16 baseline trials, each a (scheduler, base LR) pair; the base LR is log-uniform on the shaded range and the marker gives the scheduler. Bottom: the four Armijo constants $c_1 \sim U(0, 1)$ tried by the line search and by SALSA.

A Experimental Setup Details

Tuned Values Figure 4 shows every hyperparameter value that was tried. The baseline searches the joint space of scheduler type and base LR: each of its 16 trials pairs a scheduler drawn uniformly from $\{\text{cosine}, \text{linear}, \text{inverse-square-root}\}$ with a base LR drawn log-uniformly from $(10^{-7}, 10^{-2})$. The 16 pairs are frozen once and reused unchanged on every task and both model sizes; they are listed, in ascending order of base LR, in the first two columns of Tables 2 and 3. The line search tunes only the Armijo constant, and its four trials are four draws $c_1 \sim U(0, 1)$, again frozen once and reused everywhere:

$$c_1 \in \{0.844,\ 0.758,\ 0.421,\ 0.259\}$$

in draw order. We deliberately did not reject draws in a region already known to be bad: a random search that never spends a trial on a poor value is not a random search, and the smallest draw, $c_1 = 0.259$, is the one that fails on several tasks in Tables 2 and 3. SALSA receives the same four values for its own Armijo constant, so the two line searches are compared on identical draws.

Fixed Settings Table 1 lists the settings that are held fixed within each task. They are shared by all three methods, so that the arms differ only in how the LR is set. The per-dataset constants (epochs, effective batch, sequence length, DPO β, weight decay) are our own choices, fixed before any method was run and not tuned for either arm. The line-search settings of Algorithm 1 are the same on every task: the warmup phase is $T_{\text{wmp}} = \lfloor 0.03\,T \rfloor$ steps, the search budget is $N_{\text{wmp}} = 5$ warmup searches and $N_{\text{dec}} = 20$ decay searches, which fixes the two intervals through Equation 2, the grid ratio is $\rho = 0.5$, and every search moves along the lattice $0.1 \cdot \rho^k$ between the floor 10^{-8} and the ceiling 0.1. The step-0 search backtracks from the ceiling to the largest lattice point that satisfies the Armijo condition; every later warmup search grows the LR upward from the previous anchor, and every decay search shrinks it downward, so no search restarts from a fixed value. The search batch $\hat{V}$ accumulates N_{accum} microbatches, chosen on every task so that $\hat{V}$ holds four effective training batches. The decay-phase searches evaluate on a held-out batch split from the training set, disjoint from the evaluation set. Every run uses AdamW, a single seed (0), fp32 parameters with full-precision matrix multiplies, and $4\times$ A100 40GB GPUs; LoRA uses rank 16 on the query, key, value and output projections. The evaluation loss is computed once per decay-search interval on the evaluation set, the last 512 examples of the training file (1024 for OpenMathInstruct-2), which are never trained on. For SALSA, only c_1 is set by us; every other setting is its authors' default.

B Full Results

Tables 2 and 3 give every run behind Figure 2: the 16 baseline trials, the four line-search trials and the four SALSA trials on each of the eight tasks, at 0.5B and 1.5B. Every cell is the best evaluation

	Alpaca	OM2	UF	Math-Step
objective	SFT	SFT	DPO	DPO
training examples	51,490	77,184	60,623	10,283
evaluation examples	512	1,024	512	512
decay search set (held out from training)	512	1,024	512	512
epochs	6	4	1	8
optimizer steps T	2412	1206	473	642
effective batch	128	256	128	128
max sequence length	512	1024	1024	1024
DPO β	–	–	0.01	0.4
weight decay	0	0.01	0	0
warmup T_{wmp} (steps)	72	36	14	19
warmup search interval $\Delta^{\text{ls}}_{\text{wmp}}$	15	8	3	4
decay search interval $\Delta^{\text{ls}}_{\text{dec}}$	117	58	23	31
evaluation interval (steps)	117	58	23	31

Table 1: Settings held fixed within each task and shared by all methods. Both model sizes use the same values. The two search intervals follow from T_{wmp}, T and the budget $N_{\text{wmp}} = 5$, $N_{\text{dec}} = 20$ through Equation 2; on UF the decay interval is raised from 22 to 23 so that exactly 20 decay anchors fall before T. Every task runs exactly $5 + 20$ searches. T is the fixed step budget of the task; the epoch row is T times the effective batch divided by the number of training examples, rounded.

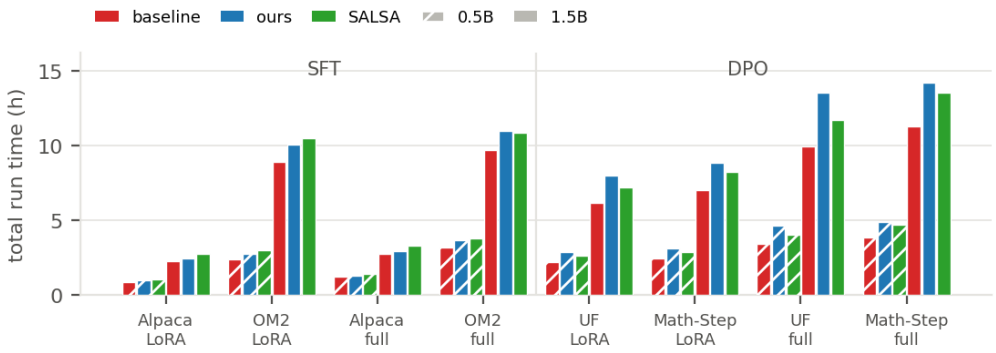


Figure 5: Total wall-clock time of one trial, for each of the eight tasks at both model sizes (hatched: 0.5B; solid: 1.5B). The three methods differ only in how the LR is set.

loss reached by that run, the quantity the β-RMPs are computed from. A cell marked *div.* is a run the divergence rule of Section 4 terminated; in every such run the evaluation loss never improved on its step-0 value; for the baseline these are the large-LR trials at the bottom of the table, and for the line search they are concentrated on the smallest draw, $c_1 = 0.259$. The shading makes the per-task comparison visible at a glance: a red baseline cell is a trial the best line-search draw beats, a blue one a trial it does not. At both sizes the blue cells sit in a narrow band of base LRs, around 5×10^{-4} for the LoRA columns and around 10^{-5} for the full fine-tuning columns, a shift of more than an order of magnitude that no single baseline trial covers; on the two Math-Step tasks no baseline trial reaches the best line-search draw at either size.

C Cost of the Search

Figure 5 shows the total wall-clock time of one trial under each method, for each of the eight tasks at both model sizes. Averaged across the eight tasks at both sizes, the two line searches have nearly identical costs: $1.19\times$ for ours and $1.20\times$ for SALSA. The largest overhead is $1.37\times$ for ours and $1.26\times$ for SALSA. Our highest-overhead cases are all DPO tasks. This reflects their shorter trials

		SFT LoRA		SFT full FT		DPO LoRA		DPO full FT	
sched.	base LR	Alpaca *loss*	OM2 *loss*	Alpaca *loss*	OM2 *loss*	UF *loss*	Math-Step *loss*	UF *loss*	Math-Step *loss*
cos	2.04e-07	2.9537	0.9975	1.6769	0.5736	0.6926	0.6134	0.6929	0.6289
lin	2.65e-07	2.8414	0.9798	1.6173	0.5619	0.6920	0.5253	0.6395	**0.2382**
isqrt	8.48e-07	2.4343	0.9030	1.4410	0.5248	0.6919	0.4597	0.6217	0.4971
cos	1.74e-06	1.9471	0.7007	**1.3269**	0.4515	0.6806	0.3503	0.6045	**0.2438**
cos	2.11e-06	1.9233	0.6790	**1.3220**	0.4458	0.6784	0.3376	0.6023	0.3096
isqrt	8.26e-06	**1.8490**	0.6260	**1.3130**	**0.4286**	0.6666	0.2898	**0.5695**	0.3191
lin	1.14e-05	**1.8232**	0.6001	**1.3098**	**0.4227**	0.6452	0.2618	**0.5569**	0.4503
isqrt	3.42e-05	**1.8099**	0.5871	**1.3149**	**0.4218**	0.6313	0.2559	**0.5674**	0.6166
isqrt	7.81e-05	**1.7948**	**0.5758**	**1.3401**	**0.4245**	0.6105	**0.2528**	0.5869	0.5713
cos	1.81e-04	**1.7746**	**0.5615**	1.4262	0.4585	**0.5775**	**0.2432**	0.6575	*div.*
isqrt	4.07e-04	**1.7730**	**0.5589**	1.4831	0.4625	**0.5680**	**0.2501**	0.6595	*div.*
cos	5.37e-04	**1.7721**	**0.5531**	1.5083	0.4958	**0.5580**	0.2988	*div.*	*div.*
isqrt	6.17e-04	**1.7728**	**0.5559**	1.5331	0.4805	**0.5565**	0.2645	*div.*	*div.*
lin	3.02e-03	**1.8179**	**0.5620**	2.1908	*div.*	*div.*	*div.*	*div.*	*div.*
cos	3.94e-03	**1.8310**	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*
cos	7.45e-03	1.8619	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*
ours c_1=0.844		**1.7989**	**0.5641**	**1.3139**	**0.4388**	**0.5712**	**0.2430**	0.5895	**0.2341**
ours c_1=0.758		**1.7824**	**0.5597**	**1.3141**	**0.4277**	0.5942	**0.2524**	**0.5640**	0.2579
ours c_1=0.421		**1.7739**	**0.5803**	*div.*	**0.4257**	*div.*	*div.*	0.6137	0.3902
ours c_1=0.259		*div.*	*div.*	1.8710	**0.4275**	*div.*	*div.*	*div.*	*div.*
SALSA c=0.844		**1.8378**	0.5944	1.7528	0.5595	**0.5771**	0.2582	**0.5847**	**0.2400**
SALSA c=0.758		**1.8172**	0.5843	1.7033	0.5276	**0.5707**	**0.2489**	**0.5784**	**0.2367**
SALSA c=0.421		**1.7911**	**0.5582**	**1.3250**	**0.4279**	**0.5696**	0.2657	**0.5576**	**0.2348**
SALSA c=0.259		**1.7830**	**0.5660**	**1.3408**	**0.4398**	0.5967	0.2683	0.5867	0.2480

Table 2: Result on Qwen2.5 0.5B base and instruct. Every column reports the best evaluation loss reached by the run (for DPO, the evaluation DPO loss). Bold = within 5% of the column optimum; a shaded baseline cell is at least as good as (blue) or worse than (red) the best of the four line-search draws. The four line-search rows are the four frozen random c_1 draws, in draw order; the SALSA rows are the same four draws with c_1 given to SALSA's own Armijo test. The column optimum is taken over the rows shown. *div.* = the run diverged: its evaluation loss never improved on step 0. Such a cell sets no column optimum, takes no bold and drives no shading.

		SFT LoRA		SFT full FT		DPO LoRA		DPO full FT	
sched.	base LR	Alpaca *loss*	OM2 *loss*	Alpaca *loss*	OM2 *loss*	UF *loss*	Math-Step *loss*	UF *loss*	Math-Step *loss*
cos	2.04e-07	2.8813	0.9176	1.5264	0.5001	0.6927	0.5848	0.6048	0.2155
lin	2.65e-07	2.8021	0.9052	1.4783	0.4908	0.6926	0.5634	0.5941	0.2119
isqrt	8.48e-07	2.4480	0.8615	1.3187	0.4609	0.6920	0.5005	0.5776	0.1983
cos	1.74e-06	1.7461	0.6735	**1.1905**	0.3896	0.6865	0.3546	0.5631	0.2046
cos	2.11e-06	1.7235	0.6366	**1.1854**	0.3848	0.6837	0.3355	0.5383	0.2121
isqrt	8.26e-06	**1.6718**	0.5478	**1.1788**	**0.3679**	0.6641	0.2743	**0.5249**	0.2744
lin	1.14e-05	**1.6510**	0.5246	**1.1783**	**0.3635**	0.6283	0.2368	**0.5076**	0.3643
isqrt	3.42e-05	**1.6385**	0.5136	**1.1819**	**0.3621**	0.6023	0.2265	**0.5126**	0.2281
isqrt	7.81e-05	**1.6258**	**0.5027**	**1.2072**	**0.3671**	0.5754	0.2198	**0.5303**	*div.*
cos	1.81e-04	**1.6107**	**0.4897**	1.2874	0.4034	**0.5268**	**0.2071**	*div.*	*div.*
isqrt	4.07e-04	**1.6113**	**0.4871**	1.3298	0.4075	**0.5186**	0.2298	*div.*	*div.*
cos	5.37e-04	**1.6108**	**0.4823**	1.3519	0.4426	**0.5229**	0.2508	*div.*	*div.*
isqrt	6.17e-04	**1.6113**	**0.4843**	1.3619	0.4139	**0.5200**	0.2554	*div.*	*div.*
lin	3.02e-03	**1.6454**	**0.4876**	1.8157	*div.*	*div.*	*div.*	*div.*	*div.*
cos	3.94e-03	**1.6557**	*div.*	2.1876	*div.*	*div.*	*div.*	*div.*	*div.*
cos	7.45e-03	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*
ours c_1=0.844		**1.6250**	**0.4887**	**1.1818**	**0.3639**	0.5559	**0.2092**	**0.5243**	**0.1907**
ours c_1=0.758		**1.6188**	**0.4873**	**1.1823**	**0.3713**	**0.5221**	**0.2050**	*div.*	**0.1864**
ours c_1=0.421		**1.6250**	**0.4825**	**1.2345**	**0.3683**	*div.*	0.4290	0.6344	0.6544
ours c_1=0.259		**1.6269**	**0.4855**	*div.*	**0.3715**	*div.*	*div.*	0.5922	*div.*
SALSA c=0.844		**1.6666**	0.5199	1.6086	0.4817	**0.5281**	0.2219	0.5661	0.2186
SALSA c=0.758		**1.6381**	**0.5051**	1.5385	0.4504	**0.5233**	0.2179	**0.5254**	0.2145
SALSA c=0.421		**1.6207**	**0.4897**	**1.1915**	**0.3721**	**0.5295**	**0.2127**	**0.5054**	0.2023
SALSA c=0.259		**1.6239**	**0.4871**	**1.2091**	**0.3759**	0.5456	**0.2139**	0.5360	0.2006

Table 3: Result on Qwen2.5 1.5B base and instruct. Every column reports the best evaluation loss reached by the run (for DPO, the evaluation DPO loss). Bold = within 5% of the column optimum; a shaded baseline cell is at least as good as (blue) or worse than (red) the best of the four line-search draws. The four line-search rows are the four frozen random c_1 draws, in draw order; the SALSA rows are the same four draws with c_1 given to SALSA's own Armijo test. The column optimum is taken over the rows shown. *div.* = the run diverged: its evaluation loss never improved on step 0. Such a cell sets no column optimum, takes no bold and drives no shading.

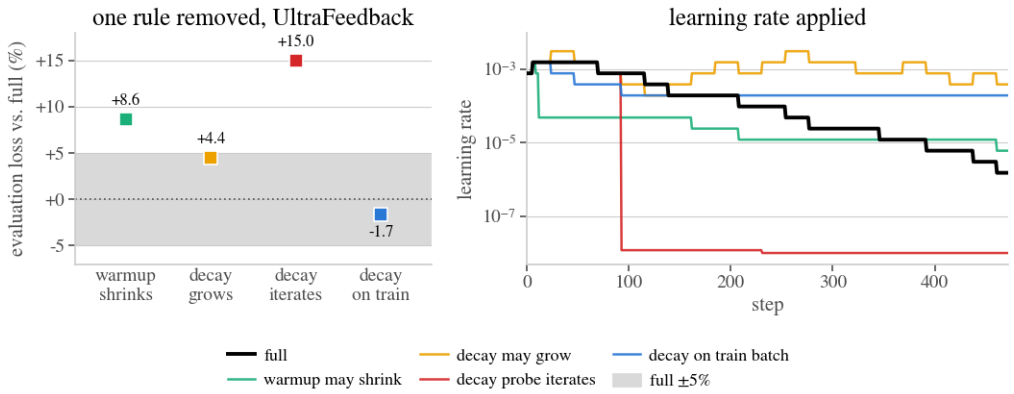


Figure 6: Removing one rule of the scheduler at a time, on UltraFeedback with the 0.5B model and LoRA. Left: evaluation loss of each ablated variant relative to the full method, with the same $\pm 5\%$ band as Figure 3; every run is one seed. Right: the learning rate each variant applied. Letting the decay probe iterate until its center wins drives the learning rate to the floor within a hundred steps and costs 15%; letting the warmup search shrink the rate ends warmup $32\times$ lower and costs 8.6%; letting the decay probe grow the rate holds it above 10^{-3} for the whole run and costs 4%. Replacing the held-out probe by an Armijo test on the training batch lowers the loss by 1.7%, within the band, but the rate then barely decays.

rather than more expensive searches: DPO trials take 473–642 optimizer steps, compared with 1206–2412 for SFT, so the same line-search budget is spread over roughly one-third as many steps.

D Ablation of the Scheduler's Rules

The scheduler of Section 3.2 rests on three rules: the warmup search only raises the LR, the decay probe scores candidates on the held-out search batch, and it takes one such step per anchor. Figure 6 shows four variants on UltraFeedback with the 0.5B model and LoRA, under the protocol of Appendix A: three remove one rule each, and the fourth replaces the held-out probe by an Armijo test on the training batch. A decay probe that iterates to convergence at every anchor halves the rate repeatedly until it reaches the floor, since among noisy held-out estimates some smaller candidate always wins, and the model stops training a hundred steps in. A warmup search allowed to shrink the LR ends warmup at 4.9×10^{-5} instead of 1.6×10^{-3} and never recovers; a decay probe allowed to grow it holds the rate above 10^{-3} for the whole run. Scoring the decay search on the training batch instead of held-out data lowers the loss by 1.7%, within what a single seed resolves, but the rate then changes only four times in the run and never leaves 2×10^{-4}, so the decay the method is designed to produce does not happen.

E Training Curves

Figures 7 and 8 show every run behind Tables 2 and 3 as a curve: one row per task, with the training loss, the evaluation loss and the learning rate against the optimizer step. The 16 baseline trials are drawn in gray, the four line-search trials in blue and the four SALSA trials in green. The training loss is a 20-step running mean; both loss panels are clipped at $1.5\times$ the step-0 evaluation loss, so a run that leaves the frame at the top is one whose loss blew up, and a curve that ends early is a run the divergence rule of Section 4 terminated. The LR panels make the three mechanisms visible. The baseline draws are smooth cosine, linear or inverse-square-root schedulers spread over five orders of magnitude, and on every task only a narrow band of them reaches the floor of the evaluation-loss panel. The line search produces a staircase: a short climb during warmup and then a sequence of halvings, each one a decay anchor at which the loss on the held-out search set at $\rho\alpha$ was lower than at α (Algorithm 2; no Armijo test is involved after warmup), which together carry the rate down one to three orders of magnitude by the end of training. SALSA adjusts the rate at every step, so its curve

is a wandering band that, on the SFT tasks, settles an order of magnitude or more below the best baseline draw; the spikes in its evaluation loss are the steps where its per-step search accepted a rate the model could not tolerate.

F A Narrower Prior on the Learning Rate

The baseline of Section 4 draws its base LR from five decades, $(10^{-7}, 10^{-2})$, because the protocol asks what a method costs on a problem it has not been tuned for. A practitioner who has trained the same kind of model before does not start from five decades; they start from a rate that is known to work for their objective and parameterization and search around it. This appendix repeats the comparison for that practitioner. The baseline grid is restricted to the trials whose base LR falls within one decade either side of a per-case reference, each the default of a published recipe: 7×10^{-6} for SFT with full fine-tuning (Qwen Team, 2024), 10^{-4} for SFT with LoRA (Zheng et al., 2024), 10^{-6} for DPO with full fine-tuning (Rafailov et al., 2023) and 5×10^{-6} for DPO with LoRA (Zheng et al., 2024). The windows leave seven of the 16 frozen trials for SFT with LoRA and six for each of the other three cases. The line-search and SALSA arms are unchanged: they receive no prior. Every other element of the protocol, including the two stopping rules, the β unit of one full baseline trial and the per-task target, is as in Section 4; the target is recomputed over the trials that remain, so it can only move in the baseline's favor.

Figure 9 shows the profiles and Tables 4 and 5 the runs behind them. The prior removes the baseline's failure mode: none of the 50 windowed trials at either size diverges, against 25 and 28 of the 128 trials of the full grid. It does not close the gap at small budgets. At the 5% tolerance and $\beta = 1.5$ the windowed baseline solves 3.5 of the 8 tasks at 0.5B and 3.2 at 1.5B, averaged over the 100 orderings, against 5.8 and 5.9 for the line search; at $\beta = 3.5$ it solves 5.3 and 4.4 against 7.3 and 7.7. Run to the end of its ladder, it reaches 6 of the 8 tasks at 0.5B and 5 at 1.5B in every ordering, while the line search reaches all 8 at both sizes by $\beta = 5.5$. The tables show where the remaining gap comes from. On the four SFT tasks the window contains the best trial of the full grid, and the windowed baseline's best run and the best line-search draw are within 1.2% of each other on every SFT task at both sizes: with a good prior, SFT is a tie. On the four DPO tasks the published rates, all of which were calibrated on models of 7B scale or larger, sit one to two decades below where the best full-grid trials fall on these 0.5B and 1.5B models (Tables 2 and 3). The windows therefore admit no trial within 5% of the target on either DPO LoRA task at either size, nor on Math-Step with full fine-tuning at 1.5B, and the best line-search draw beats the best windowed trial on all eight DPO tasks, by 0.1% on UltraFeedback with full fine-tuning at 1.5B and by 1–15% elsewhere. This is the heterogeneity of Section 4 seen from the practitioner's side: a prior transfers across objectives and parameterizations only as far as it was calibrated, and the line search does not need one. Among non-diverged runs, and against the targets recomputed over the remaining trials, 18 of its 23 trials at 0.5B and 21 of 26 at 1.5B fall within the 5% tolerance, against 23 of 50 and 21 of 50 for the windowed baseline.

G Does a Lower DPO Loss Mean a Better Model?

The Two Quantities For an evaluation preference pair (x, y_w, y_l) the DPO objective assigns the policy an implicit reward margin $m = \beta\big[(\log \pi_\theta(y_w|x) - \log \pi_{\text{ref}}(y_w|x)) - (\log \pi_\theta(y_l|x) - \log \pi_{\text{ref}}(y_l|x))\big]$. The **evaluation DPO loss** we report throughout is the mean of $-\log \sigma(m)$ over the evaluation pairs; it starts at $\log 2 \approx 0.693$ for an untrained policy and keeps decreasing as long as the margins grow. The **evaluation preference accuracy** is the fraction of evaluation pairs with $m > 0$, i.e. the pairs on which the policy prefers the chosen response more strongly than the reference model does. The two are computed from the same forward passes at the same checkpoints. They can disagree in two ways. Accuracy is a thresholded version of the margin, so it saturates once most pairs are ordered correctly while the loss can still fall by widening margins that are already positive. Conversely, the loss can be driven down by pushing the likelihood of the rejected response far below that of the reference on a few pairs, which changes the loss a lot and the accuracy not at all (Pal et al., 2024; Razin et al., 2025). Whether either effect matters in our setting is an empirical question, which we answer on our own runs.

Across All Evaluation Points Figure 10 plots the two quantities against each other at every evaluation point of every DPO run in the main grid: the 16 baseline trials, the four line-search trials

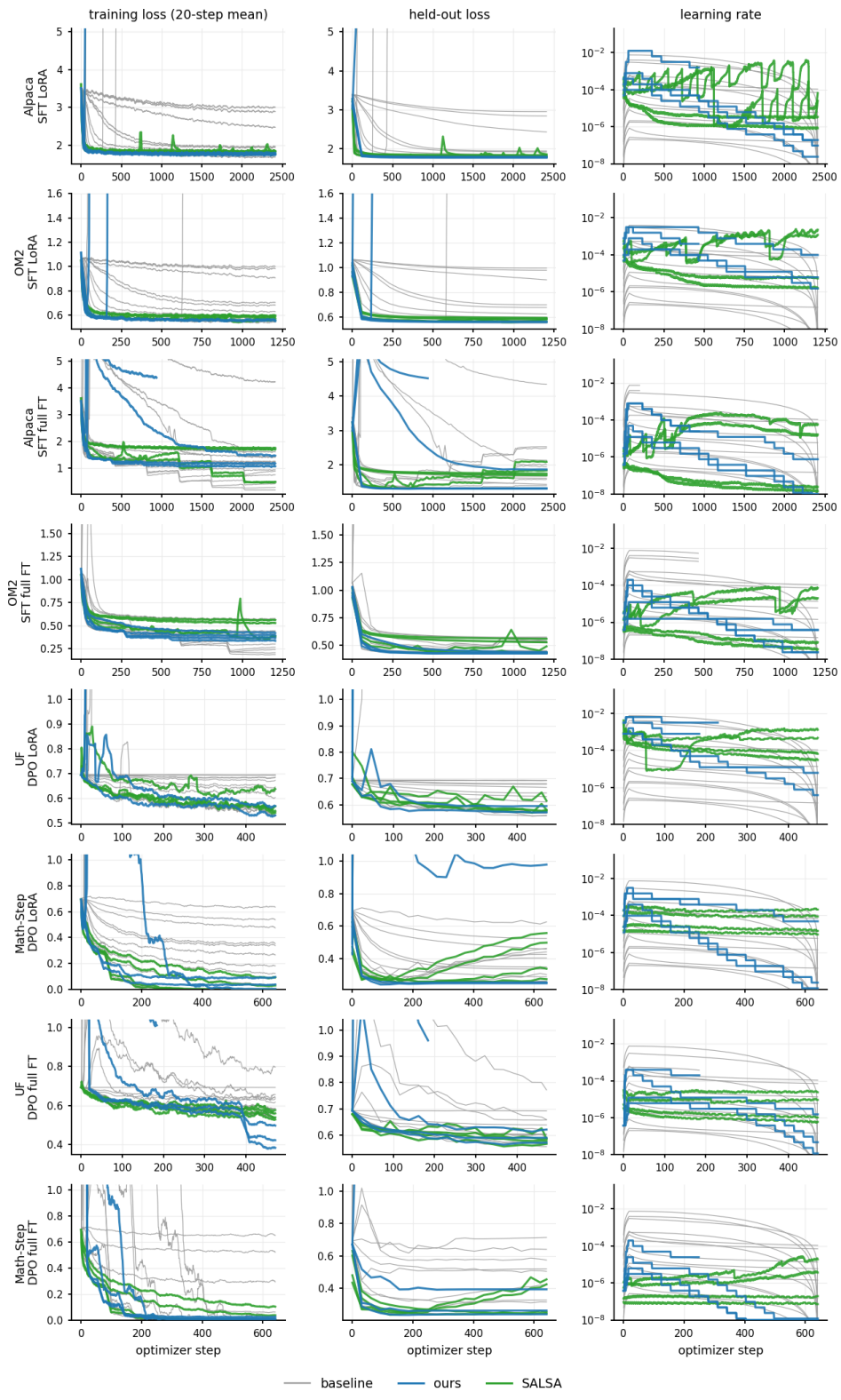


Figure 7: Qwen2.5-0.5B: training loss (20-step running mean), evaluation loss and learning rate of every run on the eight tasks. Gray: the 16 baseline trials; blue: the four line-search trials; green: the four SALSA trials. Loss panels are clipped at $1.5\times$ the step-0 evaluation loss.

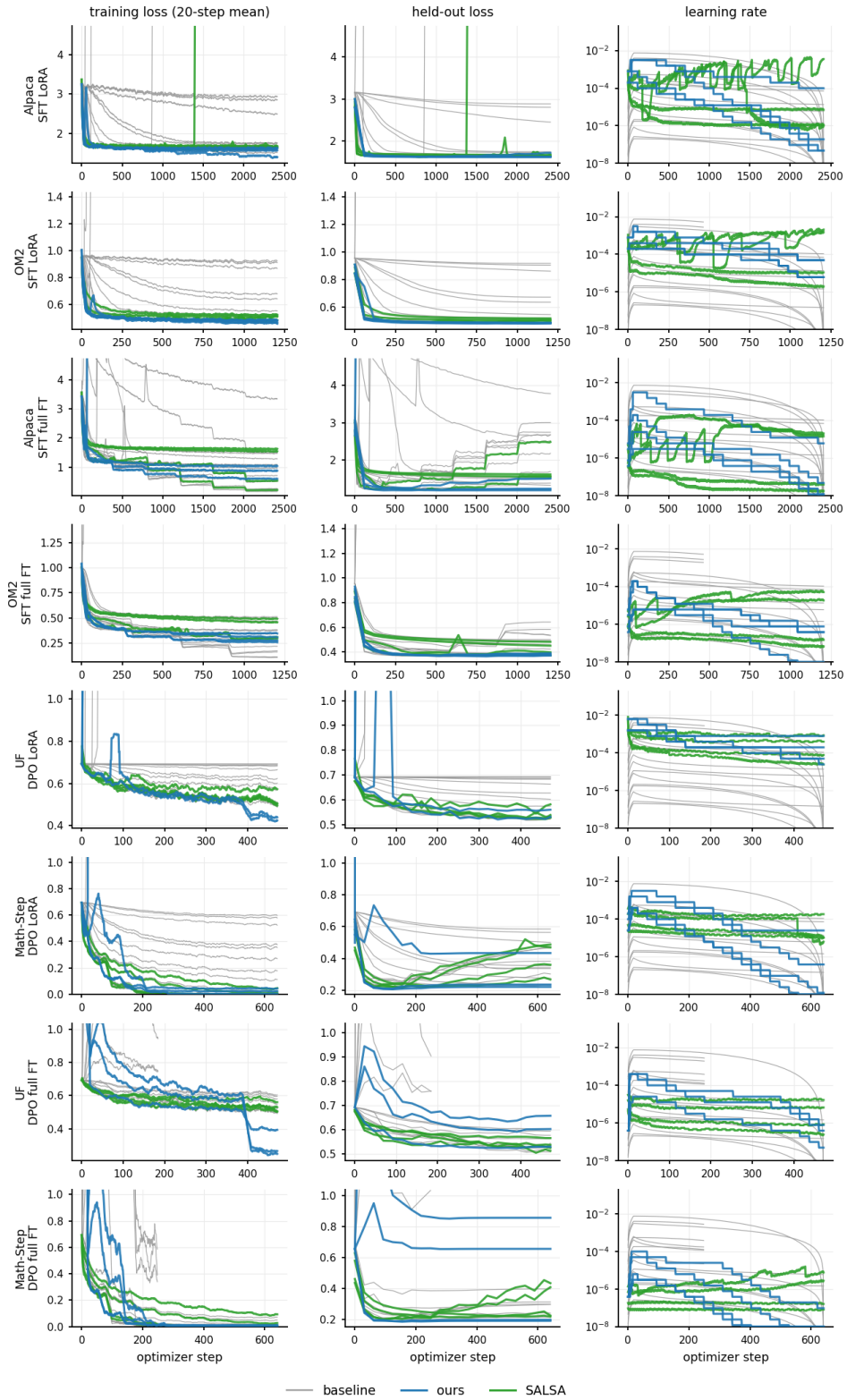


Figure 8: Qwen2.5-1.5B: the same panels as Figure 7.

		SFT LoRA		SFT full FT		DPO LoRA		DPO full FT	
sched.	base LR	Alpaca *loss*	OM2 *loss*	Alpaca *loss*	OM2 *loss*	UF *loss*	Math-Step *loss*	UF *loss*	Math-Step *loss*
cos	2.04e-07	–	–	–	–	–	–	0.6929	0.6289
lin	2.65e-07	–	–	–	–	–	–	0.6395	**0.2382**
isqrt	8.48e-07	–	–	1.4410	0.5248	0.6919	0.4597	0.6217	0.4971
cos	1.74e-06	–	–	**1.3269**	0.4515	0.6806	0.3503	0.6045	**0.2438**
cos	2.11e-06	–	–	**1.3220**	0.4458	0.6784	0.3376	0.6023	0.3096
isqrt	8.26e-06	–	–	**1.3130**	**0.4286**	0.6666	0.2898	**0.5695**	0.3191
lin	1.14e-05	**1.8232**	0.6001	**1.3098**	**0.4227**	0.6452	0.2618	–	–
isqrt	3.42e-05	**1.8099**	0.5871	**1.3149**	**0.4218**	0.6313	0.2559	–	–
isqrt	7.81e-05	**1.7948**	**0.5758**	–	–	–	–	–	–
cos	1.81e-04	**1.7746**	**0.5615**	–	–	–	–	–	–
isqrt	4.07e-04	**1.7730**	**0.5589**	–	–	–	–	–	–
cos	5.37e-04	**1.7721**	**0.5531**	–	–	–	–	–	–
isqrt	6.17e-04	**1.7728**	**0.5559**	–	–	–	–	–	–
ours $c_1{=}0.844$		**1.7989**	**0.5641**	**1.3139**	**0.4388**	**0.5712**	**0.2430**	0.5895	**0.2341**
ours $c_1{=}0.758$		**1.7824**	**0.5597**	**1.3141**	**0.4277**	**0.5942**	**0.2524**	**0.5640**	0.2579
ours $c_1{=}0.421$		**1.7739**	**0.5803**	*div.*	**0.4257**	*div.*	*div.*	0.6137	0.3902
ours $c_1{=}0.259$		*div.*	*div.*	1.8710	**0.4275**	*div.*	*div.*	*div.*	*div.*
SALSA $c{=}0.844$		**1.8378**	0.5944	1.7528	0.5595	**0.5771**	0.2582	**0.5847**	**0.2400**
SALSA $c{=}0.758$		**1.8172**	0.5843	1.7033	0.5276	**0.5707**	**0.2489**	**0.5784**	**0.2367**
SALSA $c{=}0.421$		**1.7911**	**0.5582**	**1.3250**	**0.4279**	**0.5696**	0.2657	**0.5576**	**0.2348**
SALSA $c{=}0.259$		**1.7830**	**0.5660**	**1.3408**	**0.4398**	**0.5967**	0.2683	0.5867	0.2480

Table 4: Result on Qwen2.5 0.5B base and instruct, baseline restricted to one decade either side of a per-case reference LR (SFT LoRA 10^{-4}, SFT full FT 7×10^{-6}, DPO LoRA 5×10^{-6}, DPO full FT 10^{-6}); – = a baseline trial outside the column's window, not counted. Every column reports the best evaluation loss reached by the run (for DPO, the evaluation DPO loss). Bold = within 5% of the column optimum; a shaded baseline cell is at least as good as (blue) or worse than (red) the best of the four line-search draws. The four line-search rows are the four frozen random c_1 draws, in draw order; the SALSA rows are the same four draws with c_1 given to SALSA's own Armijo test. The column optimum is taken over the rows shown. *div.* = the run diverged: its evaluation loss never improved on step 0. Such a cell sets no column optimum, takes no bold and drives no shading.

		SFT LoRA		SFT full FT		DPO LoRA		DPO full FT	
sched.	base LR	Alpaca *loss*	OM2 *loss*	Alpaca *loss*	OM2 *loss*	UF *loss*	Math-Step *loss*	UF *loss*	Math-Step *loss*
cos	2.04e-07	–	–	–	–	–	–	0.6048	0.2155
lin	2.65e-07	–	–	–	–	–	–	0.5941	0.2119
isqrt	8.48e-07	–	–	1.3187	0.4609	0.6920	0.5005	0.5776	0.1983
cos	1.74e-06	–	–	**1.1905**	0.3896	0.6865	0.3546	0.5631	0.2046
cos	2.11e-06	–	–	**1.1854**	0.3848	0.6837	0.3355	0.5383	0.2121
isqrt	8.26e-06	–	–	**1.1788**	**0.3679**	0.6641	0.2743	**0.5249**	0.2744
lin	1.14e-05	**1.6510**	0.5246	**1.1783**	**0.3635**	0.6283	0.2368	–	–
isqrt	3.42e-05	**1.6385**	0.5136	**1.1819**	**0.3621**	0.6023	0.2265	–	–
isqrt	7.81e-05	**1.6258**	**0.5027**	–	–	–	–	–	–
cos	1.81e-04	**1.6107**	**0.4897**	–	–	–	–	–	–
isqrt	4.07e-04	**1.6113**	**0.4871**	–	–	–	–	–	–
cos	5.37e-04	**1.6108**	**0.4823**	–	–	–	–	–	–
isqrt	6.17e-04	**1.6113**	**0.4843**	–	–	–	–	–	–
ours c_1=0.844		**1.6250**	**0.4887**	**1.1818**	**0.3639**	0.5559	**0.2092**	**0.5243**	**0.1907**
ours c_1=0.758		**1.6188**	**0.4873**	**1.1823**	**0.3713**	**0.5221**	**0.2050**	*div.*	**0.1864**
ours c_1=0.421		**1.6250**	**0.4825**	**1.2345**	**0.3683**	*div.*	0.4290	0.6344	0.6544
ours c_1=0.259		**1.6269**	**0.4855**	*div.*	**0.3715**	*div.*	*div.*	0.5922	*div.*
SALSA c=0.844		**1.6666**	0.5199	1.6086	0.4817	**0.5281**	0.2219	0.5661	0.2186
SALSA c=0.758		**1.6381**	**0.5051**	1.5385	0.4504	**0.5233**	0.2179	**0.5254**	0.2145
SALSA c=0.421		**1.6207**	**0.4897**	**1.1915**	**0.3721**	**0.5295**	**0.2127**	**0.5054**	0.2023
SALSA c=0.259		**1.6239**	**0.4871**	**1.2091**	**0.3759**	**0.5456**	**0.2139**	0.5360	0.2006

Table 5: Result on Qwen2.5 1.5B base and instruct, baseline restricted to one decade either side of a per-case reference LR (SFT LoRA 10^{-4}, SFT full FT 7×10^{-6}, DPO LoRA 5×10^{-6}, DPO full FT 10^{-6}); – = a baseline trial outside the column's window, not counted. Every column reports the best evaluation loss reached by the run (for DPO, the evaluation DPO loss). Bold = within 5% of the column optimum; a shaded baseline cell is at least as good as (blue) or worse than (red) the best of the four line-search draws. The four line-search rows are the four frozen random c_1 draws, in draw order; the SALSA rows are the same four draws with c_1 given to SALSA's own Armijo test. The column optimum is taken over the rows shown. *div.* = the run diverged: its evaluation loss never improved on step 0. Such a cell sets no column optimum, takes no bold and drives no shading.

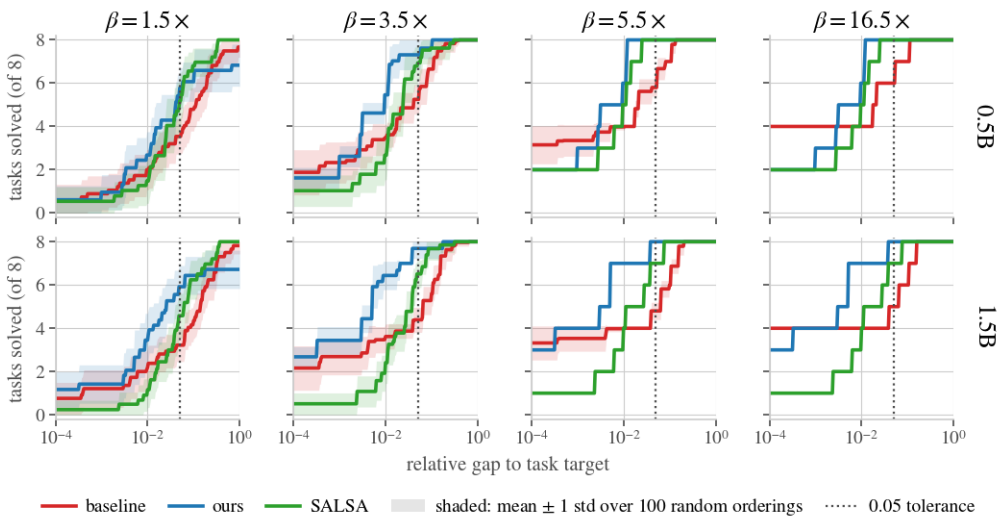


Figure 9: β-relative minimization profiles with the baseline restricted to one decade either side of a per-case reference LR. Same tasks, budgets, tolerance mark, 100 random orderings, shading and vertical axis (tasks solved, out of 8) as Figure 2; the line-search and SALSA curves are those of Figure 2, since those arms receive no prior. The windowed baseline holds six or seven trials per task, so its ladder is exhausted before $\beta = 16.5$ and its last panel is the end of its ladder.

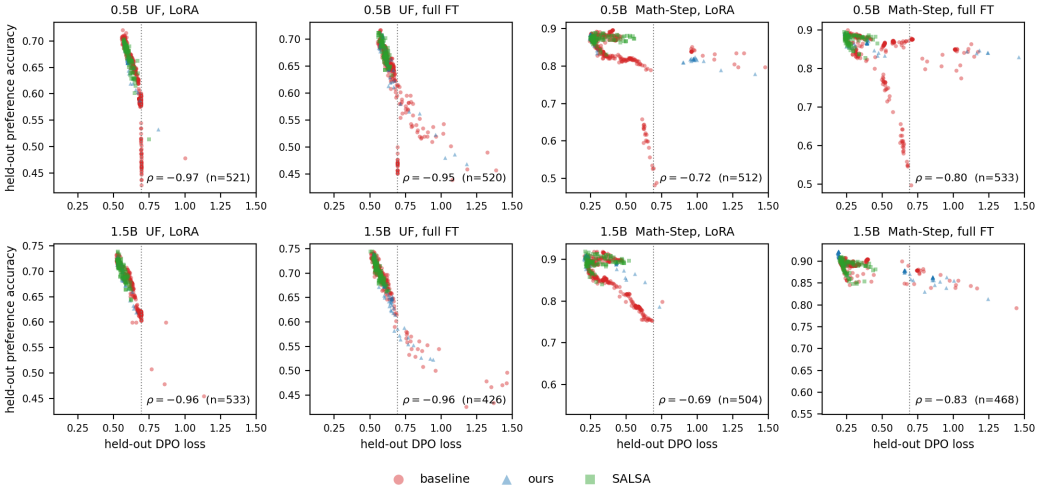


Figure 10: Evaluation DPO loss against evaluation preference accuracy at every evaluation point of every run of the three methods on the four DPO tasks (columns) at 0.5B (top) and 1.5B (bottom). The dotted line is $\log 2$, the loss of the untrained policy. ρ is the Spearman correlation over the points shown.

and the four SALSA trials, on both DPO datasets, both parameterizations and both model sizes, about 500 points per panel. The step-0 evaluation is excluded. Table 6 gives the corresponding rank correlations. On UltraFeedback the relation is close to monotone, with Spearman ρ between -0.95 and -0.97 over all points and between -0.92 and -0.96 when the runs whose loss never improved on initialization are removed. On Math-Step the correlation is weaker, ρ between -0.69 and -0.83, for the first of the two reasons above: accuracy saturates near 0.9 once the loss is below about 0.4, so the lower-left part of each Math-Step panel is a horizontal band along which the loss keeps ordering runs that accuracy no longer separates. We do not see the second effect: no run in Figure 10 reaches a low loss at a low accuracy.

			Spearman ρ		acc. given up by selecting on loss	
size	task	eval points	all runs	converged	mean	max
0.5B	UF, LoRA	521	-0.97	-0.96	0.007	0.018
0.5B	UF, full FT	520	-0.95	-0.92	0.011	0.033
0.5B	Math-Step, LoRA	512	-0.72	-0.54	0.007	0.021
0.5B	Math-Step, full FT	533	-0.80	-0.60	0.010	0.037
1.5B	UF, LoRA	533	-0.96	-0.95	0.006	0.023
1.5B	UF, full FT	426	-0.96	-0.93	0.007	0.021
1.5B	Math-Step, LoRA	504	-0.69	-0.54	0.005	0.035
1.5B	Math-Step, full FT	468	-0.83	-0.64	0.003	0.010

Table 6: Agreement between evaluation DPO loss and evaluation preference accuracy on the runs of Figure 10. *Converged* restricts to runs whose evaluation loss improved on its initialization. The last two columns are, per run, the highest accuracy reached at any evaluation minus the accuracy at the evaluation with the lowest loss, averaged and maximized over the converged runs of the task.

Selecting Checkpoints on Loss Every number in the paper is taken at the evaluation with the lowest evaluation loss. The last two columns of Table 6 measure what this costs in accuracy: within a run, the checkpoint with the lowest loss is on average within 0.003–0.011 of the run's most accurate checkpoint, and never more than 0.037 below it. On 512 evaluation pairs the binomial standard error of an accuracy near 0.7 is about 0.02 and near 0.9 about 0.013, so the checkpoint that loss selects is, within noise, the most accurate one.

Ordering of the Methods Tables 7 and 8 repeat the DPO columns of Tables 2 and 3 with preference accuracy in place of loss, each cell being the accuracy of the same run at the same lowest-loss checkpoint. Taking each method's best trial, the method with the lowest loss is also the method with the highest accuracy on five of the eight (size, task) combinations. On the other three the two metrics pick different methods, but the accuracies of the methods they pick differ by 0.008, 0.002 and 0.018, all within one standard error of the accuracy estimate. Over the range our runs reach, then, a lower evaluation DPO loss does correspond to a more accurate model, the ordering of the three methods is the same under either quantity up to noise, and the caveats of Pal et al. (2024) and Razin et al. (2025) do not change the conclusions drawn from the loss. We do not repeat this check for the SFT tasks: there the evaluation loss is the likelihood of the target behavior on unseen data, which is the standard metric for that objective, and token-level accuracy is a thresholded version of the same quantity rather than independent evidence.

H A Tuning-Free Optimizer as a Reference: Prodigy

The line search removes the sweep over the base LR by testing candidate steps against the objective. Tuning-free optimizers remove the same sweep by estimating the step from the trajectory instead (Section 2.3). To place our results against that family we run Prodigy (Mishchenko & Defazio, 2024) on the tasks of the main table, one run per task at seed 0 with no tuning, since a sweep would defeat the purpose of the method. Every setting a task fixes (the number of steps T, the batch shape, the sequence length, the DPO β, the evaluation set, the search batch and the divergence criteria) is the one the other arms use (Table 1); the only changes are the optimizer and the rule that sets the step size. Following the authors' recommendation for deep learning, Prodigy runs with its LR multiplier at 1, the same warmup (3% of T) and cosine annealing as the baseline's cosine draws, its default $d_0 = 10^{-6}$ and the warmup safeguard on, so the shape of the LR trajectory is the baseline's and its scale is Prodigy's estimate d. Each run is scored like the others, by the minimum evaluation loss over its evaluation points.

Figure 11 and Table 9 give the result, and Figure 12 the learning rate each method actually applied. On supervised fine-tuning the two approaches are equivalent: over the five SFT tasks Prodigy lands within 0.2% of the best-of-sixteen oracle on four and 1.4% above it on the fifth, and it is below the line search's median c_1 draw on four of the five. The step it settles on is the tuned one: its d sits within a factor of two of the best baseline draw's peak learning rate on every SFT task, a few 10^{-4} for LoRA and 2×10^{-5} for full fine-tuning. Preference optimization separates the two, and the

		SFT LoRA		SFT full FT		DPO LoRA		DPO full FT	
sched.	base LR	Alpaca *loss*	OM2 *loss*	Alpaca *loss*	OM2 *loss*	UF *acc*	Math-Step *acc*	UF *acc*	Math-Step *acc*
cos	2.04e-07	2.9537	0.9975	1.6769	0.5736	0.516	0.658	0.467	0.627
lin	2.65e-07	2.8414	0.9798	1.6173	0.5619	0.582	0.820	0.646	**0.881**
isqrt	8.48e-07	2.4343	0.9030	1.4410	0.5248	0.545	0.824	0.654	0.771
cos	1.74e-06	1.9471	0.7007	**1.3269**	0.4515	0.584	0.830	**0.684**	**0.881**
cos	2.11e-06	1.9233	0.6790	**1.3220**	0.4458	0.592	0.838	0.676	0.844
isqrt	8.26e-06	**1.8490**	0.6260	**1.3130**	**0.4286**	0.629	**0.852**	**0.713**	**0.879**
lin	1.14e-05	**1.8232**	0.6001	**1.3098**	**0.4227**	0.645	**0.869**	**0.695**	0.830
isqrt	3.42e-05	**1.8099**	0.5871	**1.3149**	**0.4218**	0.650	**0.871**	**0.695**	**0.869**
isqrt	7.81e-05	**1.7948**	**0.5758**	**1.3401**	**0.4245**	0.660	**0.869**	0.666	**0.867**
cos	1.81e-04	**1.7746**	**0.5615**	1.4262	0.4585	**0.705**	**0.867**	0.648	*div.*
isqrt	4.07e-04	**1.7730**	**0.5589**	1.4831	0.4625	**0.693**	**0.885**	0.627	*div.*
cos	5.37e-04	**1.7721**	**0.5531**	1.5083	0.4958	**0.707**	**0.883**	*div.*	*div.*
isqrt	6.17e-04	**1.7728**	**0.5559**	1.5331	0.4805	**0.703**	**0.871**	*div.*	*div.*
lin	3.02e-03	**1.8179**	**0.5620**	2.1908	*div.*	*div.*	*div.*	*div.*	*div.*
cos	3.94e-03	**1.8310**	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*
cos	7.45e-03	1.8619	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*
ours c_1=0.844		**1.7989**	**0.5641**	**1.3139**	**0.4388**	**0.697**	**0.875**	**0.705**	**0.893**
ours c_1=0.758		**1.7824**	**0.5597**	**1.3141**	**0.4277**	0.664	**0.877**	**0.684**	**0.891**
ours c_1=0.421		**1.7739**	**0.5803**	*div.*	**0.4257**	*div.*	*div.*	0.656	**0.867**
ours c_1=0.259		*div.*	*div.*	1.8710	**0.4275**	*div.*	*div.*	*div.*	*div.*
SALSA c=0.844		**1.8378**	0.5944	1.7528	0.5595	**0.695**	**0.877**	**0.691**	**0.881**
SALSA c=0.758		**1.8172**	0.5843	1.7033	0.5276	**0.699**	**0.875**	**0.686**	**0.881**
SALSA c=0.421		**1.7911**	**0.5582**	**1.3250**	**0.4279**	**0.701**	**0.881**	**0.703**	**0.895**
SALSA c=0.259		**1.7830**	**0.5660**	**1.3408**	**0.4398**	**0.686**	**0.887**	**0.688**	**0.885**

Table 7: Result on Qwen2.5 0.5B base and instruct. SFT reports the best evaluation loss, DPO the evaluation preference accuracy at the best-loss step. Bold = within 5% of the column optimum; a shaded baseline cell is at least as good as (blue) or worse than (red) the best of the four line-search draws. The four line-search rows are the four frozen random c_1 draws, in draw order; the SALSA rows are the same four draws with c_1 given to SALSA's own Armijo test. The column optimum is taken over the rows shown. *div.* = the run diverged: its evaluation loss never improved on step 0. Such a cell sets no column optimum, takes no bold and drives no shading.

		SFT LoRA		SFT full FT		DPO LoRA		DPO full FT	
sched.	base LR	Alpaca *loss*	OM2 *loss*	Alpaca *loss*	OM2 *loss*	UF *acc*	Math-Step *acc*	UF *acc*	Math-Step *acc*
cos	2.04e-07	2.8813	0.9176	1.5264	0.5001	0.617	0.787	0.686	**0.895**
lin	2.65e-07	2.8021	0.9052	1.4783	0.4908	0.615	0.789	0.689	**0.896**
isqrt	8.48e-07	2.4480	0.8615	1.3187	0.4609	0.621	0.816	0.699	**0.908**
cos	1.74e-06	1.7461	0.6735	**1.1905**	0.3896	0.619	0.854	**0.711**	**0.896**
cos	2.11e-06	1.7235	0.6366	**1.1854**	0.3848	0.617	0.850	0.705	**0.900**
isqrt	8.26e-06	**1.6718**	0.5478	**1.1788**	**0.3679**	0.625	0.859	**0.725**	**0.896**
lin	1.14e-05	**1.6510**	0.5246	**1.1783**	**0.3635**	0.676	**0.875**	**0.738**	**0.896**
isqrt	3.42e-05	**1.6385**	0.5136	**1.1819**	**0.3621**	0.688	**0.895**	**0.730**	**0.900**
isqrt	7.81e-05	**1.6258**	**0.5027**	**1.2072**	**0.3671**	0.695	**0.900**	**0.729**	*div.*
cos	1.81e-04	**1.6107**	**0.4897**	1.2874	0.4034	**0.715**	**0.908**	*div.*	*div.*
isqrt	4.07e-04	**1.6113**	**0.4871**	1.3298	0.4075	**0.730**	**0.902**	*div.*	*div.*
cos	5.37e-04	**1.6108**	**0.4823**	1.3519	0.4426	**0.721**	**0.908**	*div.*	*div.*
isqrt	6.17e-04	**1.6113**	**0.4843**	1.3619	0.4139	**0.721**	**0.873**	*div.*	*div.*
lin	3.02e-03	**1.6454**	**0.4876**	1.8157	*div.*	*div.*	*div.*	*div.*	*div.*
cos	3.94e-03	**1.6557**	*div.*	2.1876	*div.*	*div.*	*div.*	*div.*	*div.*
cos	7.45e-03	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*	*div.*
ours c_1=0.844		**1.6250**	**0.4887**	**1.1818**	**0.3639**	0.689	**0.904**	**0.732**	**0.912**
ours c_1=0.758		**1.6188**	**0.4873**	**1.1823**	**0.3713**	**0.729**	**0.900**	*div.*	**0.920**
ours c_1=0.421		**1.6250**	**0.4825**	**1.2345**	**0.3683**	*div.*	**0.889**	0.643	**0.875**
ours c_1=0.259		**1.6269**	**0.4855**	*div.*	**0.3715**	*div.*	*div.*	0.678	*div.*
SALSA c=0.844		**1.6666**	0.5199	1.6086	0.4817	**0.721**	**0.885**	0.701	**0.893**
SALSA c=0.758		**1.6381**	**0.5051**	1.5385	0.4504	**0.723**	**0.902**	**0.723**	**0.898**
SALSA c=0.421		**1.6207**	**0.4897**	**1.1915**	**0.3721**	**0.738**	**0.918**	**0.744**	**0.908**
SALSA c=0.259		**1.6239**	**0.4871**	**1.2091**	**0.3759**	**0.715**	**0.904**	0.701	**0.902**

Table 8: Result on Qwen2.5 1.5B base and instruct. SFT reports the best evaluation loss, DPO the evaluation preference accuracy at the best-loss step. Bold = within 5% of the column optimum; a shaded baseline cell is at least as good as (blue) or worse than (red) the best of the four line-search draws. The four line-search rows are the four frozen random c_1 draws, in draw order; the SALSA rows are the same four draws with c_1 given to SALSA's own Armijo test. The column optimum is taken over the rows shown. *div.* = the run diverged: its evaluation loss never improved on step 0. Such a cell sets no column optimum, takes no bold and drives no shading.

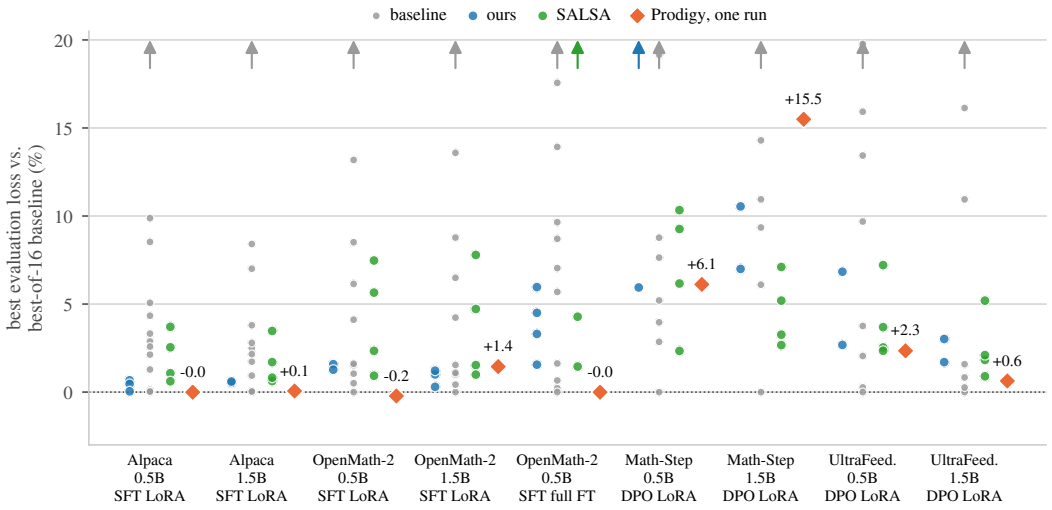


Figure 11: Prodigy against the three arms of the main table. Every point is one run's best evaluation loss relative to the best of that task's sixteen random (LR, scheduler) baseline draws, so 0 is as good as an oracle over sixteen tuned runs and a strip's spread is what each method gives without tuning. Gray: the sixteen baseline draws; blue: the four c_1 draws of our method; green: SALSA on the same four draws; orange diamond: the single Prodigy run, with its value printed. Arrows mark runs above 20%. Only tasks whose Prodigy run has the main table's T are shown. Values in Table 9.

task	size, method	Prodigy	ours median	ours best	baseline best	baseline median	Δ vs. best
Alpaca	0.5B, SFT LoRA	1.7720	1.7824	1.7739	1.7721	1.8271	+0.0%
Alpaca	1.5B, SFT LoRA	1.6117	1.6250	1.6188	1.6107	1.6510	+0.1%
OpenMath-2	0.5B, SFT LoRA	0.5518	0.5641	0.5597	0.5531	0.5936	-0.2%
OpenMath-2	1.5B, SFT LoRA	0.4893	0.4864	0.4825	0.4823	0.5191	+1.4%
OpenMath-2	0.5B, SFT full FT	0.4217	0.4276	0.4257	0.4218	0.4585	+0.0%
Math-Step	0.5B, DPO LoRA	0.2581	0.2477	0.2430	0.2432	0.2898	+6.1%
Math-Step	1.5B, DPO LoRA	0.2392	0.2092	0.2050	0.2071	0.2554	+15.5%
UltraFeedback	0.5B, DPO LoRA	0.5696	0.5827	0.5712	0.5565	0.6452	+2.3%
UltraFeedback	1.5B, DPO LoRA	0.5219	0.5390	0.5221	0.5186	0.6283	+0.6%

Table 9: Prodigy against the three arms of the main table on every task where its run has the main table's T. Entries are the best evaluation loss of a run (the minimum over its evaluation points; lower is better). *Prodigy* is one run at seed 0 with no tuning; *ours* pools the four c_1 draws of the line search and *baseline* the sixteen random (LR, scheduler) draws, so *baseline best* is what an oracle over sixteen tuned runs obtains. The last column is Prodigy relative to that oracle.

separation is not a matter of scale. On UltraFeedback, Prodigy is within 2.3% of the oracle and beats the median line-search draw on both sizes. On Math-Step it finishes 6% (0.5B) and 15% (1.5B) above the oracle and above the line search's two best draws at both sizes, yet Figure 12 shows its learning rate is not out of range: d is equal to the best draw's peak at 0.5B and half of it at 1.5B, and the two runs trace the same cosine curve. Both reach their evaluation-loss minimum at the same step, about 150 of 642, and both then overfit, the evaluation loss rising by 50–80% by the end of training; the scoring rule takes the minimum, so the rise costs neither run, and the gap between them is the depth of that minimum, 0.239 against 0.207 at 1.5B. The line search at $c_1 = 0.9$, the run whose learning rate Figure 12 shows, reaches 0.222 on the same task by a different route: it drops the learning rate by two orders of magnitude within the first 200 steps, reaches its minimum later, at step 250, and stays there, its final loss within 0.001 of its best. On the task where the evaluation-loss minimum is sharp and early, the search that shrinks the step when the objective stops improving finds a lower minimum than any fixed cosine scheduler except the tuned oracle's. In sum, Prodigy is a strong reference on SFT, where it obtains a tuned run's loss with no tuning at all and without the search's overhead, and the line search's margin over it lies on the objective where the shape, not the scale, of the LR trajectory decides the result.

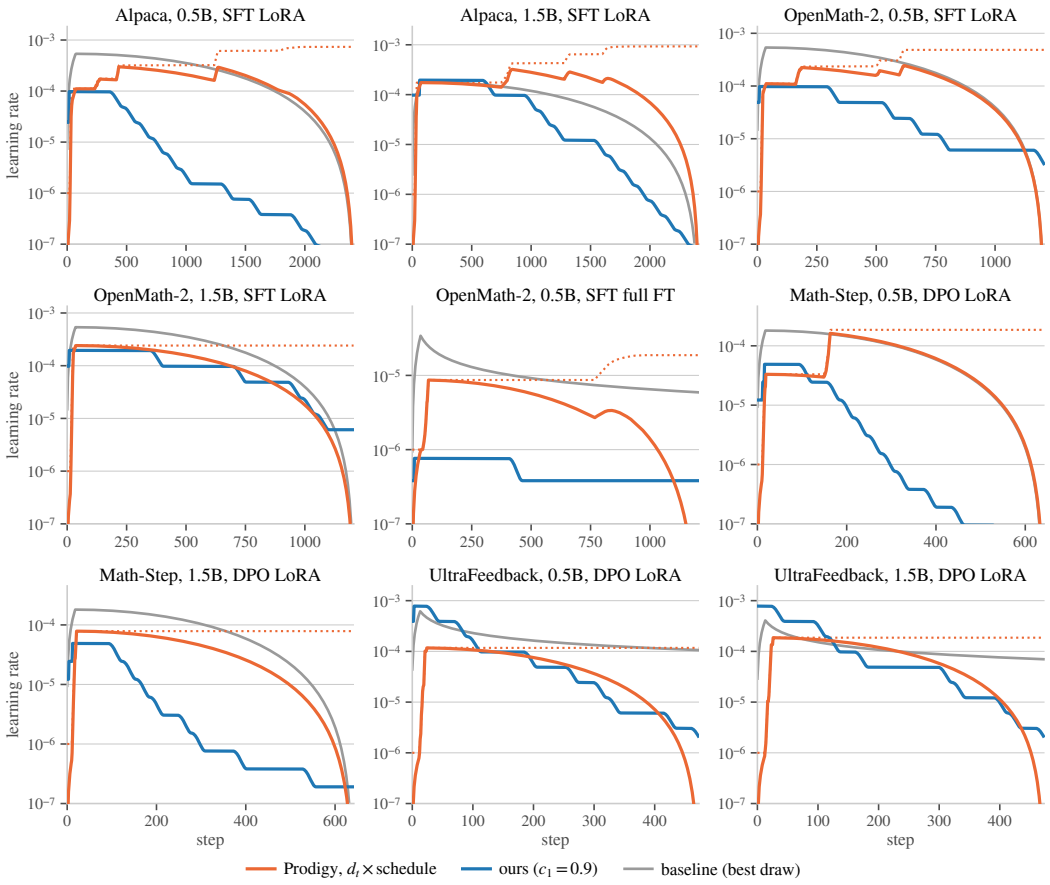


Figure 12: The learning rate applied at each step, for the tasks of Figure 11. Orange: Prodigy, d_t times the warmup-cosine scheduler (solid) and the estimate d_t itself (dotted), which grows in a few jumps and never decreases. Blue: our method at $c_1 = 0.9$. Gray: the best of the task's sixteen random (LR, scheduler) baseline draws, the oracle the strips of Figure 11 are relative to. Log axis floored at 10^{-7}; the cosine tails continue below it.