DR. POST-TRAINING: A DATA REGULARIZATION PERSPECTIVE ON LLM POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

LLM post-training faces a critical challenge: effectively leveraging scarce, high-fidelity target data alongside abundant but imperfectly aligned general training data, a problem that existing methods frame as data selection. We introduce **Dr. Post-Training** (**D**ata-**R**egularized Post-Training), a framework that instead treats general training data as a data-induced regularizer that stabilizes learning from the scarce target objective. Specifically, our framework proposes that at each training step, construct a feasible set of model update directions that depends on the general training data alone, and project the model update direction specified by the scarce target data onto that feasible set. Standard training and existing data selection methods arise as special cases with different choices of the data-induced regularizer, and these methods correspond to different points on a bias–variance spectrum with different regularization strength. The framework thus provides a principled lens for understanding these tradeoffs and guiding the design of new methods. Building on this view, we propose a family of methods that expose the regularization strength as an explicit design choice, and a theory that ties this choice to how well the general data covers the target signals and how much target noise it lets through. For practical LLM-scale use, we introduce careful system optimizations that realize these methods at a per-step cost on par with standard training. Experiments across SFT, RLHF, and RLVR follow the predicted tradeoff: the best regularization strength tracks the mismatch between the general and target data, with our methods gaining most where the target is diluted in the general data and standard training remaining competitive where the two are aligned.

1 INTRODUCTION

Post-training adapts pretrained large language models (LLMs) to desired downstream behaviors, such as instruction following and domain-specific reasoning, through supervised fine-tuning (SFT) (Mishra et al., 2022; Muennighoff et al., 2025) or reinforcement learning (RL) variants such as reinforcement learning from human feedback (RLHF) (Christiano et al., 2017; Ouyang et al., 2022) and reinforcement learning with verifiable rewards (RLVR) (Guo et al., 2025). In many practical settings, however, the data that most directly reflects the downstream objective is scarce: high-quality demonstrations (Ouyang et al., 2022), domain-specific examples (Ling et al., 2025), preference labels (Casper et al., 2023), or task-specific validation data (Perez et al., 2021) are expensive to obtain, whereas general training data, such as broad instruction-tuning mixtures (Wang et al., 2023b; Lambert et al., 2025), are abundant and cheap but only partially aligned with the target objective (Gururangan et al., 2020; Xia et al., 2024a; He et al., 2024). This creates a natural tradeoff: training on the scarce target data is faithful to the downstream objective but statistically noisy, while training on the abundant general data is more stable but may be biased by the misalignment with the target.

This tradeoff has led to a growing literature on data-centric methods for post-training (see Appendix A for related works), including heuristic filtering (Bai et al., 2022; Ethayarajh et al., 2022), data mixture optimization (Iyer et al., 2022; Wang et al., 2023b), offline data selection (He et al., 2024; Xia et al., 2024a), and online data selection (Wang et al., 2024a; Hu et al., 2025b). Despite their algorithmic differences, most existing approaches prioritize "high-utility" samples from the general training data, framing the problem as *data selection* (Albalak et al., 2024). This framing, however, obscures a more fundamental question: not which general data to select, but how the general data should influence the model's updates so as to improve a target objective defined by much scarcer data.

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.

We introduce **Dr.** (**D**ata-**R**egularized) Post-Training, a framework that formalizes this idea. At each training step, the target signal specifies the ideal update direction, while the general training data constrain the update by projecting it onto a feasible set that depends on the training data alone. General training data thus act as a data-induced *regularizer*: they do not define the target objective, but restrict how the model can move toward it, stabilizing learning from the scarce target signals.

Standard training and existing data selection methods arise as special cases of this framework with different choices of the data-induced regularizer, and they can be viewed as different points on a bias–variance spectrum: a feasible set incurs bias insofar as it fails to cover the target signals and variance insofar as it exposes the update to target noise, and enlarging the set, i.e., weakening the regularization, trades the former for the latter. We show that these two quantities govern both the error of a single update and whether repeated updates converge on the target objective.

Building on this framework, we propose *Group-Wise Subset Update*, a family of methods that tune the regularization strength, offering more flexible bias–variance tradeoffs across downstream scenarios. Realizing them at LLM scale is nontrivial: they depend on intermediate quantities that standard training discards, and a naive implementation incurs substantial memory or compute overhead. We develop system-level optimizations that make them practical in modern post-training pipelines.

Empirically, we validate Dr. Post-Training across SFT, RLHF, and RLVR, and the results are consistent with the bias–variance tradeoff: the preferred regularization strength depends on the mismatch between the general and target data, our methods gain most where the target is diluted in the general data, and standard training remains competitive when the two are aligned, while system benchmarks show a per-step cost on par with standard training at LLM scale. In summary, our contributions are:

- **Framework.** Dr. Post-Training, which treats general training data as a data regularizer, subsumes existing data selection methods as special cases, and comes with a theory tying the feasible set to the bias–variance tradeoff of each update and to the convergence of the training trajectory.
- **Method.** Group-Wise Subset Update, a broader family of data regularizers with tunable strength.
- **System.** System-level optimizations that realize these methods under the memory constraints of modern LLM training, with peak memory and per-step cost close to standard training.
- **Experiments.** Validation across SFT, RLHF, and RLVR, where the gains track the mismatch between the general and target data as predicted, together with extensive system benchmarks.

2 Dr. Post-Training: A Data Regularization Framework

We now introduce the **Dr. Post-Training** framework, which views general training data *not* as a pool of examples to select from, but as a *data regularizer* for optimizing a target objective.

2.1 Problem Setup and Framework

We consider post-training settings with abundant data from a *general training distribution* and limited data from a *target distribution* representing the downstream objective. This arises in: (i) general SFT with a curated mixture as training and a specific target benchmark (Lambert et al., 2025; Grattafiori et al., 2024; Qwen Team, 2025), (ii) domain-adaptive fine-tuning (Gururangan et al., 2020), and (iii) targeted instruction tuning (Xia et al., 2024a; He et al., 2024; Wang et al., 2024a).

Notation. Let $\ell(\theta; z)$ be a differentiable loss function over model parameters $\theta \in \mathbb{R}^d$ and a data sample z. Let the target distribution be $\mathbb{P}_\star$, the corresponding population loss and the gradient are then defined as $\mathcal{L}_\star(\theta) := \mathbb{E}_{z\sim\mathbb{P}_\star}[\ell(\theta; z)]$ and $g_\star(\theta) := \nabla_\theta \mathcal{L}_\star(\theta) \in \mathbb{R}^d$. Similarly, for the general training distribution $\mathbb{P}_{\mathrm{tr}}$, we define $g_{\mathrm{tr}}(\theta) := \nabla_\theta \mathcal{L}_{\mathrm{tr}}(\theta)$ such that $\mathcal{L}_{\mathrm{tr}}(\theta) := \mathbb{E}_{z\sim\mathbb{P}_{\mathrm{tr}}}[\ell(\theta; z)]$.

2.1.1 Framework

Target-driven one-step update. Modern post-training proceeds by iterative optimization (Wei et al., 2022; Schulman et al., 2017; Ouyang et al., 2022; Shao et al., 2024): at step t, the parameters are updated via $\theta_{t+1} = \theta_t - \eta_t u_t$ for step size $\eta_t > 0$ and direction $u_t \in \mathbb{R}^d$. We adopt a one-step target-driven view and choose u_t to minimize the next-step target loss within a *feasible set* U_t:

$$u_t = \arg\min_{u\in U_t} \mathcal{L}_\star(\theta_t - \eta_t u). \tag{1}$$

Training-batch-induced feasible set. At each step, we draw two batches: $B_t = \{z_i\}_{i=1}^{n} \overset{\text{i.i.d.}}{\sim} \mathbb{P}_{\text{tr}}$ and $B_t^\star = \{z_j^\star\}_{j=1}^{m} \overset{\text{i.i.d.}}{\sim} \mathbb{P}_\star$. For any set B, let $g_B(\theta_t) \coloneqq \frac{1}{|B|}\sum_{z\in B}\nabla_\theta \ell(\theta_t; z)$; we then write per-sample gradients $g_i \coloneqq \nabla_\theta \ell(\theta_t; z_i)$, $g_j^\star \coloneqq \nabla_\theta \ell(\theta_t; z_j^\star)$, and batch averages $\hat{g}_{\text{tr}} \coloneqq \frac{1}{n}\sum_i g_i$, $\hat{g}_\star \coloneqq \frac{1}{m}\sum_j g_j^\star$ (suppressing θ_t hereafter). The key modeling choice is that U_t depends only on B_t (typically through $\{g_i\}$), while $B_t^\star$ enters only through the target objective: *the target batch identifies the desired direction; the training batch constrains which directions are admissible*.

2.1.2 The dual perspective of data selection and data regularization

This formulation admits a dual perspective between *data selection* and *data regularization* (Figure 1):

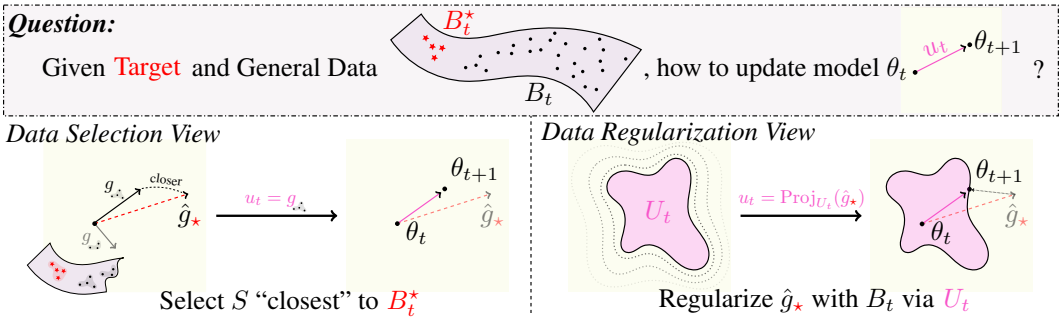

Figure 1: The dual view of data selection and data regularization.

The conventional *data selection* view treats the target as choosing which training examples are most useful (Xia et al., 2024a; He et al., 2024; Wang et al., 2024a). The *data regularization* view instead treats the target as specifying the ideal update direction, and the training batch as restricting it to U_t: without regularization one would optimize the target directly (unstable when target data are scarce); constraining to training-supported directions reduces instability at the cost of bias under distribution misalignment. The complexity of U_t thus controls a **bias–variance tradeoff** (Section 2.3).

2.2 Concrete instantiations of the feasible sets

We now turn the framework into concrete methods. We first need a tractable approximation of Eq. (1), since directly minimizing $\mathcal{L}_\star(\theta_t - \eta_t u)$ over $u \in U_t$ is generally intractable, especially when U_t is discrete. We therefore apply the majorization–minimization principle (Lange et al., 2000; Mairal, 2013; Lange, 2016) assuming that $\mathcal{L}_\star$ is β-smooth (Bottou et al., 2018; Nesterov, 2004), and substitute the population target gradient $g_\star$ by its batch estimate $\hat{g}_\star$. With the step size $\eta_t = 1/\beta$, this yields a Euclidean projection $u_t = \text{Proj}_{U_t}(\hat{g}_\star)$, or equivalently (derivation in Appendix C.1):

$$u_t = \arg\min_{u\in U_t} \|u - \hat{g}_\star\|^2. \tag{2}$$

2.2.1 Existing methods as special cases

Several existing training paradigms arise as special cases of Eq. (2) under different choices of U_t.

Target-Only Update. Choosing $U_t^\star \coloneqq \mathbb{R}^d$ imposes no directional constraint, so Eq. (2) yields $u_t = \hat{g}_\star$, i.e., training on the target distribution, as in fine-tuning solely on task-specific data (Zhou et al., 2023a; Gunasekar et al., 2023). It is unbiased for $g_\star$ but has high variance when m is small.

Full-Training Update. At the opposite extreme, the singleton $U_t^{\text{tr}} \coloneqq \{\hat{g}_{\text{tr}}\}$ imposes the strongest constraint and gives $u_t = \hat{g}_{\text{tr}}$, i.e., training on the general training distribution with no per-step target feedback, as in general-purpose SFT (Lambert et al., 2025; Grattafiori et al., 2024).

Global Subset Update. Existing data selection methods let the update be the average gradient of a selected subset: for a subset size k, define $U_t^{\text{glob}}(k) \coloneqq \{g_S = \frac{1}{k}\sum_{i\in S} g_i \colon S \subseteq [n], |S| = k\}$ so that Eq. (2) picks $S_t \in \arg\min_{|S|=k}\|g_S - \hat{g}_\star\|^2$ and sets $u_t = g_{S_t}$. Solving for S_t is combinatorial and balances a first-order *alignment* term, which decomposes into per-sample scores $s_i \coloneqq \langle g_i, \hat{g}_\star\rangle$,

against a second-order *redundancy* penalty $\|g_S\|^2$. Existing methods are tractable approximations of it (Appendix C.2): top-k selection (Han & Tsvetkov, 2021; Hu et al., 2024; He et al., 2024) and thresholding (Hu et al., 2025b) keep the highest-scoring samples and ignore the redundancy penalty, whereas greedy construction (Wang et al., 2024a) accounts for it at $O(nk)$ extra cost.

2.2.2 New data-regularization designs via group-wise decomposition

We further introduce a new family of data regularizers by relaxing the global coupling imposed by $U_t^{\mathrm{glob}}(k)$, where a single subset S is shared across all parameter coordinates. The regularization imposed by $U_t^{\mathrm{glob}}(k)$ is overly restrictive when different parameter groups align with the target signal in different ways: a training sample useful for one layer may be less informative for another. We instead partition the parameter coordinates into groups and let each group choose its own subset.

Group-Wise Subset Update. Let $\mathcal{G} = \{G_p\}_{p=1}^P$ be a partition of $[d]$ into P disjoint groups. For any vector $v \in \mathbb{R}^d$, let $v^{(p)}$ denote the subvector indexed by G_p. For a fixed subset size k, define

$$U_t^{\mathrm{grp}}(k;\mathcal{G}) := \left\{ u = (u^{(1)}, \dots, u^{(P)}) \colon u^{(p)} \in U_t^{(p)}(k), \forall p \in [P] \right\},$$

where $U_t^{(p)}(k) := \{\frac{1}{k}\sum_{i\in S} g_i^{(p)} \colon |S| = k\}$. Intuitively, each group G_p now chooses its own subset $S_{t,p}$. An important special case is the *Layer-Wise Subset Update* (Algorithm D.3), where each group corresponds to a single layer, since it enjoys an efficient implementation (Section 3).

Global Subset Update is recovered when all groups share one subset, so $U_t^{\mathrm{glob}}(k) \subseteq U_t^{\mathrm{grp}}(k;\mathcal{G})$, typically strictly: the group-wise set is larger, weakening data regularization and reducing bias.

Crucially, the feasible set factorizes as $U_t^{\mathrm{grp}}(k;\mathcal{G}) = \prod_{p=1}^P U_t^{(p)}(k)$, so Eq. (2) decomposes into P independent per-group selections (Appendix C.3). Hence, selection strategies such as top-k or thresholding apply within each group: e.g., group-wise top-k scores each training sample by $s_i^{(p)} := \langle g_i^{(p)}, \hat{g}_\star^{(p)} \rangle$ and selects per group. The partition $\mathcal{G}$ becomes a new design knob: coarser partitions impose stronger data regularization, finer partitions permit more flexible target-driven updates, producing a spectrum between Global Subset and Target-Only Updates analyzed in Section 2.3.

2.3 Bias–variance tradeoffs and convergence

We now quantify the bias–variance tradeoff of the feasible set. The feasible sets are nested, from the singleton U_t^{tr} to $U_t^\star = \mathbb{R}^d$: for fixed k and $\mathcal{G}$, $U_t^{\mathrm{glob}}(k) \subseteq U_t^{\mathrm{grp}}(k;\mathcal{G}) \subseteq U_t^\star$, and both collapse to U_t^{tr} at $k = n$, so a larger set means weaker data regularization. We bound the error of one update by two quantities of U_t that also govern convergence; proofs are in Appendices C.4 and C.5.

Two sources of error. Fix θ_t and take expectations over both batches. One step with update u decreases the expected target loss by at least $\frac{\eta_t}{2}(\|g_\star\|^2 - \mathrm{MSE}(u))$ (Theorem C.2), where $\mathrm{MSE}(u) := \mathbb{E}[\|u - g_\star\|^2]$, so we quantify $\mathrm{MSE}(u_t)$ for $u_t = \mathrm{Proj}_{U_t}(\hat{g}_\star)$, which can miss $g_\star$ for two reasons:

- **Bias.** Restricted to training-supported directions, U_t may contain nothing near $g_\star$.
- **Variance.** The projection uses $\hat{g}_\star = g_\star + \xi$, so the target-batch noise ξ perturbs its solution.

> **Intuition:** Only the noise along candidate differences matters, since the choice between $u, u' \in U_t$ depends on ξ only through $\langle u - u', \xi \rangle$; noise orthogonal to them does not.

Formally, for a feasible set U_t, we define the *bias* as $\mathcal{B}(U_t) := \mathbb{E}\big[\inf_{u\in U_t}\|u - g_\star\|^2\big]$, and *exposed variance* as $\mathcal{V}(U_t) := \mathbb{E}[\|\mathrm{Proj}_{\mathcal{D}(U_t)}(\xi)\|^2]$ where $\mathcal{D}(U_t) := \mathrm{span}\{u - u' \colon u, u' \in U_t\}$.

Theorem 2.1 (One-step error; informal version of Theorem C.6)**.** *For any feasible set U_t determined by the training batch, the projected update of Eq. (2) satisfies*

$$\sqrt{\mathrm{MSE}(u_t)} \le \sqrt{\mathcal{B}(U_t)} + 2\sqrt{\mathcal{V}(U_t)}.$$

Enlarging U_t can only decrease the bias and increase the exposed variance.

For target noise with variance at most σ^2/m per direction, Table 1 lists both quantities, with Σ_{tr} and $\Sigma_\star$ the gradient covariances under $\mathbb{P}_{\mathrm{tr}}$ and $\mathbb{P}_\star$. Full-Training Update pays the full mismatch but sees

Table 1: Bias and exposed variance of the four updates (Theorem 2.1).

	Full-Training	Global Subset	Group-Wise Subset	Target-Only
	← strong: $\mathcal{B}\uparrow$ $\mathcal{V}\downarrow$	data regularization	weak: $\mathcal{B}\downarrow$ $\mathcal{V}\uparrow$ →	
$\mathcal{B}(U_t)$	$\|g_{\mathrm{tr}} - g_\star\|^2 + \mathrm{tr}(\Sigma_{\mathrm{tr}})/n$	$\mathcal{B}(U_t^{\mathrm{glob}}(k))$	$\leq \mathcal{B}(U_t^{\mathrm{glob}}(k))$	0
$\mathcal{V}(U_t)$	0	$\leq (n-1)\sigma^2/m$	$\leq P(n-1)\sigma^2/m$	$\mathrm{tr}(\Sigma_\star)/m \leq d\sigma^2/m$

no target noise, Target-Only Update is unbiased but exposed to all d noise directions, and refining the partition lowers the bias while its exposed-variance bound grows at most P-fold. Squaring the bound gives $\mathrm{MSE}(u_t) \leq \mathcal{B}(U_t) + 4\sqrt{\mathcal{B}(U_t)\mathcal{V}(U_t)} + 4\mathcal{V}(U_t)$. Since $\mathcal{V}(U_t) = O(1/m)$ while the bias does not depend on m, the bound on the excess error $\mathrm{MSE}(u_t) - \mathcal{B}(U_t)$ decays as $1/m$ when $\mathcal{B}(U_t) = 0$ but only as $1/\sqrt{m}$ otherwise, the rate set by the cross term (Appendix C.4.1).

From one step to the trajectory. With Theorem 2.1 in hand, we ask when repeated updates converge on the target objective. By the descent bound above, step t decreases the target loss by at least $\frac{\eta_t}{2}(\|g_\star(\theta_t)\|^2 - \mathrm{MSE}(u_t))$; summed over steps, this bounds the average target gradient by the average one-step error, so convergence hinges on the error relative to the current gradient. An absolute bound on $\mathrm{MSE}(u_t)$ yields progress only while $\|g_\star(\theta_t)\|^2$ exceeds it, whereas a relative bound $\mathrm{MSE}(u_t) \leq q^2\|g_\star(\theta_t)\|^2$ with $q < 1$ retains at every step a $(1-q^2)$ fraction of the decrease of an exact target-gradient step. We say that the feasible sets *cover* the target gradient along the trajectory if $\sqrt{\mathcal{B}(U_t)} + 2\sqrt{\mathcal{V}(U_t)} \leq q\|g_\star(\theta_t)\| + \phi_t$ for every t, with $q < 1$ (Assumption 3): some feasible direction approximates the target gradient to within a fraction q of its length while exposing comparable noise, up to a floor $\phi_t \geq 0$ from the training–target mismatch and the target noise. Under this condition, summing the descent bound over steps yields the following guarantee.

Theorem 2.2 (Convergence on the target; informal version of Theorems C.10 and C.11)**.** *Let the feasible sets cover the target gradient with $q < 1$, $0 < \eta_t \leq 1/\beta$, and $T \geq 1$. Then*

$$\frac{\sum_{t<T}\eta_t\,\mathbb{E}[\|g_\star(\theta_t)\|^2]}{\sum_{t<T}\eta_t} \leq \frac{4}{1-q^2}\cdot\frac{\mathcal{L}_\star(\theta_0) - \inf_\theta \mathcal{L}_\star(\theta)}{\sum_{t<T}\eta_t} + \frac{2(1+q^2)}{(1-q^2)^2}\cdot\frac{\sum_{t<T}\eta_t\,\phi_t^2}{\sum_{t<T}\eta_t}.$$

If the floors vanish and the step size is constant, then $g_\star(\theta_t) \to 0$ almost surely, and under the Polyak–Łojasiewicz condition the target loss converges linearly and θ_t to a minimizer of $\mathcal{L}_\star$.

The first term is the optimization term, the initial suboptimality over the total step size, which vanishes as $\sum_t \eta_t \to \infty$; the second is a stationarity floor set by the part of the one-step error that does not shrink with the gradient, such as distribution mismatch and target noise. The average squared target gradient is thus driven to the floor, at rate $1/(\eta T)$ when the floor vanishes and $\eta_t \equiv \eta$; the second statement gives almost sure convergence. Refining the partition lowers the bias part of the coverage condition but raises its variance part, as the group-wise set covers the target gradient better but exposes more noise; a finer partition is certified when the coverage gained outweighs the noise exposed, and since exposed noise scales as $1/m$, larger target batches certify finer ones (Appendix C.5).

Takeaway: *Larger feasible sets reduce bias but raise exposed variance*: the update gains freedom to align with $g_\star$ but **also with the noise** ξ. The same two quantities govern convergence along the trajectory. Full-Training is stable but biased, Target-Only unbiased but unstable at small m, and the subset updates interpolate between them; which is best depends on the mismatch (Section 4).

2.4 Generalization beyond two-distribution setting

Although we developed the framework in the two-distribution setting, the underlying principle is more general: it requires (i) a ***training signal*** that can be optimized reliably but might introduce bias, and (ii) a ***target signal*** that is too noisy or costly to optimize directly. In the two-distribution setting, the target batch provides the noisy signal and the training batch induces the feasible set that stabilizes the update; the convergence results of Section 2.3 are stated for that setting, and the structure motivates the RL instantiations without extending those guarantees to them. Other post-training paradigms admit the same structure: in reinforcement learning, the target signal of interest is the expected reward, but optimizing it directly via policy gradient is high-variance, whereas training signals such as proximal policy optimization (PPO) (Schulman et al., 2017) or the group-relative estimator of group relative policy optimization (GRPO) (Shao et al., 2024) stabilize training at the cost of optimization bias. We demonstrate these principles in Sections 4.2 and 4.3.

3 EFFICIENT REALIZATION UNDER MEMORY CONSTRAINTS

Modern LLM training infrastructures are heavily constrained by memory bottlenecks, and efficiently realizing the proposed method under such tight memory limits requires nontrivial system-level optimizations. The proposed methods introduce extra computational dependencies on *per-sample* quantities that standard backpropagation does not preserve, which a naive implementation can resolve only at the cost of either an extra forward-backward pass or retaining the entire computation graph—both prohibitive at LLM scale. We develop three system-level optimizations that together make Dr. Post-Training practical, with full technical details deferred to Appendix D.

One-pass tensor lifetime scheduling (Appendix D.2). We design a customized tensor lifetime schedule that achieves the data-regularized update in a *single* forward-backward pass while preserving a peak memory footprint close to standard training. The key idea is to *swap* the forward-cached pre-activations with their backward-generated gradients at each layer, retaining only the minimal local information needed for downstream scoring and update assembly. The schedule naturally specializes to *layer-aligned partitions* where each group $G_p \in \mathcal{G}$ contains entire layers, e.g., Global Subset Update, Layer-Wise Subset Update (introduced in Algorithm D.3), and extends to general partitions, with a two-pass alternative when memory budgets are too tight to retain the required local tensors.

Efficient per-sample scoring (Appendix D.3). Per-sample scoring is the major computational overhead introduced by data-regularized updates. We revisit two existing methods (Direct, Ghost Inner Product (GIP) (Wang et al., 2024a)) and propose two new variants (**Per-Token Inner Product** (PIP), **Compressed Scoring**), and provide a systematic complexity analysis revealing regime-dependent trade-offs not characterized in prior work. The high-level idea behind PIP is to factor scoring as a matrix-vector product against the *aggregated* target gradient followed by a token-level dot product, thereby avoiding both the materialization of per-sample gradients (which Direct suffers from) and the quadratic dependence on sequence length (which GIP suffers from); this makes PIP the preferable exact scoring method whenever the context length is moderate relative to the per-layer hidden dimension. Beyond exact scoring, we also explore Compressed Scoring, which removes the dependence on the per-layer parameter dimension by working in a low-dimensional sketch, at some loss of selection fidelity; it is the cheapest backend at long contexts and many target examples.

Compatibility with memory-saving techniques (Appendix D.4). We show that the proposed methods integrate naturally with widely-used memory-saving techniques in modern LLM training pipelines, including low-rank adaptation (LoRA), memory-efficient subspace optimization (MeSO), activation checkpointing, and gradient accumulation. In particular, Layer-Wise Subset Update aligns well with *all* of these techniques: its fine-grained partition makes scoring, selection, and update assembly entirely *local* to a single layer, so it composes flexibly with any memory-saving schedule that operates at or above layer granularity without requiring our two-pass fallback.

4 EXPERIMENTS

Our experiments ask whether the data-regularization view yields useful update rules, and how the preferred regularization strength changes with the relation between the general pool and the target. Across SFT, RLHF, and RLVR, we compare standard training (Full-Training Update) and the state-of-the-art data selection baselines (variants of Global Subset Update), with the proposed Layer-Wise Subset Update. Target-Only Update is additionally reported in the question-answering setting (Section 4.1.1). Finally, every arm of a setting shares the hyperparameters of standard training without per-method tuning; since the curated updates in general converge faster and the least regularized one can overfit within the same budget (Appendix E.1.1), their reported gains are conservative.

Unless stated otherwise, all results are averaged over 5 seeds and reported as mean $\pm$ standard error.

4.1 SUPERVISED FINE-TUNING

We first compare the update rules on question answering, and then scale to reasoning and instruction following. Throughout SFT, each step draws $n = 8$ candidate training examples and $m = 1$ target example, and the curated updates keep $k = 4$ of the candidates by top-k selection.

4.1.1 QUESTION ANSWERING

We fine-tune LLAMA-3.2-1B (Grattafiori et al., 2024) with full-parameter, LoRA, and MeSO fine-tuning on four general/target pairs: `alpaca` (Taori et al., 2023)/`samsum` (Gliwa et al., 2019), `less-mix` (Xia et al., 2024a)/`tydiqa` (Clark et al., 2020), `triviaqa` (Joshi et al., 2017)/`nq_open` (Kwiatkowski et al., 2019), and `less-mix`/`squad` (Rajpurkar et al., 2016) (closed-book). The first two pairs are taken from GREATS (Wang et al., 2024a) (Global Subset Update), every pair uses a target set of only 16 examples, and alignment scores are computed with exact GIP.

Table 2: Question-answering SFT (LoRA): final-checkpoint Rouge-L (`samsum`) / F1 (%).

Method	`alpaca`/`samsum`	`less-mix`/`tydiqa`	`triviaqa`/`nq_open`	`less-mix`/`squad`
Full-Training	22.4 ±0.3	4.9 ±0.3	12.7 ±0.2	9.0 ±0.2
Global Subset	22.7 ±0.5	13.7 ±1.5	13.5 ±0.3	9.1 ±0.2
Layer-Wise Subset	29.3 ±1.4	48.3 ±1.7	**14.0 ±0.3**	**10.0 ±0.4**
Target-Only	**32.8 ±0.6**	**51.1 ±2.9**	9.3 ±0.4	8.7 ±0.4

Results. Table 2 reports the LoRA runs' downstream performance of all four methods; Appendix E.1.1 adds held-out perplexities and training dynamics. Layer-Wise Subset Update outperforms Global Subset Update, which improves over or matches Full-Training; the gap is largest under the broad `less-mix` pool, the most misaligned with its targets, where data regularization matters most. Target-Only Update (early-stopped, Appendix E.1.1) splits the pairs: it leads or ties on `samsum` and `tydiqa`, whose answers are summaries or spans of the given passage, so a handful of target examples fixes the answer style, and never comes first on the closed-book targets, which test knowledge they cannot supply. It thus measures what the target set alone carries, not how much of the pool to use, the question the curated updates answer against Full-Training.

A case study (Appendix E.1.2) suggests why Global Subset Update falls short: the layers rank the candidates differently, and its shared ranking follows a few of them, so most layers train on data chosen for another layer. Supporting this, the ablations in Appendix E.1.3 show that rescaling each layer's scores before summing them (layer-normalized Global Subset Update) does not close the gap, whereas relaxing the shared subset, with *Block-Wise* and *Sublayer-Wise* Subset Update grouping the layers of each transformer block or of each attention/MLP sublayer, moves performance monotonically from Global to Layer-Wise Subset Update in many cases. They also show that the gains are not an artifact of the smaller batch, as Random Subset controls bring no gain over Full-Training; that the ordering is unchanged under full-parameter fine-tuning and MeSO as well.

4.1.2 REASONING AND INSTRUCTION FOLLOWING

We next test the regime dependence predicted in Section 2.3 by varying the pool–target mismatch and the construction of the target set. We fine-tune QWEN3-1.7B-BASE (Qwen Team, 2025) with full-parameter fine-tuning and exact PIP scoring on four 32K-example pools: the instruction, reasoning, and mixed pools of `Dolci-Instruct` (Olmo Team, 2025) and the `Tulu 3` SFT mixture (Lambert et al., 2025). Each target set has 32 examples and is evaluated on OLMo 3's benchmarks for precise instruction following (`IFEval` (Zhou et al., 2023b), `IFBench` (Pyatkin et al., 2025)) and mathematics (`MATH500` (Lightman et al., 2024), `GSM8K` (Cobbe et al., 2021)): *Precise IF (`Dolci`)* and *Math (`Dolci`)* are held-out rows of `Dolci`'s own sources, whereas *Math (curated)* consists of `MATH`-train problems (Hendrycks et al., 2021) re-solved by QWEN3-32B with verified final answers, mimicking a small, carefully curated target set.

Results. Table 3 reports the results. Selection helps most where the target skill is diluted: on `Tulu 3`, both curated updates improve clearly over Full-Training on `IFEval`, and Layer-Wise Subset Update also on `MATH500`, whereas on the in-distribution `Dolci` pools the gains are smaller and the strongly regularized Full-Training and Global Subset Updates are competitive or best, consistent with the proposed tradeoff. Math gains also hinge on the target set: Math (`Dolci`) yields only modest `GSM8K` gains, whereas Math (curated) lets Layer-Wise Subset Update gain on both `MATH500` and `GSM8K`. Additional ablations over k and the target-set size are reported in Appendix E.1.4.

4.2 REINFORCEMENT LEARNING FROM HUMAN FEEDBACK

We next apply Dr. Post-Training to RLHF, going beyond the two-distribution setting (Section 2.4): the PPO surrogate now acts as the stabilizing training signal for the high-variance policy gradient.

Table 3: Reasoning and instruction-following SFT: downstream metric (%) at the final checkpoint. **Top:** accuracy on `IFEval` and `IFBench`. **Bottom:** accuracy on `MATH500` and `GSM8K`.

Method	Dolci instruction/Precise IF (Dolci)		Dolci mixed/Precise IF (Dolci)		Tulu 3/Precise IF (Dolci)	
	IFEval	IFBench	IFEval	IFBench	IFEval	IFBench
Full-Training	42.7 ±0.6	17.9 ±0.2	40.9 ±0.6	**17.9 ±0.3**	32.5 ±0.2	16.2 ±0.3
Global Subset	**44.5 ±0.2**	18.5 ±0.3	**41.8 ±0.3**	16.5 ±0.4	**35.3 ±0.3**	16.0 ±0.3
Layer-Wise Subset	44.1 ±0.4	**18.7 ±0.5**	41.7 ±0.4	17.1 ±0.3	**35.3 ±0.4**	**16.4 ±0.9**

Method	Dolci reasoning/Math (Dolci)		Dolci mixed/Math (Dolci)		Tulu 3/Math (Dolci)		Dolci reasoning/Math (curated)	
	MATH500	GSM8K	MATH500	GSM8K	MATH500	GSM8K	MATH500	GSM8K
Full-Training	63.6 ±0.5	79.9 ±0.2	**64.2 ±0.3**	78.8 ±0.1	57.1 ±0.5	78.8 ±0.2	63.6 ±0.5	79.9 ±0.2
Global Subset	**64.4 ±0.3**	**80.7 ±0.2**	**64.2 ±0.4**	**80.4 ±0.3**	58.3 ±0.4	79.4 ±0.2	64.0 ±0.2	79.9 ±0.1
Layer-Wise Subset	62.8 ±0.2	80.3 ±0.2	63.9 ±0.4	**80.4 ±0.3**	**60.6 ±0.6**	**79.8 ±0.3**	**65.1 ±0.5**	**81.0 ±0.2**

We follow the standard PPO (Schulman et al., 2017) detoxification pipeline (Hugging Face, 2023; Hu et al., 2025b) with `TRL` (von Werra et al., 2020) on GPT-NEO-2.7B (Black et al., 2021) with LoRA and exact PIP scoring, using prompts from `real-toxicity-prompts` (Gehman et al., 2020). Continuations are rewarded by `LFTW R4 Target` (Vidgen et al., 2021) and judged by the independent `da-electra-hatespeech-detection` classifier. We compare standard PPO (Full-Training), Iterative Influence-based Filtering (IIF) (Hu et al., 2025b) (offline selection once per PPO round), Global Subset Update, and Layer-Wise Subset Update (online selection at every mini-batch step), all with *negative score filtering*. We ablate the *target source* (self-referencing the rollout batch vs. fresh rollouts on held-out prompts) and the *target loss* (expected reward vs. training loss). The target gradient is computed once per PPO round and reused across its mini-batches: one extra backward pass per round, plus one target rollout batch for the held-out source.

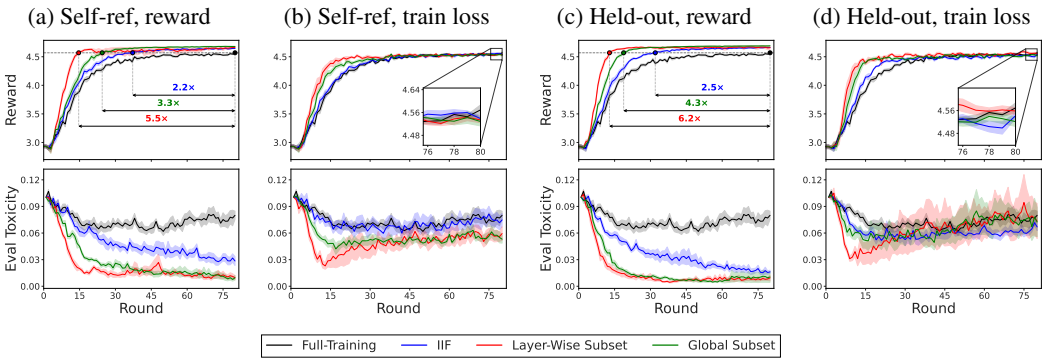

Figure 2: Detoxification RLHF for two target sources (self-reference/held-out) and two target losses (expected reward/training loss). **Top:** training reward. **Bottom:** evaluation toxicity.

Results. Three patterns stand out in Figure 2. First, the expected-reward target (Figures 2(a) and 2(c)) is far more effective than the PPO training loss (Figures 2(b) and 2(d)), and under it held-out rollouts converge faster than self-referencing. Second, the ordering Layer-Wise, Global, IIF, standard PPO holds early in training in every configuration, in both reward and toxicity: under the reward target the curated updates reach standard PPO's reward at step 80 about five, three to four, and two to three times sooner, and Layer-Wise and Global Subset Updates end an order of magnitude below standard PPO in toxicity, with IIF in between. Third, the reward target separates the methods at convergence in both metrics, whereas under the training-loss target the early speed-up vanishes at convergence (insets of Figures 2(b) and 2(d)), as expected. The gains also hold under three other toxicity judges and with compressed scoring (Appendices E.2.4 and E.2.5).

4.3 REINFORCEMENT LEARNING WITH VERIFIABLE REWARDS

Finally, we consider RLVR, where an automatic verifier provides the reward, using `Verl` (Sheng et al., 2024) with GRPO (Shao et al., 2024) on `MATH` (Hendrycks et al., 2021) and full-parameter fine-tuning of QWEN3-1.7B-BASE. We instantiate the framework as *in-run cross-validation*: every fourth round draws a small training batch as the target batch and estimates its expected verifier reward

by REINFORCE (as in Section 4.2); the resulting target gradient is reused until the next refresh, and exact PIP scores drive negative score filtering.

Results. Figure 3 shows the evaluation accuracy during the training. We see that Layer-Wise Subset Update performs best: Global Subset Update accelerates early training but ends near standard GRPO, whereas Layer-Wise sustains its advantage throughout and ends consistently higher (see Table 14). Both curated arms pay for the reward-based target signal with extra target rollouts, but as in RLHF the target gradient is captured once and reused, here across four rounds, so the amortized target cost is an eighth of the training rollouts (Appendix E.3). Layer-Wise Subset Update thus converges to a higher final accuracy than standard GRPO at a marginal overhead.

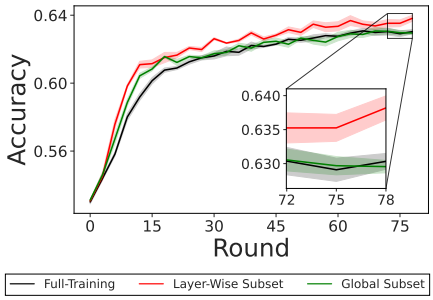

Figure 3: Math RLVR: evaluation accuracy over training rounds.

4.4 System efficiency

We benchmark training overhead and scoring cost on three models (SmolLM2-360M (Allal et al., 2025), TinyLlama-1.1B (Zhang et al., 2024), Llama-3.2-3B (Grattafiori et al., 2024)) on one `A40` GPU, using synthetic batches of fixed sequence length T (Table 4).

Table 4: System efficiency on one `A40` GPU ($n = 8$).

(a) Per-step component breakdown (ms) at $\mathsf{T} = 512$, $m = 1$, $k = 4$ with exact GIP scoring.

	SmolLM2-360M			TinyLlama-1.1B			Llama-3.2-3B		
Component	Full-Training	Layer-Wise Subset	Global Subset	Full-Training	Layer-Wise Subset	Global Subset	Full-Training	Layer-Wise Subset	Global Subset
Forward	78.7	90.5	89.1	134.0	145.7	146.2	340.0	387.6	387.4
Backward	155.2	194.1	191.6	278.5	306.6	306.3	821.0	752.4	758.8
`a.grad`	32.6	33.5	33.3	78.3	81.2	81.4	226.2	254.7	252.4
`scoring`	—	32.2	32.5	—	42.0	42.3	—	85.5	85.1
`w.grad`	27.5	21.6	21.2	75.9	44.4	44.4	338.1	126.2	136.0
`autograd`	95.1	106.8	104.5	124.3	139.1	138.1	256.7	286.0	285.3
Optimizer	26.6	26.7	26.6	80.7	80.7	80.8	235.5	235.4	235.5
Total	**260**	**311** (+20%)	**307** (+18%)	**493**	**533** (+8%)	**533** (+8%)	**1397**	**1375** (-2%)	**1382** (-1%)

(b) Standalone scoring cost (ms/step) sweeping T at $m = 1$ and m at $\mathsf{T} = 512$; cheapest exact method in bold.

	SmolLM2-360M						TinyLlama-1.1B						Llama-3.2-3B					
	sweep T			sweep m			sweep T			sweep m			sweep T			sweep m		
Scoring	512	2048	8192	1	4	8	512	2048	8192	1	4	8	512	2048	8192	1	4	8
Direct	56	153	575	56	66	80	132	366	1353	132	161	198	414	1128	4091	414	485	587
GIP	**29**	356	5950	**29**	95	182	**39**	519	10855	**39**	138	267	**73**	1001	18777	**73**	**260**	508
PIP	40	**122**	**463**	40	**52**	**70**	88	**335**	**1351**	88	**121**	**160**	251	**961**	**3880**	251	325	**429**
Compressed	29	47	162	29	29	29	25	57	218	25	25	25	39	106	412	39	39	40

Results. Table 4(a) reports the per-step breakdown: the curated updates add one target example and its scoring but skip half of the weight gradient (only the $k = 4$ kept samples enter it). On the two small, launch-bound models the fixed cost dominates ($+18$–20% and $+8\%$); once compute-bound, the saving pays for it, and on Llama-3.2-3B the curated step is 1–2% faster than Full-Training; peak memory is close to Full-Training (Table 16).

Table 4(b) isolates the scoring cost, the only component of Table 4(a) that the backend changes: GIP wins only at short contexts with $m = 1$, PIP is the cheapest exact method beyond them (by $\mathsf{T} = 2048$ or $m = 4$–8) and 5–13$\times$ cheaper than GIP at $\mathsf{T} = 8192$, and compressed scoring is cheaper still, at the accuracy cost of Appendix E.1.3.

5 Conclusion

We propose **Dr. Post-Training**, a data regularization framework for LLM post-training that uses abundant general training data as a regularizer on target-driven updates. This framework subsumes existing data selection methods and leads to natural generalizations whose regularization strength is tunable, clarifying their tradeoffs and convergence. Careful system optimization makes them efficient under modern memory constraints. Experiments across SFT, RLHF, and RLVR show that the best regularization strength tracks the mismatch between the general and target data: data-regularized updates gain most where the target is diluted, whereas standard training remains competitive when the two are aligned, and our system optimizations keep their per-step cost on par with standard training.

AI USE STATEMENT

We used generative AI tools to assist with writing, namely to edit and polish the text of this paper. We also implemented the experiment and developed the codebase with AI tools. Separately, one experimental input is model-generated: the Math (curated) target set of Section 4.1.2 consists of MATH-train problems whose solutions were re-generated by QWEN3-32B and verified against the reference answers (Appendix E.1.4). All AI-assisted text was reviewed and verified by the authors, and we take full responsibility for the final content of this work, including all text and claims.

ETHICS STATEMENT

This paper presents work whose goal is to advance the understanding of LLM post-training. By providing a principled framework and extensive system optimization, our work may help practitioners build more efficient and reliable post-training pipelines. We do not foresee specific negative societal consequences beyond those common to general improvements in LLM training.

REPRODUCIBILITY STATEMENT

The theoretical results of Section 2 are summarized there informally; their formal statements, assumptions, and complete proofs are given in Appendix C. Section 4 and Appendix E specify the models, datasets and their licenses, baselines, hyperparameters, and evaluation protocols for the SFT, RLHF, and RLVR experiments, and all reported results include standard errors. Appendix D describes the memory-efficient realization of the proposed methods in enough detail to reimplement them. Source code, including the optimized implementations of all baselines and the setup instructions needed to reproduce our results, is provided as supplementary material.

REFERENCES

Ahmad Ajalloeian and Sebastian U Stich. On the convergence of SGD with biased gradients. *arXiv preprint arXiv:2008.00051*, 2020.

Alon Albalak, Yanai Elazar, Sang Michael Xie, Shayne Longpre, Nathan Lambert, Xinyi Wang, Niklas Muennighoff, Bairu Hou, Liangming Pan, Haewon Jeong, Colin Raffel, Shiyu Chang, Tatsunori Hashimoto, and William Yang Wang. A survey on data selection for language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=XfHWcNTSHp. Survey Certification.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

Shai Ben-David, John Blitzer, Koby Crammer, Alex Kulesza, Fernando Pereira, and Jennifer Wortman Vaughan. A theory of learning from different domains. *Machine Learning*, 79(1-2):151–175, 2010.

Sid Black, Gao Leo, Phil Wang, Connor Leahy, and Stella Biderman. GPT-Neo: Large Scale Autoregressive Language Modeling with Mesh-Tensorflow, March 2021. URL https://doi.org/10.5281/zenodo.5297715. If you use this software, please cite it using these metadata.

Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM review*, 60(2):223–311, 2018.

Yihan Cao, Yanbin Kang, Chi Wang, and Lichao Sun. Instruction mining: Instruction data selection for tuning large language models. *arXiv preprint arXiv:2307.06290*, 2023.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, et al. Alpagasus: Training a better alpaca with fewer data. In *The Twelfth International Conference on Learning Representations*, 2024.

Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Sang Keun Choe, Hwijeen Ahn, Juhan Bae, Kewen Zhao, Minsoo Kang, Youngseog Chung, Adithya Pratapa, Willie Neiswanger, Emma Strubell, Teruko Mitamura, et al. What is your data worth to gpt? llm-scale data valuation with influence functions. In *Advances in Neural Information Processing Systems*, 2025.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Jonathan H Clark, Eunsol Choi, Michael Collins, Dan Garrette, Tom Kwiatkowski, Vitaly Nikolaev, and Jennimaria Palomaki. TyDi QA: A benchmark for information-seeking question answering in typologically diverse languages. *Transactions of the Association for Computational Linguistics*, 8: 454–470, 2020.

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

Hal Daumé III. Frustratingly easy domain adaptation. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pp. 256–263, 2007.

Junwei Deng, Yuzheng Hu, Pingbang Hu, Ting-Wei Li, Shixuan Liu, Jiachen T. Wang, Dan Ley, Qirun Dai, Benhao Huang, Jin Huang, Cathy Jiao, Hoang Anh Just, Yijun Pan, Jingyan Shen, Yiwen Tu, Weiyi Wang, Xinhe Wang, Shichang Zhang, Shiyuan Zhang, Ruoxi Jia, Himabindu Lakkaraju, Hao Peng, Weijing Tang, Chenyan Xiong, Jieyu Zhao, Hanghang Tong, Han Zhao, and Jiaqi W. Ma. A survey of data attribution: Methods, applications, and evaluation in the era of generative ai. *SSRN*, 2025. doi: 10.2139/ssrn.5451054. Available at SSRN: https://ssrn.com/abstract=5451054.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Kawin Ethayarajh, Yejin Choi, and Swabha Swayamdipta. Understanding dataset difficulty with $\mathcal{V}$-usable information. In *International Conference on Machine Learning*, pp. 5988–6008. PMLR, 2022.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky. Domain-adversarial training of neural networks. *Journal of machine learning research*, 17(59):1–35, 2016.

Samuel Gehman, Suchin Gururangan, Maarten Sap, Yejin Choi, and Noah A Smith. Realtoxicityprompts: Evaluating neural toxic degeneration in language models. *arXiv preprint arXiv:2009.11462*, 2020.

Saeed Ghadimi and Guanghui Lan. Stochastic first- and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368, 2013.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pp. 2242–2251. PMLR, 2019.

Bogdan Gliwa, Iwona Mochol, Maciej Biesek, and Aleksander Wawer. SAMSum corpus: A human-annotated dialogue dataset for abstractive summarization. In *Proceedings of the 2nd Workshop on New Frontiers in Summarization*, pp. 70–79. Association for Computational Linguistics, 2019.

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

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Jiang Guo, Darsh Shah, and Regina Barzilay. Multi-source domain adaptation with mixture of experts. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pp. 4694–4703, 2018.

Suchin Gururangan, Ana Marasović, Swabha Swayamdipta, Kyle Lo, Iz Beltagy, Doug Downey, and Noah A Smith. Don't stop pretraining: Adapt language models to domains and tasks. In *Proceedings of the 58th annual meeting of the association for computational linguistics*, pp. 8342–8360, 2020.

Xiaochuang Han and Yulia Tsvetkov. Influence tuning: Demoting spurious correlations via instance attribution and instance-driven updates. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 4398–4409, Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.findings-emnlp.374. URL https://aclanthology.org/2021.findings-emnlp.374/.

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

Elad Hazan. Introduction to online convex optimization. *Foundations and Trends in Optimization*, 2 (3-4):157–325, 2016.

Luxi He, Mengzhou Xia, and Peter Henderson. What is in your safe data? identifying benign data that breaks safety. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=Hi8jKh4HE9.

Yutong He, Pengrui Li, Yipeng Hu, Chuyan Chen, and Kun Yuan. Subspace optimization for large language models with convergence guarantees. In *Forty-second International Conference on Machine Learning*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

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

Pingbang Hu, Joseph Melkonian, Weijing Tang, Han Zhao, and Jiaqi W. Ma. GraSS: Scalable data attribution with gradient sparsification and sparse projection. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025a. URL https://openreview.net/forum?id=o0HgWRmyY1.

Pingbang Hu, Yuzheng Hu, Jiaqi W. Ma, and Han Zhao. A unified theory of random projection for influence functions. In *ICLR 2026 Workshop on Navigating and Addressing Data Problems for Foundation Models*, 2026.

Yuzheng Hu, Pingbang Hu, Han Zhao, and Jiaqi Ma. Most influential subset selection: Challenges, promises, and beyond. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 119778–119810. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/d8684e49752e06ac5e4b554b60ad212a-Paper-Conference.pdf.

Yuzheng Hu, Fan Wu, Haotian Ye, David Forsyth, James Zou, Nan Jiang, Jiaqi W. Ma, and Han Zhao. A snapshot of influence: A local data attribution framework for online reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025b. URL https://openreview.net/forum?id=sYK4yPDuT1.

Hugging Face. Detoxifying a language model using ppo. https://huggingface.co/docs/trl/en/detoxifying_a_lm, 2023. TRL documentation (v0.17.0), accessed May 8, 2025.

Michael F Hutchinson. A stochastic estimator of the trace of the influence matrix for laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

Hamish Ivison, Noah A Smith, Hannaneh Hajishirzi, and Pradeep Dasigi. Data-efficient finetuning using cross-task nearest neighbors. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 9036–9061, 2023.

Srinivasan Iyer, Xi Victoria Lin, Ramakanth Pasunuru, Todor Mihaylov, Daniel Simig, Ping Yu, Kurt Shuster, Tianlu Wang, Qing Liu, Punit Singh Koura, et al. Opt-iml: Scaling language model instruction meta learning through the lens of generalization. *arXiv preprint arXiv:2212.12017*, 2022.

Cathy Jiao, Weizhen Gao, Aditi Raghunathan, and Chenyan Xiong. On the feasibility of in-context probing for data attribution. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 5140–5155, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.286. URL https://aclanthology.org/2025.findings-naacl.286/.

Mandar Joshi, Eunsol Choi, Daniel S Weld, and Luke Zettlemoyer. Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1601–1611, 2017.

Hamed Karimi, Julie Nutini, and Mark Schmidt. Linear convergence of gradient and proximal-gradient methods under the Polyak-łojasiewicz condition. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 795–811, 2016.

Angelos Katharopoulos and François Fleuret. Not all samples are created equal: Deep learning with importance sampling. In *International conference on machine learning*, pp. 2525–2534. PMLR, 2018.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Dawid Jan Kopiczko, Sagar Vaze, Tijmen Blankevoort, and Yuki M Asano. Data repetition beats data scaling in long-cot supervised fine-tuning. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=SbmQuvutVt.

Po-Nien Kung, Fan Yin, Di Wu, Kai-Wei Chang, and Nanyun Peng. Active instruction tuning: Improving cross-task generalization by training on prompt sensitive tasks. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1813–1829, 2023.

Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, et al. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7:453–466, 2019.

Hynek Kydlíček. Math-Verify: Math verification library. https://github.com/huggingface/Math-Verify, 2025. Version 0.9.0.

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

Guanghui Lan. *First-order and stochastic optimization methods for machine learning*, volume 1. Springer, 2020.

Kenneth Lange. *MM optimization algorithms*. SIAM, 2016.

Kenneth Lange, David R Hunter, and Ilsoon Yang. Optimization transfer using surrogate objective functions. *Journal of computational and graphical statistics*, 9(1):1–20, 2000.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Chen Ling, Xujiang Zhao, Jiaying Lu, Chengyuan Deng, Can Zheng, Junxiang Wang, Tanmoy Chowdhury, Yun Li, Hejie Cui, Xuchao Zhang, et al. Domain specialization as the key to make large language models disruptive: A comprehensive survey. *ACM Computing Surveys*, 58(3):1–39, 2025.

Zijian Liu, Ta Duy Nguyen, Thien Hang Nguyen, Alina Ene, and Huy Nguyen. High probability convergence of stochastic gradient methods. In *International conference on machine learning*, pp. 21884–21914. PMLR, 2023.

Ilya Loshchilov and Frank Hutter. Online batch selection for faster training of neural networks. *arXiv preprint arXiv:1511.06343*, 2015.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Keming Lu, Hongyi Yuan, Zheng Yuan, Runji Lin, Junyang Lin, Chuanqi Tan, Chang Zhou, and Jingren Zhou. #instag: Instruction tagging for analyzing supervised fine-tuning of large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=pszewhybU9.

Julien Mairal. Optimization with first-order surrogate functions. In *International conference on machine learning*, pp. 783–791. PMLR, 2013.

Joseph Melkonian. Learning which data to learn: The terrific meta-optimizer, 2025. URL https://joemelko.github.io/blog.html?post=TopicReweighting.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, and Yarin Gal. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 15630–15649. PMLR, 2022.

Swaroop Mishra, Daniel Khashabi, Chitta Baral, and Hannaneh Hajishirzi. Cross-task generalization via natural language crowdsourcing instructions. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3470–3487, 2022.

Niklas Muennighoff, Hongjin Su, Liang Wang, Nan Yang, Furu Wei, Tao Yu, Amanpreet Singh, and Douwe Kiela. Generative representational instruction tuning. In *International Conference on Learning Representations*, 2025.

Aashiq Muhamed, Oscar Li, David Woodruff, Mona Diab, and Virginia Smith. Grass: Compute efficient low-memory llm training with structured sparse gradients. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 14978–15003, 2024.

Nihal V Nayak, Paula Rodriguez-Diaz, Neha Hulkund, Sara Beery, and David Alvarez-Melis. A critical look at targeted instruction selection: Disentangling what matters (and what doesn't). *arXiv preprint arXiv:2602.14696*, 2026.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87 of *Applied Optimization*. Kluwer Academic Publishers, 2004.

Olmo Team. Olmo 3. *arXiv preprint arXiv:2512.13961*, 2025.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010.

Sneh Pandya, Purvik Patel, Brian D Nord, Mike Walmsley, and Aleksandra Ćiprijanović. Sidda: Sinkhorn dynamic domain adaptation for image classification with equivariant neural networks. *Machine Learning: Science and Technology*, 6(3):035032, 2025.

Ethan Perez, Douwe Kiela, and Kyunghyun Cho. True few-shot learning with language models. *Advances in neural information processing systems*, 34:11054–11070, 2021.

Boris T Polyak. Gradient methods for the minimisation of functionals. *USSR Computational Mathematics and Mathematical Physics*, 3(4):864–878, 1963.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. *Advances in Neural Information Processing Systems*, 33: 19920–19930, 2020.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=yfYgwjj5F8.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 2383–2392, 2016.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 3505–3506, 2020.

Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3982–3992. Association for Computational Linguistics, 2019.

Adithya Renduchintala, Tugrul Konuk, and Oleksii Kuchaiev. Tied-lora: Enhancing parameter efficiency of lora with weight tying. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 8694–8705, 2024.

John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. doi: 10.64434/tml.20250929. https://thinkingmachines.ai/blog/lora/.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

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

Zhennan Shen, Yanshu Li, Qingyu Yin, Chak Tou Leong, Zhilin Wang, Yanxu Chen, Rongduo Han, Sunbowen Lee, and Yi R Fung. On the geometry of on-policy distillation. *arXiv preprint arXiv:2606.07082*, 2026.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Ying Sheng, Shiyi Cao, Dacheng Li, Coleman Hooper, Nicholas Lee, Shuo Yang, Christopher Chou, Banghua Zhu, Lianmin Zheng, Kurt Keutzer, Joseph E. Gonzalez, and Ion Stoica. S-lora: Serving thousands of concurrent lora adapters. *arXiv preprint arXiv:2311.03285*, 2023.

Masashi Sugiyama, Matthias Krauledat, and Klaus-Robert Müller. Covariate shift adaptation by importance weighted cross validation. *Journal of Machine Learning Research*, 8(35):985–1005, 2007.

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.

Sharan Vaswani, Francis Bach, and Mark Schmidt. Fast and faster convergence of sgd for over-parameterized models and an accelerated perceptron. In *The 22nd international conference on artificial intelligence and statistics*, pp. 1195–1204. PMLR, 2019.

Roman Vershynin. *High-Dimensional Probability: An Introduction with Applications in Data Science*. Cambridge University Press, 2018.

Bertie Vidgen, Tristan Thrush, Zeerak Waseem, and Douwe Kiela. Learning from the worst: Dynamically generated datasets to improve online hate detection. In *ACL*, 2021.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Jiachen Tianhao Wang, Tong Wu, Dawn Song, Prateek Mittal, and Ruoxi Jia. Greats: Online selection of high-quality data for llm training in every iteration. *Advances in Neural Information Processing Systems*, 37:131197–131223, 2024a.

Jingtan Wang, Xiaoqiang Lin, Rui Qiao, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. Helpful or harmful data? fine-tuning-free shapley attribution for explaining language model predictions. In *Forty-first International Conference on Machine Learning*, 2024b. URL https://openreview.net/forum?id=WSpPC1Jm0p.

Jingtan Wang, Xiaoqiang Lin, Rui Qiao, Pang Wei Koh, Chuan-Sheng Foo, and Bryan Kian Hsiang Low. NICE data selection for instruction tuning in LLMs with non-differentiable evaluation metric. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 63662–63689. PMLR, 2025.

Yiming Wang, Yu Lin, Xiaodong Zeng, and Guannan Zhang. Multilora: Democratizing lora for better multi-task learning. *arXiv preprint arXiv:2311.11501*, 2023a.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36:74764–74786, 2023b.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=gEZrGCozdqR.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning*, pp. 54104–54132. PMLR, 2024a.

Wenhan Xia, Chengwei Qin, and Elad Hazan. Chain of lora: Efficient fine-tuning of language models via residual learning. *arXiv preprint arXiv:2401.04151*, 2024b.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. *Advances in neural information processing systems*, 32, 2019.

Longteng Zhang, Lin Zhang, Shaohuai Shi, Xiaowen Chu, and Bo Li. LoRA-FA: Memory-efficient low-rank adaptation for large language models fine-tuning. *arXiv preprint arXiv:2308.03303v1*, 2023.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. Tinyllama: An open-source small language model, 2024.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. Galore: Memory-efficient llm training by gradient low-rank projection. In *International Conference on Machine Learning*, pp. 61121–61143. PMLR, 2024.

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

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

A Related work

A.1 Data optimization for LLM post-training

Data optimization for LLM post-training aims to improve downstream performance by prioritizing high-utility training examples from large, heterogeneous corpora. We summarize several common families and refer readers to Albalak et al. (2024); Deng et al. (2025) for a comprehensive overview.

Most existing methods approach this as a *data selection* problem, leveraging downstream performance-related signals to determine which training examples the model sees. These span heuristic filtering and quality scoring (Bai et al., 2022; Ethayarajh et al., 2022; Chen et al., 2024; Ivison et al., 2023; Lu et al., 2024; Kung et al., 2023), dataset-level mixture design (Cao et al., 2023; Wei et al., 2022; Iyer et al., 2022; Wang et al., 2023b), and offline example-level selection via influence-style estimators, validation-based scoring, or importance weighting (Xia et al., 2024a; He et al., 2024; Wang et al., 2024b; Koh & Liang, 2017; Pruthi et al., 2020; Ghorbani & Zou, 2019; Melkonian, 2025). Online methods adapt selection during training using per-step signals (e.g., loss- or gradient-based) to dynamically choose or weight examples (Wang et al., 2024a; Jiao et al., 2025; Hu et al., 2025b), better tracking non-stationary utility at the cost of tighter compute constraints. In contrast, our proposed framework takes the *dual* perspective, where the training examples are used to regularize the admissible downstream performance-driven updates, and unlock a novel and broader design space.

A.2 Memory saving techniques in LLM post-training

Modern LLM training is typically under stringent computational constraints, especially memory bottlenecks. Extensive research effort has therefore been devoted to parameter-efficient methods and memory-saving scheduling techniques that allow post-training to fit within available hardware budgets.

A common strategy restricts learning to a low-dimensional subspace. Low-Rank Adaptation (LoRA) (Hu et al., 2022) injects trainable low-rank adapters into selected layers while keeping the pretrained parameters frozen, significantly reducing memory and compute requirements (Han et al., 2024; Renduchintala et al., 2024; Sheng et al., 2023; Zhang et al., 2023; Xia et al., 2024b; Wang et al., 2023a; Dettmers et al., 2023). Complementary to LoRA, Memory-efficient Subspace Optimization (MeSO) methods (Zhao et al., 2024; Muhamed et al., 2024; He et al., 2025) instead compress optimizer states and gradients by projecting them into a low-dimensional representation, applying updates in that space, and mapping them back to the original parameter space for actual parameter updates. For further details, please see Appendix B.

Complementary to low-dimensional subspace methods, memory-saving techniques restructure how computation is scheduled to further lower peak memory. Activation checkpointing (Chen et al., 2016) discards intermediate activations during the forward pass and recomputes them on-the-fly during the backward pass, trading additional computation for reduced memory footprint. Another technique, called gradient accumulation, is widely adopted in popular large-scale training frameworks (Rasley et al., 2020; von Werra et al., 2020; Sheng et al., 2024), partitions a large batch into micro-batches processed sequentially, accumulating gradients across micro-batches to simulate a larger effective batch size at a fraction of the peak memory cost. Both techniques preserve the optimization trajectory exactly and are widely composed with parameter-efficient methods in practice, making the resulting memory management landscape particularly intricate for methods that introduce additional per-sample computational dependencies beyond standard training.

In this work, we demonstrate that our proposed method integrates seamlessly with these techniques, showing its compatibility with common post-training pipelines (Appendix D.4).

A.3 Connection to domain adaptation

The mismatch between general-purpose pretraining corpora and a narrow target task is a recurring theme in transfer learning and domain adaptation (Pan & Yang, 2010). A long line of work has studied when and how such transfer succeeds, highlighting source–target divergence and the risk of negative transfer as central concerns (Ben-David et al., 2010). Classical remedies operate at the instance level: reweighting examples under covariate shift (Sugiyama et al., 2007), adapting feature representations (Daumé III, 2007), aligning domains adversarially (Ganin et al., 2016), or combining

multiple source domains (Guo et al., 2018). In the LLM era, continued in-domain or task-adaptive pretraining has become the de facto approach for reducing this mismatch (Gururangan et al., 2020).

Our framework addresses the same general training–target gap, but at a fundamentally different level of abstraction. Where instance reweighting corrects the empirical risk via density ratios and representation alignment enforces domain invariance in hidden states, both are specific algorithmic choices for bridging the domain gap. Dr. Post-Training instead formulates general training data as a *regularizer* on the target-driven update direction, yielding an explicit per-step bias–variance tradeoff governed by the feasible set U. This perspective not only subsumes existing instance-level strategies as special cases of the feasible set but also opens the door to *beyond-instance* designs such as Group-Wise Subset Update, where different parameter groups draw on different training subsets within a single optimization step—a capability that falls outside the scope of standard domain-adaptation formulations.

B Implementation details for LoRA and MeSO

We briefly review the two low-dimensional parameter training methods mentioned in Appendix A, LoRA and MeSO.

We use the notation of Appendix D.1: $l \in [L]$ indexes the linear layers, $a_{i,\tau}^{(l)}$ and $e_{i,\tau}^{(l)}$ are the input activation and the pre-activation of sample i at token $\tau \in [\mathsf{T}]$, and `Forward`, `Backward`, and `Release` are the primitives of Algorithm D.1. For a linear layer l, we write its weight matrix as $W^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l)} \times d_{\text{in}}^{(l)}}$, where $d_{\text{in}}^{(l)}$ and $d_{\text{out}}^{(l)}$ denote the input and output dimensions; unlike Appendix D.1, we do not assume $d_{\text{in}}^{(l)} = d_{\text{out}}^{(l)} = \sqrt{d/L}$.

B.1 LoRA

LoRA (Hu et al., 2022) reparameterizes each linear layer l as $W^{(l)} + B^{(l)}A^{(l)}$, where $B^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l)} \times r}$ and $A^{(l)} \in \mathbb{R}^{r \times d_{\text{in}}^{(l)}}$ are low-rank adapter matrices with *rank* $r \ll \min(d_{\text{in}}^{(l)}, d_{\text{out}}^{(l)})$. During training, the pretrained weight $W^{(l)}$ is kept frozen, and only the adapter parameters $A^{(l)}$ and $B^{(l)}$ are updated. We collectively refer to these trainable parameters as the *adapters*. Because the number of trainable parameters is much smaller than in full fine-tuning, LoRA substantially reduces memory usage: gradients and optimizer states (which can be several times larger than the gradients themselves) need to be stored only for the adapters rather than for the full weight matrix.

B.2 MeSO

MeSO (Zhao et al., 2024) maintains optimizer states in a compressed subspace of dimension $\sum_l \kappa^{(l)}$ rather than the full parameter space $\mathbb{R}^d$. At each layer l, MeSO uses a projection $\Pi^{(l)} \in \mathbb{R}^{\kappa^{(l)} \times d^{(l)}}$ (where $d^{(l)} = d_{\text{in}}^{(l)} d_{\text{out}}^{(l)}$ is the number of parameters, $\kappa^{(l)}$ is the compressed dimension for layer l, and $d = \sum_l d^{(l)}$) to compress the vectorized gradient $\text{vec}(g^{(l)})$. The standard MeSO update is given in Algorithm B.1.

Compared to LoRA, MeSO achieves memory savings through a different mechanism. LoRA reduces memory by restricting training to a small set of adapter parameters, so gradients and optimizer states are stored only for those parameters. In contrast, MeSO keeps the original parameterization of the model, but performs optimization in a low-dimensional subspace. As a result, gradient-related quantities such as optimizer states need only be maintained in the compressed space rather than in the full parameter space. This can lead to even greater memory savings, especially when the subspace dimension is smaller than the number of trainable parameters used by LoRA.[1]

B.2.1 Subspace refreshing

We note that in many MeSO methods, each projection $\Pi^{(l)}$ is periodically refreshed from Π_{old} to Π_{new} (we drop the layer index in what follows) to allow updating different subspaces, which requires

[1]Technically, it is possible to combine LoRA and MeSO. However, this leads to instability and is not a common practice.

Algorithm B.1: Standard MeSO Training

Data: Model θ_t, training data $\{z_i\}_{i=1}^n$, learning rate η_t, projections $\{\Pi^{(l)}\}_{l\in[L]}$
Result: Updated model parameters θ_{t+1}

$(\ell, \{a_i^{(l)}, e_i^{(l)}\}_{l,i}) \leftarrow$ `Forward` $(\theta_t, \{z_i\}_{i=1}^n)$
for $l = L, \dots, 1$ **do**
  $\{\partial\ell/\partial e_i^{(l)}\}_{i=1}^n \leftarrow$ `Backward` $(\ell, \theta_t^{(l)})$
  $u_t^{(l)} \leftarrow \frac{1}{n}\sum_{i=1}^n \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$
  `Release` $(a_i^{(l)}, \partial\ell/\partial e_i^{(l)})$ for all i
  $\widetilde{u}_t^{(l)} \leftarrow \Pi^{(l)}(u_t^{(l)})$ `// Compress to` $\mathbb{R}^{\kappa^{(l)}}$
  $\theta_{t+1}^{(l)} \leftarrow \theta_t^{(l)} - \eta_t \cdot ((\Pi^{(l)})^\top \widetilde{u}_t^{(l)})$ `// Back-project and update`
  `Release` $(\widetilde{u}_t^{(l)})$
return θ_{t+1}

transferring the optimizer's momentum states to the new subspace. Prior works omit this nuance by setting the momentum states to zero for simplicity (Zhao et al., 2024). Here, we provide a strategy for AdamW (Loshchilov & Hutter, 2019) to mitigate this.

First moment. For the *first moment* $\hat{m}_{\text{old}} \in \mathbb{R}^{\kappa^{(l)}}$, the transfer is straightforward: back-project to full space via $\Pi_{\text{old}}^\top$, then forward-project via Π_{new}.

Second moment. For the *second moment* $\hat{v}_{\text{old}} \in \mathbb{R}^{\kappa^{(l)}}$, a naive linear transformation is incorrect since variance is a quadratic quantity. In our implementation we consider a diagonal covariance approximation $\hat{\Sigma}_{\text{old}} \approx \operatorname{diag}(\hat{v}_{\text{old}})$. This allows us to transform the second moment as:

$$\hat{v}_{\text{new}} = \operatorname{diag}(M \operatorname{diag}(\hat{v}_{\text{old}}) M^\top) = (M \odot M)\hat{v}_{\text{old}},$$

where $M = \Pi_{\text{new}}\Pi_{\text{old}}^\top$. When M is too large to form explicitly, this diagonal can be estimated stochastically via Hutchinson's method (Hutchinson, 1989): $\hat{v}_{\text{new}} \approx \frac{1}{N_{\text{probe}}} \sum_{j=1}^{N_{\text{probe}}} r_j \odot (M \operatorname{diag}(\hat{v}_{\text{old}}) M^\top r_j)$ with Rademacher probes $r_j \in \{-1,1\}^{\kappa^{(l)}}$.

C Omitted details from Section 2

This section collects the technical results deferred from Section 2. Appendix C.1 derives the projection formulation (Eq.(2)) from the majorization–minimization relaxation and notes its optimizer-aware variant (Theorem C.1), Appendix C.2 derives the objective of Global Subset Update, and Appendix C.3 provides a derivation on solving Group-Wise Subset Update. Appendix C.4 states and proves the one-step results summarized in Theorem 2.1: the descent lemma (Theorem C.2) that reduces the comparison of methods to their MSE, the exact endpoints (Theorem C.3), and the noisy projection oracle inequality (Theorem C.6); Appendix C.4.1 derives the rates in the target batch size, Appendix C.4.2 compares the oracle inequality with the classical finite-class bound (Theorem C.8) and unifies the two (Theorem C.9), and Appendix C.4.3 sketches the resulting regimes. Appendix C.5 states and proves the trajectory results summarized in Theorem 2.2 (Theorems C.10 and C.11) from a generic stochastic-descent theorem.

For convenience, we collect all the assumptions used throughout the proofs.

Assumption 1 (Target smoothness)**.** *The target population loss $\mathcal{L}_\star$ is β-smooth for some $\beta > 0$, and bounded below, $\mathcal{L}_\star^{\inf} := \inf_\theta \mathcal{L}_\star(\theta) > -\infty$.*

Assumption 2 (Sub-Gaussian noise)**.** *The target gradient noise $\xi := \hat{g}_\star - g_\star$ is sub-Gaussian with parameter $\sigma/\sqrt{m}$ for some $\sigma > 0$, i.e., $\mathbb{E}[\exp(\lambda\langle\xi, v\rangle) \mid \theta_t] \le \exp(\lambda^2\sigma^2\|v\|^2/(2m))$ for all $v \in \mathbb{R}^d$ and $\lambda \in \mathbb{R}$.*

Assumption 3 (Coverage and directional noise along the trajectory)**.** *There are constants $\rho \in [0, 1)$, $\omega \ge 0$ and deterministic $b_t, \varsigma_t \ge 0$ such that, almost surely for every t, $\mathcal{B}(U_t) \le \rho\|g_\star(\theta_t)\|^2 + b_t^2$ and $\mathcal{V}(U_t) \le \omega\|g_\star(\theta_t)\|^2 + \varsigma_t^2$, where $\mathcal{B}(U_t)$ and $\mathcal{V}(U_t)$ are the bias and the exposed variance of Section 2.3, conditional on the history up to step t.*

Section 2.3 states this condition in the combined form $\sqrt{\mathcal{B}(U_t)} + 2\sqrt{\mathcal{V}(U_t)} \le q\|g_\star(\theta_t)\| + \phi_t$ with $q := \sqrt{\rho} + 2\sqrt{\omega}$ and $\phi_t := b_t + 2\varsigma_t$; Assumption 3 implies it (Theorem C.12), and the trajectory proofs use only this combined form.

Assumption 4 (Polyak–Łojasiewicz condition (Polyak, 1963; Karimi et al., 2016))**.** *There is $\mu > 0$ such that $\|g_\star(\theta)\|^2 \ge 2\mu\big(\mathcal{L}_\star(\theta) - \mathcal{L}_\star^{\inf}\big)$ for all $\theta \in \mathbb{R}^d$.*

Assumption 5 (Bounded gradients)**.** *There exists $C > 0$ such that $\|g_i\| \le C$ for all $i \in [n]$.*

Assumptions 1 and 2 are used by the one-step results of Appendix C.4, Assumptions 3 and 4 only by the trajectory results of Appendix C.5, and Assumption 5 only by the classical finite-class bound (Theorem C.8).

C.1 Derivation of the projection formulation

By Assumption 1, the target loss admits the quadratic majorization

$$\mathcal{L}_\star(\theta_t - \eta_t u) \le \mathcal{L}_\star(\theta_t) - \eta_t\langle g_\star, u\rangle + \frac{\beta\eta_t^2}{2}\|u\|^2.$$

Minimizing the upper bound on the right-hand side over $u \in U_t$, dropping the constant $\mathcal{L}_\star(\theta_t)$, and substituting the target batch estimate $\hat{g}_\star$ for the unknown $g_\star$ yields

$$u_t = \arg\max_{u\in U_t}\left\{\langle \hat{g}_\star, u\rangle - \frac{\beta\eta_t}{2}\|u\|^2\right\} \iff u_t \in \arg\min_{u\in U_t}\left\|u - \frac{\hat{g}_\star}{\beta\eta_t}\right\|^2,$$

where the equivalence follows from completing the square:

$$\langle \hat{g}_\star, u\rangle - \frac{\beta\eta_t}{2}\|u\|^2 = -\frac{\beta\eta_t}{2}\left\|u - \frac{1}{\beta\eta_t}\hat{g}_\star\right\|^2 + \frac{\|\hat{g}_\star\|^2}{2\beta\eta_t}.$$

Setting $\eta_t = 1/\beta$ (Bottou et al., 2018) absorbs the scaling, giving Eq. (2).

Remark C.1 (Optimizer-aware selection)**.** *The analysis treats u_t as the parameter step, whereas the experiments pass u_t to AdamW, as the compared selection methods do, so all arms are compared under a common optimizer. Freezing the moment estimates within a step linearizes the AdamW update to $\eta_t P_t u_t$ with a diagonal preconditioner P_t, and the same majorization then replaces Eq. (2) by $\arg\min_{u\in U_t}\|P_t u - \hat{g}_\star\|^2$, whose first-order scores are $\langle g_i, P_t\hat{g}_\star\rangle$: optimizer-aware selection reuses the scoring machinery of Section 3 with a preconditioned target gradient. We leave this simple extension as a future work.*

C.2 Derivation of the Global Subset objective

For the Global Subset Update with feasible set $U_t^{\mathrm{glob}}(k) = \{g_S = \frac{1}{k}\sum_{i\in S} g_i \colon S \subseteq [n], |S| = k\}$, Eq. (2) reduces to

$$S_t \in \arg\min_{S\subseteq[n],|S|=k}\|g_S - \hat{g}_\star\|^2, \qquad u_t = g_{S_t}.$$

Expanding the squared norm and dropping the term $\|\hat{g}_\star\|^2$ (independent of S) gives the equivalent objective

$$\min_{S\subseteq[n],|S|=k}\|g_S\|^2 - 2\langle g_S, \hat{g}_\star\rangle.$$

The two terms admit natural interpretations:

- *Alignment term.* $2\langle g_S, \hat{g}_\star\rangle = \frac{2}{k}\sum_{i\in S}\langle g_i, \hat{g}_\star\rangle$, favoring samples whose per-sample gradient aligns with the target gradient. This term decomposes additively into independent per-sample scores $s_i := \langle g_i, \hat{g}_\star\rangle$.
- *Redundancy penalty.* $\|g_S\|^2 = \frac{1}{k^2}\sum_{i,i'\in S}\langle g_i, g_{i'}\rangle$, discouraging selecting highly correlated training gradients.

The redundancy penalty couples all samples in S, making exact optimization intractable; the selection rules used in practice (Section 2.2.1) are tractable approximations of this combinatorial problem.

C.3 SEPARABILITY OF THE GROUP-WISE PROJECTION

Because the group-wise feasible set factorizes as $U_t^{\mathrm{grp}}(k;\mathcal{G}) = \prod_{p=1}^{P} U_t^{(p)}(k)$ and the squared Euclidean norm decomposes as $\|v\|^2 = \sum_{p=1}^{P} \|v^{(p)}\|^2$, the projection problem in Eq. (2) decomposes into independent per-group subproblems:

$$\min_{u \in U_t^{\mathrm{grp}}(k;\mathcal{G})} \|u - \hat{g}_\star\|^2 = \sum_{p=1}^{P} \min_{u^{(p)} \in U_t^{(p)}(k)} \|u^{(p)} - \hat{g}_\star^{(p)}\|^2.$$

Equivalently, for each group p, one chooses

$$S_{t,p} \in \operatorname*{arg\,min}_{S \subseteq [n], |S|=k} \|g_S^{(p)} - \hat{g}_\star^{(p)}\|^2, \qquad u_t^{(p)} = g_{S_{t,p}}^{(p)} = \frac{1}{k} \sum_{i \in S_{t,p}} g_i^{(p)}.$$

Because each per-group subproblem is structurally identical to the Global Subset Update objective (Appendix C.2), the same selection rules apply independently within each group, with the per-group scores $s_i^{(p)} \coloneqq \langle g_i^{(p)}, \hat{g}_\star^{(p)} \rangle$.

C.4 ONE-STEP ANALYSIS: FORMAL STATEMENTS AND PROOFS

This section states and proves the one-step results summarized in Theorem 2.1. Throughout, θ_t is fixed and $\mathrm{MSE}(u) \coloneqq \mathbb{E}[\|u - g_\star\|^2 \mid \theta_t]$, with expectation over B_t and $B_t^\star$. We first prove the descent lemma that reduces one-step progress to the MSE.

Lemma C.2 (Descent lemma)**.** *Assume Assumption 1. Fix θ_t and let $0 < \eta_t \le 1/\beta$. For any update u with $\mathbb{E}[\|u\|^2 \mid \theta_t] < \infty$ and $\theta_{t+1} = \theta_t - \eta_t u$,*

$$\mathbb{E}\left[\mathcal{L}_\star(\theta_{t+1}) \mid \theta_t\right] \le \mathcal{L}_\star(\theta_t) - \frac{\eta_t}{2}\|g_\star\|^2 + \frac{\eta_t}{2}\,\mathrm{MSE}(u).$$

Proof. By Assumption 1 and linearity of expectation,

$$\mathbb{E}[\mathcal{L}_\star(\theta_{t+1}) \mid \theta_t] \le \mathcal{L}_\star(\theta_t) - \eta_t \langle g_\star, \mathbb{E}[u \mid \theta_t] \rangle + \frac{\beta \eta_t^2}{2} \mathbb{E}[\|u\|^2 \mid \theta_t].$$

Expanding $\mathrm{MSE}(u) = \mathbb{E}[\|u\|^2 \mid \theta_t] - 2\langle g_\star, \mathbb{E}[u \mid \theta_t] \rangle + \|g_\star\|^2$ and rearranging gives

$$-\langle g_\star, \mathbb{E}[u \mid \theta_t] \rangle + \frac{1}{2}\mathbb{E}[\|u\|^2 \mid \theta_t] = -\frac{1}{2}\|g_\star\|^2 + \frac{1}{2}\,\mathrm{MSE}(u).$$

Substituting and using $\eta_t \le 1/\beta$ to drop the nonpositive remainder $(\frac{\beta \eta_t^2}{2} - \frac{\eta_t}{2})\mathbb{E}[\|u\|^2 \mid \theta_t] \le 0$ yields the result. □

Given the projected update u_t of Eq. (2), write $\mathrm{MSE}(u_t) = \mathcal{B}(U_t) + \mathcal{E}(u_t)$, where $\mathcal{B}(U_t) \coloneqq \mathbb{E}[\inf_{u \in U_t} \|u - g_\star\|^2 \mid \theta_t]$ is the bias of Section 2.3 and $\mathcal{E}(u_t) \coloneqq \mathrm{MSE}(u_t) - \mathcal{B}(U_t)$ is the excess error of selecting within U_t with the noisy estimate $\hat{g}_\star$. Define $\Sigma_{\mathrm{tr}} \coloneqq \mathrm{Cov}(g_i \mid \theta_t)$ and $\Sigma_\star \coloneqq \mathrm{Cov}(g_j^\star \mid \theta_t)$. For the two endpoints the decomposition is exact.

Proposition C.3 (Bias and excess error at the endpoints)**.** *Fix θ_t and let $\eta_t = 1/\beta$. Then:*

(i) ***Full-Training Update.*** *For $U_t^{\mathrm{tr}} = \{\hat{g}_{\mathrm{tr}}\}$,*

$$\mathcal{B}(U_t^{\mathrm{tr}}) = \|g_{\mathrm{tr}} - g_\star\|^2 + \frac{\mathrm{tr}(\Sigma_{\mathrm{tr}})}{n}, \qquad \mathcal{E}(u_t^{\mathrm{tr}}) = 0.$$

(ii) ***Target-Only Update.*** *For $U_t^\star = \mathbb{R}^d$,*

$$\mathcal{B}(U_t^\star) = 0, \qquad \mathcal{E}(u_t^\star) = \frac{\mathrm{tr}(\Sigma_\star)}{m}.$$

Proof. We prove this one by one.

(i) *Part (i) (Full-Training Update).* Since $u_t^{\mathrm{tr}} = \hat{g}_{\mathrm{tr}}$ has mean g_{tr} and covariance Σ_{tr}/n,

$$\mathrm{MSE}(u_t^{\mathrm{tr}}) = \|\mathbb{E}[\hat{g}_{\mathrm{tr}} - g_\star \mid \theta_t]\|^2 + \mathrm{tr}(\mathrm{Cov}(\hat{g}_{\mathrm{tr}} - g_\star \mid \theta_t)) = \|g_{\mathrm{tr}} - g_\star\|^2 + \frac{1}{n}\,\mathrm{tr}(\Sigma_{\mathrm{tr}}).$$

Furthermore, note that since $U_t^{\mathrm{tr}} = \{\hat{g}_{\mathrm{tr}}\}$, we also have $\mathcal{B}(U_t^{\mathrm{tr}}) = \mathrm{MSE}(u_t^{\mathrm{tr}})$.

(ii) *Part (ii) (Target-Only Update).* With $u_t^\star = \hat{g}_\star$ and $\hat{g}_\star - g_\star$ has mean zero and covariance $\Sigma_\star/m$,

$$\mathrm{MSE}(u_t^\star) = \mathbb{E}[\|\hat{g}_\star - g_\star\|^2 \mid \theta_t] = \mathrm{tr}(\mathrm{Cov}(\hat{g}_\star \mid \theta_t)) = \frac{1}{m}\,\mathrm{tr}(\Sigma_\star).$$

We conclude the proof by noting that $\mathcal{B}(U_t^\star) = 0$ as $U_t^\star = \mathbb{R}^d$.

□

Remark C.4 (Bias includes the fluctuation of the feasible set)**.** *$\mathcal{B}(U_t)$ is the expected squared distance from $g_\star$ to the random feasible set, so it includes the sampling fluctuation of U_t itself: for Full-Training Update it contains the training-batch term $\mathrm{tr}(\Sigma_{\mathrm{tr}})/n$, which a classical bias–variance decomposition of the estimator $\hat{g}_{\mathrm{tr}}$ would call variance. We count it as bias because it persists with a noiseless target ($m \to \infty$), whereas the excess error $\mathcal{E}(u_t)$, the additional error of selecting within U_t with $\hat{g}_\star$ rather than with $g_\star$, vanishes in that limit (Theorem C.6).*

We next state and prove the noisy projection oracle inequality, with the update-difference subspace $\mathcal{D}(U_t)$ and the exposed variance $\mathcal{V}(U_t)$ as defined in Section 2.3. Throughout, condition on θ_t; the training batch B_t determines U_t and hence $\mathcal{D}(U_t)$, and the target batch $B_t^\star$ is independent of B_t with $\mathbb{E}[\xi \mid B_t, \theta_t] = 0$ and $\mathrm{Cov}(\xi \mid B_t, \theta_t) = \Sigma_\star/m$. The key observation is that the projection is blind to noise outside the update-difference subspace.

Lemma C.5. *For any nonempty closed $U \subseteq \mathbb{R}^d$ and any $g, \xi \in \mathbb{R}^d$,*

$$\arg\min_{u \in U} \|u - g - \xi\|^2 = \arg\min_{u \in U} \|u - g - \mathrm{Proj}_{\mathcal{D}(U)}(\xi)\|^2.$$

Proof. For $u, v \in U$, $\|u - g - \xi\|^2 - \|v - g - \xi\|^2 = \|u - g\|^2 - \|v - g\|^2 - 2\langle u - v, \xi\rangle$, and $\langle u - v, \xi\rangle = \langle u - v, \mathrm{Proj}_{\mathcal{D}(U)}(\xi)\rangle$ because $u - v \in \mathcal{D}(U)$. Every pairwise comparison of the two objectives is therefore unchanged when ξ is replaced by $\mathrm{Proj}_{\mathcal{D}(U)}(\xi)$, so the two minimizer sets coincide. □

Theorem C.6 (Noisy projection oracle inequality)**.** *Fix θ_t, and let U_t be any nonempty closed feasible set determined by B_t. The update u_t of Eq.* (2) *satisfies*

$$\sqrt{\mathrm{MSE}(u_t)} \le \sqrt{\mathcal{B}(U_t)} + 2\sqrt{\mathcal{V}(U_t)}, \qquad \textit{equivalently} \qquad \mathcal{E}(u_t) \le 4\sqrt{\mathcal{B}(U_t)\,\mathcal{V}(U_t)} + 4\mathcal{V}(U_t),$$

where $\mathcal{V}(U_t) = \mathbb{E}[\mathrm{tr}(\mathrm{Proj}_{\mathcal{D}(U_t)}\,\Sigma_\star) \mid \theta_t]/m$. Moreover:

- *(i)* ***Orderings.*** *If $U_t \subseteq U_t'$ are two feasible sets determined by B_t, then $\mathcal{B}(U_t') \le \mathcal{B}(U_t)$ and $\mathcal{V}(U_t) \le \mathcal{V}(U_t')$. In particular, for any $k \in [n]$ and partition $\mathcal{G}$, $\mathcal{B}(U_t^{\mathrm{grp}}(k;\mathcal{G})) \le \mathcal{B}(U_t^{\mathrm{glob}}(k))$ and $\mathcal{V}(U_t^{\mathrm{glob}}(k)) \le \mathcal{V}(U_t^{\mathrm{grp}}(k;\mathcal{G}))$, while $\mathcal{V}(U_t^{\mathrm{tr}}) = 0$ and $\mathcal{V}(U_t^\star) = \mathrm{tr}(\Sigma_\star)/m$.*
- *(ii)* ***Exposed-variance bounds.*** *Under Assumption 2, for any $k \in [n]$ and partition $\mathcal{G}$ with P groups, $\mathcal{V}(U_t^{\mathrm{glob}}(k)) \le (n-1)\sigma^2/m$ and $\mathcal{V}(U_t^{\mathrm{grp}}(k;\mathcal{G})) \le \sigma^2 \sum_{p=1}^P \min\{|G_p|, n-1\}/m \le P(n-1)\sigma^2/m$.*

Proof. Fix B_t (hence U_t) and ξ, and write $\mathrm{Proj} := \mathrm{Proj}_{\mathcal{D}(U_t)}$ and $\zeta := \mathrm{Proj}(\xi)$. Let $u^\dagger \in \arg\min_{u \in U_t}\|u - g_\star\|^2$ be the oracle choice. By Theorem C.5, $u_t \in \arg\min_{u \in U_t}\|u - g_\star - \zeta\|^2$, so $\|u_t - g_\star - \zeta\| \le \|u^\dagger - g_\star - \zeta\|$ because $u^\dagger \in U_t$ is a competitor; combined with the triangle inequality on both sides,

$$\|u_t - g_\star\| \le \|u_t - g_\star - \zeta\| + \|\zeta\| \le \|u^\dagger - g_\star - \zeta\| + \|\zeta\| \le \|u^\dagger - g_\star\| + 2\|\zeta\|.$$

This holds for every realization of $(B_t, B_t^\star)$. For a random vector X, write $\|X\|_{L^2} := \sqrt{\mathbb{E}[\|X\|^2 \mid \theta_t]}$ for its root-mean-square size over both batches. This map is monotone ($0 \le A \le B$ pointwise

implies $\|A\|_{L^2} \le \|B\|_{L^2}$) and satisfies the triangle inequality (Minkowski's inequality), so applying it to both sides of the display gives

$$\sqrt{\mathrm{MSE}(u_t)} = \|u_t - g_\star\|_{L^2} \le \|u^\dagger - g_\star\|_{L^2} + 2\|\zeta\|_{L^2} = \sqrt{\mathcal{B}(U_t)} + 2\sqrt{\mathcal{V}(U_t)},$$

where $\mathbb{E}[\|u^\dagger - g_\star\|^2 \mid \theta_t] = \mathbb{E}[\inf_{u\in U_t}\|u - g_\star\|^2 \mid \theta_t] = \mathcal{B}(U_t)$ because $u^\dagger$ attains the infimum, and $\mathbb{E}[\|\zeta\|^2 \mid \theta_t] = \mathcal{V}(U_t)$ by definition. Squaring both (nonnegative) sides expands the right-hand side to $\mathcal{B}(U_t) + 4\sqrt{\mathcal{B}(U_t)\mathcal{V}(U_t)} + 4\mathcal{V}(U_t)$, and subtracting $\mathcal{B}(U_t)$ gives $\mathcal{E}(u_t) = \mathrm{MSE}(u_t) - \mathcal{B}(U_t) \le 4\sqrt{\mathcal{B}(U_t)\mathcal{V}(U_t)} + 4\mathcal{V}(U_t)$; adding $\mathcal{B}(U_t)$ back and taking square roots returns the first form, so the two are equivalent. For the covariance form of $\mathcal{V}(U_t)$, condition on B_t: $\mathcal{D}(U_t)$ is then a fixed subspace, so Proj is a fixed matrix with $\mathrm{Proj}^\top = \mathrm{Proj}$ and $\mathrm{Proj}^2 = \mathrm{Proj}$, while ξ has conditional mean zero and conditional covariance $\Sigma_\star/m$ because the target batch is independent of B_t. Hence

$$\begin{aligned}\mathbb{E}[\|\mathrm{Proj}(\xi)\|^2 \mid B_t, \theta_t] &= \mathbb{E}[\mathrm{tr}(\mathrm{Proj}\,\xi\xi^\top\,\mathrm{Proj}) \mid B_t, \theta_t] = \mathrm{tr}(\mathrm{Proj}\;\mathbb{E}[\xi\xi^\top \mid B_t, \theta_t]\;\mathrm{Proj}) \\ &= \mathrm{tr}(\mathrm{Proj}\,\Sigma_\star\,\mathrm{Proj})/m = \mathrm{tr}(\mathrm{Proj}\,\Sigma_\star)/m,\end{aligned}$$

using $\|v\|^2 = \mathrm{tr}(vv^\top)$, linearity of the trace and of the expectation, and cyclicity of the trace together with $\mathrm{Proj}^2 = \mathrm{Proj}$; taking the expectation over B_t yields $\mathcal{V}(U_t) = \mathbb{E}[\mathrm{tr}(\mathrm{Proj}_{\mathcal{D}(U_t)}\,\Sigma_\star) \mid \theta_t]/m$.

Part (i). For $U_t \subseteq U_t'$, the bias ordering holds because the infimum over a larger set is smaller, and $\mathcal{D}(U_t) \subseteq \mathcal{D}(U_t')$, so the difference of the two orthogonal projectors is itself an orthogonal projector, hence positive semidefinite, and $\mathrm{tr}\big((\mathrm{Proj}_{\mathcal{D}(U_t')} - \mathrm{Proj}_{\mathcal{D}(U_t)})\Sigma_\star\big) \ge 0$ as the trace of a product of two positive semidefinite matrices. The global and group-wise sets are nested by Section 2.2.2. For the singleton U_t^{tr} we have $\mathcal{D}(U_t^{\mathrm{tr}}) = \{0\}$, and for $U_t^\star = \mathbb{R}^d$ we have $\mathrm{Proj} = I$.

Part (ii). Conditional on B_t, let $v_1, \dots, v_{\dim \mathcal{D}(U_t)}$ be an orthonormal basis of $\mathcal{D}(U_t)$. By Assumption 2 each $\langle \xi, v_j\rangle$ is sub-Gaussian with variance proxy σ^2/m, and a sub-Gaussian variable has second moment at most its variance proxy, so $\mathbb{E}[\|\mathrm{Proj}(\xi)\|^2 \mid B_t, \theta_t] = \sum_{j=1}^{\dim \mathcal{D}(U_t)} \mathbb{E}[\langle \xi, v_j\rangle^2 \mid B_t, \theta_t] \le \dim \mathcal{D}(U_t) \cdot \sigma^2/m$. It remains to bound $\dim \mathcal{D}(U_t)$. Any two elements of $U_t^{\mathrm{glob}}(k)$ differ by $g_S - g_{S'} = \sum_{i=1}^n c_i g_i$ with $\sum_i c_i = 0$, so $\mathcal{D}(U_t^{\mathrm{glob}}(k)) \subseteq \mathrm{span}\{g_i - g_1 \colon 2 \le i \le n\}$, whose dimension is at most $n-1$. For the product set, $U_t^{\mathrm{grp}}(k;\mathcal{G}) - U_t^{\mathrm{grp}}(k;\mathcal{G}) = \prod_p \big(U_t^{(p)}(k) - U_t^{(p)}(k)\big)$, so $\mathcal{D}(U_t^{\mathrm{grp}}(k;\mathcal{G})) = \bigoplus_{p=1}^P \mathcal{D}(U_t^{(p)}(k))$ with each summand supported on the coordinates G_p, and the same argument bounds its dimension by $\min\{|G_p|, n-1\}$. □

Remark C.7 (Non-convex feasible sets)**.** *The proof of Theorem C.6 compares u_t with the oracle choice $u^\dagger$ only through the minimality of u_t, so it applies to the discrete, non-convex feasible sets of Section 2.2. For these sets the projection need not be unique and is not 1-Lipschitz in its input, since an arbitrarily small change of $\hat{g}_\star$ can switch the selected subset; the standard argument through the non-expansiveness of projections onto convex sets is therefore not available.*

Bias ordering. The bias inequality $\mathcal{B}(U_t^{\mathrm{grp}}(k;\mathcal{G})) \le \mathcal{B}(U_t^{\mathrm{glob}}(k))$ follows directly from $U_t^{\mathrm{glob}}(k) \subseteq U_t^{\mathrm{grp}}(k;\mathcal{G})$. More generally, for two partitions $\mathcal{G}, \mathcal{G}'$ where $\mathcal{G}'$ refines $\mathcal{G}$, we have $U_t^{\mathrm{grp}}(k;\mathcal{G}) \subseteq U_t^{\mathrm{grp}}(k;\mathcal{G}')$, and hence $\mathcal{B}(U_t^{\mathrm{grp}}(k;\mathcal{G}')) \le \mathcal{B}(U_t^{\mathrm{grp}}(k;\mathcal{G}))$. For different subset sizes $k \ne k'$, no inclusion relation generally holds between $U_t^{\mathrm{glob}}(k)$ and $U_t^{\mathrm{glob}}(k')$ (or between $U_t^{\mathrm{grp}}(k;\mathcal{G})$ and $U_t^{\mathrm{grp}}(k';\mathcal{G})$), so bias is generally not monotonic in k. Intuitively, however, larger feasible sets give lower bias: when $k = n$, $U_t^{\mathrm{tr}} = U_t^{\mathrm{glob}}(n) = U_t^{\mathrm{grp}}(n;\mathcal{G})$ degenerates to a singleton; when $k = n/2$, $U_t^{\mathrm{glob}}(k)$ admits $\binom{n}{n/2}$ candidates for approximating $g_\star$, which boosts the chance of a good approximation and hence yields a lower bias.

C.4.1 Rates in the target batch size

Squaring the oracle inequality of Theorem C.6 bounds the excess error by

$$\mathcal{E}(u_t) = \mathrm{MSE}(u_t) - \mathcal{B}(U_t) \le 4\sqrt{\mathcal{B}(U_t)\,\mathcal{V}(U_t)} + 4\mathcal{V}(U_t).$$

The two quantities on the right scale differently with m. Under Assumption 2, the proof of Theorem C.6(ii) gives $\mathcal{V}(U_t) \le d_t\sigma^2/m$ with $d_t := \mathbb{E}[\dim \mathcal{D}(U_t) \mid \theta_t] \le \sum_p \min\{|G_p|, n-1\}$, so

$\mathcal{V}(U_t) = O(1/m)$; this bound depends on the feasible set only through the dimension of its update-difference subspace, not on k or on the cardinality $\binom{n}{k}$, and it vanishes at $k = n$, where the feasible set is a singleton. The bias $\mathcal{B}(U_t)$, in contrast, does not depend on m at all: it is determined by the training batch and the training–target mismatch. In the realizable case $\mathcal{B}(U_t) = 0$ the cross term vanishes and $\mathcal{E}(u_t) \le 4\mathcal{V}(U_t) = O(1/m)$, the rate of Target-Only Update in Theorem C.3. When $\mathcal{B}(U_t) > 0$, the cross term $4\sqrt{\mathcal{B}(U_t)\mathcal{V}(U_t)}$ is $O(1/\sqrt{m})$ whereas $4\mathcal{V}(U_t)$ is $O(1/m)$, so the cross term is the larger of the two as soon as $\mathcal{V}(U_t) < \mathcal{B}(U_t)$, i.e., once m exceeds about $d_t\sigma^2/\mathcal{B}(U_t)$, and the bound on the excess error decays only as $1/\sqrt{m}$. This slower rate is a property of the bound rather than of the excess error itself: when the candidates in U_t are separated from the oracle choice by a fixed margin, a mis-selection requires the noise to overturn that margin, an event of probability exponentially small in m; the $1/\sqrt{m}$ rate is the worst case over near-ties, which is the relevant regime when many per-sample gradients are similar.

C.4.2 Two bounds on the excess error

Figure 4 illustrates the excess error: the update chosen with the noisy estimate $\hat{g}_\star$ may differ from the oracle choice within U_t, and a larger feasible set gives the update more freedom to align with $g_\star$ but also with the noise ξ. The classical finite-class argument controls this freedom through the cardinality of the feasible set under a gradient-norm bound (Assumption 5) (Shalev-Shwartz & Ben-David, 2014; Hazan, 2016; Lan, 2020; Liu et al., 2023), whereas Theorem C.6 uses only the noise components inside the update-difference subspace, whose energy is bounded by the dimension of that subspace. We state the classical bound, compare the two, and then show that the oracle inequality can be refined to pay, group by group, the smaller of the two complexity measures.

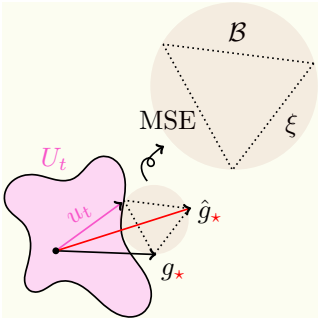

Figure 4: Geometry of the one-step error for a chosen U_t: the bias is the distance from $g_\star$ to U_t; the noise ξ moves the projection to u_t, and the gap between MSE and $\mathcal{B}$ is the excess error.

Theorem C.8 (Finite-class excess-error bounds)**.** *Assume Assumptions 1, 2 and 5 and fix θ_t and let $\eta_t = 1/\beta$. Then for any $k \in [n]$:*

(i) ***Global Subset Update.*** *For $U_t^{\mathrm{glob}}(k)$,*

$$\mathcal{E}(u_t^{\mathrm{glob}}(k)) \le \frac{4C\sigma}{\sqrt{m}}\sqrt{2\log\left(2\binom{n}{k}\right)}.$$

(ii) ***Group-Wise Subset Update.*** *For $U_t^{\mathrm{grp}}(k;\mathcal{G})$ where $\mathcal{G} = \{G_p\}_{p=1}^P$ contains P groups,*

$$\mathcal{E}(u_t^{\mathrm{grp}}(k;\mathcal{G})) \le \frac{4CP\sigma}{\sqrt{m}}\sqrt{2\log\left(2\binom{n}{k}\right)}.$$

Proof. We prove this one by one.

(i) *Part (i) (Global Subset Update).* Fix k and write $U := U_t^{\mathrm{glob}}(k)$. Let $\xi := \hat{g}_\star - g_\star$. Define

$$u^* := u_t^{\mathrm{glob}} \in \arg\min_{u\in U}\|u - \hat{g}_\star\|^2, \qquad u^\dagger \in \arg\min_{u\in U}\|u - g_\star\|^2.$$

By the projection property, $\|u^* - \hat{g}_\star\|^2 \le \|u^\dagger - \hat{g}_\star\|^2$. Expanding both sides via $\hat{g}_\star = g_\star + \xi$:

$$\|u^* - g_\star\|^2 - 2\langle u^* - g_\star, \xi\rangle \le \|u^\dagger - g_\star\|^2 - 2\langle u^\dagger - g_\star, \xi\rangle.$$

Rearranging, we get

$$\|u^* - g_\star\|^2 \le \|u^\dagger - g_\star\|^2 + 2\langle u^* - u^\dagger, \xi\rangle \le \|u^\dagger - g_\star\|^2 + 4 \sup_{u\in U} |\langle \xi, u\rangle|.$$

Taking expectation over B_t and $B_t^\star$ (recall that MSE is the expectation over both), we have

$$\mathrm{MSE}(u_t^{\mathrm{glob}}) \le \mathbb{E}[\|u^\dagger - g_\star\|^2 \mid \theta_t] + 4\mathbb{E}\left[\sup_{u\in U} |\langle \xi, u\rangle| \middle| \theta_t\right],$$

The first term is simply $\mathcal{B}(U) := \mathbb{E}[\inf_{u\in U}\|u - g_\star\|^2 \mid \theta_t]$. Hence, we have

$$\mathcal{E}(u_t^{\mathrm{glob}}) = \mathrm{MSE}(u_t^{\mathrm{glob}}) - \mathcal{B}(U) \le 4\mathbb{E}\left[\sup_{u\in U} |\langle \xi, u\rangle| \middle| \theta_t\right],$$

and it remains to bound $\mathbb{E}[\sup_{u\in U}|\langle \xi, u\rangle| \mid \theta_t]$. Condition on B_t, so that U is a fixed finite set independent of ξ. By the sub-Gaussian assumption on the target noise, for each fixed u the random variable $\langle \xi, u\rangle$ is sub-Gaussian with variance proxy $\sigma^2\|u\|^2/m$, i.e.,

$$\mathbb{E}\left[\exp(\lambda\langle \xi, u\rangle) \middle| \theta_t\right] \le \exp\left(\frac{\lambda^2\sigma^2}{2m}\|u\|^2\right), \quad \forall \lambda \in \mathbb{R}.$$

By Assumption 5 and the triangle inequality (every $u \in U$ is an average of k per-sample gradients), $\|u\| \le C$ for all $u \in U$. The sub-Gaussian maximal inequality (Vershynin, 2018) states that for $|U|$ mean-zero sub-Gaussian random variables $X_1, \dots, X_{|U|}$ with common variance proxy ν^2, $\mathbb{E}[\max_i |X_i|] \le \nu\sqrt{2\log(2|U|)}$. Applying this conditionally on B_t with $|U| \le \binom{n}{k}$ and $\nu^2 = \sigma^2 C^2/m$:

$$\mathbb{E}\left[\sup_{u\in U} |\langle \xi, u\rangle| \middle| B_t, \theta_t\right] \le \frac{\sigma C}{\sqrt{m}}\sqrt{2\log(2|U|)}.$$

Taking expectation over B_t finishes the proof.

(ii) *Part (ii) (Group-Wise Subset Update).* The argument follows Part (i). Write $U := U_t^{\mathrm{grp}}(k; \mathcal{G})$. By Assumption 5, each per-group component satisfies $\|u^{(p)}\| \le C$, so $\|u\|^2 \le PC^2$. Hence, the same argument yields

$$\mathcal{E}(u_t^{\mathrm{grp}}) \le \frac{4C\sqrt{P}\sigma}{\sqrt{m}}\sqrt{2\log(2|U|)}.$$

The cardinality bound follows from the product structure of U: each group p chooses its own $S_{t,p} \subset [n]$ with $|S_{t,p}| = k$, so $|U| \le \binom{n}{k}^P$. Substituting: $\log(2\binom{n}{k}^P) = \log 2 + P\log\binom{n}{k} \le P\log(2\binom{n}{k})$ for $P \ge 1$, so $C\sqrt{P}\cdot\sqrt{2P\log(2\binom{n}{k})} = CP\sqrt{2\log(2\binom{n}{k})}$.

□

Comparison. Theorems C.6 and C.8 bound the same excess error $\mathcal{E}(u_t)$ through two different complexity measures of the feasible set, the dimension of its update-difference subspace and the logarithm of its cardinality, and neither bound dominates the other. The finite-class bound has no cross term, but it needs the gradient-norm bound C and decays as $1/\sqrt{m}$ even when $\mathcal{B}(U_t) = 0$. The oracle inequality needs no gradient-norm bound and decays as $1/m$ in that case, but it carries the cross term $4\sqrt{\mathcal{B}(U_t)\mathcal{V}(U_t)}$; by Young's inequality the cross term can be traded for an inflated bias coefficient, $\mathrm{MSE}(u_t) \le (1+\epsilon)\mathcal{B}(U_t) + 4(1+\epsilon^{-1})\mathcal{V}(U_t)$ for any $\epsilon > 0$, which is an additive bound of the same form as Theorem C.8. The two complexity measures are not ordered either: per group, the dimension is at most $n-1$ while the log-cardinality term of Theorem C.9 below is $2\log\binom{n}{k} + 2$, so the dimension is the smaller measure for our $n = 8$, $k = 4$ (7 versus about 10.5), whereas the log-cardinality is smaller for large candidate batches with small subsets.

Unification. The oracle-inequality argument can be refined so that every group pays the smaller of the two measures.

Proposition C.9 (Unified excess-error bound)**.** *Under Assumption 2, for $U_t = U_t^{\mathrm{grp}}(k;\mathcal{G})$ with P groups ($P = 1$ for Global Subset Update),*

$$\sqrt{\mathrm{MSE}(u_t)} \le \sqrt{\mathcal{B}(U_t)} + 2\sqrt{\frac{\sigma^2}{m}\sum_{p=1}^{P} \min\Big\{\mathbb{E}[\dim \mathcal{D}(U_t^{(p)}(k)) \mid \theta_t],\, 2\log\tbinom{n}{k} + 2\Big\}}.$$

Proof. For each group p, let v_p be the oracle choice in $U_t^{(p)}(k)$, and let Z_p be the largest normalized noise component along a candidate difference,

$$Z_p \coloneqq \max\left\{0,\ \max_{w \in U_t^{(p)}(k)\setminus\{v_p\}} \frac{\langle \xi^{(p)}, w - v_p\rangle}{\|w - v_p\|}\right\}.$$

Per-group nearest-point optimality of $u_t^{(p)}$ against v_p, expanded as in the proof of Theorem C.8, gives

$$\begin{aligned}\|u_t^{(p)} - g_\star^{(p)}\|^2 - \|v_p - g_\star^{(p)}\|^2 &\le 2\langle u_t^{(p)} - v_p, \xi^{(p)}\rangle \\ &\le 2Z_p\|u_t^{(p)} - v_p\| \\ &\le 2Z_p\big(\|u_t^{(p)} - g_\star^{(p)}\| + \|v_p - g_\star^{(p)}\|\big),\end{aligned}$$

and dividing by $\|u_t^{(p)} - g_\star^{(p)}\| + \|v_p - g_\star^{(p)}\|$ (the case $u_t^{(p)} = v_p$ being trivial) yields

$$\|u_t^{(p)} - g_\star^{(p)}\| \le \|v_p - g_\star^{(p)}\| + 2Z_p.$$

Summing the squares over groups, Minkowski's inequality in ℓ_2 and then the L^2 step from the proof of Theorem C.6 give

$$\sqrt{\mathrm{MSE}(u_t)} \le \sqrt{\mathcal{B}(U_t)} + 2\sqrt{\mathbb{E}\left[\sum_{p=1}^{P} Z_p^2 \,\middle|\, \theta_t\right]},$$

so it remains to bound $\mathbb{E}[Z_p^2 \mid B_t, \theta_t]$, which we do in two ways. First, since $w - v_p \in \mathcal{D}(U_t^{(p)}(k))$, the Cauchy–Schwarz inequality gives $Z_p \le \|\mathrm{Proj}_{\mathcal{D}(U_t^{(p)}(k))}(\xi^{(p)})\|$, and as in Part (ii) of Theorem C.6,

$$\mathbb{E}[Z_p^2 \mid B_t, \theta_t] \le \dim \mathcal{D}(U_t^{(p)}(k))\,\frac{\sigma^2}{m}.$$

Second, each $\langle \xi^{(p)}, w - v_p\rangle/\|w - v_p\|$ is sub-Gaussian with variance proxy σ^2/m by Assumption 2, so a union bound over the $N_p \coloneqq |U_t^{(p)}(k)| \le \binom{n}{k}$ candidates gives $\Pr(Z_p \ge z \mid B_t, \theta_t) \le N_p \exp(-mz^2/(2\sigma^2))$ for $z > 0$, and integrating the tail,

$$\begin{aligned}\mathbb{E}[Z_p^2 \mid B_t, \theta_t] &= \int_0^\infty \Pr(Z_p^2 \ge s \mid B_t, \theta_t)\,\mathrm{d}s \le \int_0^\infty \min\big\{1, N_p e^{-ms/(2\sigma^2)}\big\}\,\mathrm{d}s \\ &= \frac{\sigma^2}{m}\big(2\log N_p + 2\big).\end{aligned}$$

Taking the smaller of the two bounds for each group and then the expectation over B_t completes the proof. □

Theorem C.9 recovers the $\log\binom{n}{k}$ dependence of Theorem C.8 whenever it is the smaller measure. The exposed-variance form of Section 2.3 therefore subsumes the log-cardinality dependence of the classical analysis, while retaining the $1/m$ rate when $\mathcal{B}(U_t) = 0$ and requiring no gradient-norm bound.

C.4.3 REGIMES

Which update dominates depends on the target batch size, the training–target mismatch, and the feasible-set complexity. Figure 5 sketches three representative regimes with synthetic parameter choices for the bounds of Theorems C.3, C.6 and C.8:

- **Small mismatch.** All four methods occupy distinct regimes. As m grows, the preferred update transitions from the Full-Training Update to the Global Subset Update, then to the Group-Wise Subset Update, and finally to the Target-Only Update.
- **Moderate mismatch.** The Group-Wise Subset Update dominates the Global Subset Update across a wider range of m, yielding a transition from the Full-Training Update to the Group-Wise Subset Update and then to the Target-Only Update.
- **Large mismatch.** The bias of full-training and Global Subset Updates is large enough that the Group-Wise Subset Update dominates even for relatively small m, before eventually giving way to the Target-Only Update when enough target data are available.

The pools of Section 4.1.2 span these regimes: the in-distribution `Dolci` instruction and reasoning pools, the mixed pool, and `Tulu 3`, in which the target skill is most diluted.

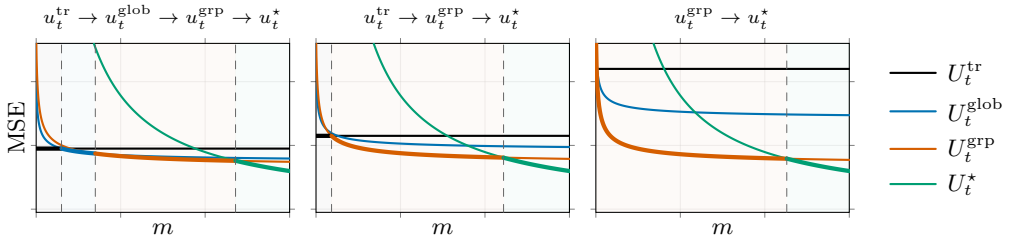

Figure 5: Illustration of the MSE bounds of Theorems C.3, C.6 and C.8 as a function of the target batch size m: Full-Training is constant, Global and Group-Wise Subset Updates decay as $1/\sqrt{m}$ (Group-Wise with the larger excess-error constant but the smaller bias), and Target-Only decays as $1/m$. The excess-error constants are shared across panels; only the bias terms grow with the distribution mismatch (left to right). Shaded regions indicate the method with the smallest MSE; as the mismatch increases, Group-Wise Subset Update dominates over a wider range of m.

C.4.4 CONNECTION TO CLASSICAL REGULARIZATION

Both data regularization and classical regularization (e.g., weight decay, KL penalties, LoRA's low-rank constraint) ultimately constrain the parameter update, but they differ in the *source* of the constraint. Classical regularization is usually derived from a *data-agnostic* complexity measure on θ, restricting the update through intrinsic properties of the parameters themselves. Data regularization, by contrast, imposes a *data-induced* constraint in the parameter-update space via the feasible set U_t, whose admissible directions are shaped by the available training samples.

C.5 TRAJECTORY ANALYSIS

This section states and proves the trajectory results summarized in Theorem 2.2. The arguments are standard descent arguments (Ghadimi & Lan, 2013; Karimi et al., 2016; Ajalloeian & Stich, 2020); what is specific to our setting is that the required one-step control is supplied by the projection geometry through Theorem C.6.

Setting. Let $\mathcal{F}_t$ be the σ-algebra generated by the batches $(B_s, B_s^\star)_{s<t}$, with θ_0 deterministic, so that θ_t is $\mathcal{F}_t$-measurable and $(B_t, B_t^\star)$ is independent of $\mathcal{F}_t$. Write $\mathbb{E}_t[\cdot] := \mathbb{E}[\cdot \mid \mathcal{F}_t]$ and $g_{\star,t} := g_\star(\theta_t)$. Because the batches at step t are fresh, every conditional quantity of Section 2.3, written there as conditional on θ_t, coincides with its $\mathcal{F}_t$-conditional version: $\mathrm{MSE}(u_t) = \mathbb{E}_t[\|u_t - g_{\star,t}\|^2] = \mathcal{B}(U_t) + \mathcal{E}(u_t)$, and Theorems C.2 and C.6 hold with $\mathbb{E}_t$ in place of $\mathbb{E}[\cdot \mid \theta_t]$. Step sizes are deterministic, all second moments are finite, and the projection is selected measurably with a fixed tie-breaking rule. We write $\Delta_\star := \mathcal{L}_\star(\theta_0) - \mathcal{L}_\star^{\inf}$, $\Lambda_T := \sum_{t<T} \eta_t$, and $D_t := \mathbb{E}[\mathcal{L}_\star(\theta_t)] - \mathcal{L}_\star^{\inf}$.

Theorem C.10 (Trajectory bound)**.** *Under Assumptions 1 and 3 with $q < 1$ and deterministic step sizes $0 < \eta_t \le 1/\beta$, for every $T \ge 1$,*

$$\frac{\sum_{t<T} \eta_t \, \mathbb{E}[\|g_{\star,t}\|^2]}{\Lambda_T} \le \frac{4\Delta_\star}{(1-q^2)\Lambda_T} + \frac{2(1+q^2)}{(1-q^2)^2} \cdot \frac{\sum_{t<T} \eta_t (b_t + 2\varsigma_t)^2}{\Lambda_T}.$$

Corollary C.11 (Exact and whole-sequence convergence)**.** *Under Assumptions 1 and 3 with $b_t = \varsigma_t = 0$, $q < 1$, and a constant step size $0 < \eta \le 1/\beta$, $\frac{1}{T}\sum_{t<T} \mathbb{E}[\|g_{\star,t}\|^2] \le 2\Delta_\star/(\eta(1-q^2)T)$ and $g_{\star,t} \to 0$ almost surely. If in addition Assumption 4 holds, then $D_T \le (1 - \mu\eta(1-q^2))^T \Delta_\star$, and θ_t converges almost surely, with finite path length $\sum_t \|\theta_{t+1} - \theta_t\| < \infty$, to a random θ_∞ with $\mathcal{L}_\star(\theta_\infty) = \mathcal{L}_\star^{\inf}$ (Theorem C.13).*

We first turn Assumption 3 into a relative bound on the MSE of the projected update.

Lemma C.12 (Relative MSE from coverage and exposed variance)**.** *Under Assumption 3, with q and ϕ_t as defined after it, almost surely $\sqrt{\mathrm{MSE}(u_t)} \le q\|g_{\star,t}\| + \phi_t$; in particular $\mathrm{MSE}(u_t) \le q^2\|g_{\star,t}\|^2$ when $\phi_t = 0$, and for $q < 1$*

$$\mathrm{MSE}(u_t) \le \frac{1+q^2}{2}\|g_{\star,t}\|^2 + \frac{1+q^2}{1-q^2}\phi_t^2.$$

Proof. Theorem C.6 (with $\mathbb{E}_t$) and Assumption 3 give $\sqrt{\mathrm{MSE}(u_t)} \le \sqrt{\rho\|g_{\star,t}\|^2 + b_t^2} + 2\sqrt{\omega\|g_{\star,t}\|^2 + \varsigma_t^2} \le q\|g_{\star,t}\| + \phi_t$ by $\sqrt{x+y} \le \sqrt{x} + \sqrt{y}$. If $q = 0$, the last display is immediate. Otherwise, squaring and applying Young's inequality $2q\|g_{\star,t}\|\phi_t \le \epsilon q^2\|g_{\star,t}\|^2 + \epsilon^{-1}\phi_t^2$ with $\epsilon = (1-q^2)/(2q^2)$ gives $(1+\epsilon)q^2 = (1+q^2)/2$ and $1 + \epsilon^{-1} = (1+q^2)/(1-q^2)$. □

Proof of Theorem C.10. Substituting Theorem C.12 into Theorem C.2 (with $\mathbb{E}_t$) gives, for $0 < \eta_t \le 1/\beta$,

$$\mathbb{E}_t[\mathcal{L}_\star(\theta_{t+1})] \le \mathcal{L}_\star(\theta_t) - \frac{\eta_t}{2} \cdot \frac{1-q^2}{2}\|g_{\star,t}\|^2 + \frac{\eta_t}{2} \cdot \frac{1+q^2}{1-q^2}\phi_t^2.$$

Taking expectations, summing over $t < T$, and using $\mathcal{L}_\star(\theta_T) \ge \mathcal{L}_\star^{\inf}$, $\frac{1-q^2}{4}\sum_{t<T} \eta_t \mathbb{E}[\|g_{\star,t}\|^2] \le \Delta_\star + \frac{1+q^2}{2(1-q^2)}\sum_{t<T} \eta_t \phi_t^2$; multiplying by $4/((1-q^2)\Lambda_T)$ yields the claim. □

Proof of Theorem C.11. With $\phi_t = 0$, Theorem C.12 gives $\mathrm{MSE}(u_t) \le q^2\|g_{\star,t}\|^2$, and Theorem C.2 gives $\mathbb{E}_t[\mathcal{L}_\star(\theta_{t+1})] \le \mathcal{L}_\star(\theta_t) - \frac{\eta(1-q^2)}{2}\|g_{\star,t}\|^2$. Telescoping yields $\sum_{t<T} \mathbb{E}[\|g_{\star,t}\|^2] \le 2\Delta_\star/(\eta(1-q^2))$ for every T, which is the stated rate; since the right-hand side does not depend on T, $\mathbb{E}[\sum_t \|g_{\star,t}\|^2] < \infty$, so by monotone convergence $\sum_t \|g_{\star,t}\|^2 < \infty$ and hence $g_{\star,t} \to 0$ almost surely. Under Assumption 4, substituting $\|g_{\star,t}\|^2 \ge 2\mu(\mathcal{L}_\star(\theta_t) - \mathcal{L}_\star^{\inf})$ gives $D_{t+1} \le \chi D_t$ with $\chi := 1 - \mu\eta(1-q^2)$, and $\chi \in [0,1)$ because $\eta \le 1/\beta$ and $\mu \le \beta$ (Assumptions 1 and 4 together imply $\mu \le \beta$), so $D_T \le \chi^T D_0$. Whole-sequence convergence is Theorem C.13 below. □

Theorem C.13 (Almost-sure finite-length convergence)**.** *Under the conditions of Theorem C.11 and Assumption 4, there is a random $\theta_\infty \in \mathbb{R}^d$ such that $\sum_{t\ge0}\|\theta_{t+1} - \theta_t\| < \infty$ and $\theta_t \to \theta_\infty$ almost surely, with $\mathcal{L}_\star(\theta_\infty) = \mathcal{L}_\star^{\inf}$ almost surely, and*

$$\mathbb{E}[\|\theta_T - \theta_\infty\|] \le \frac{\eta(1+q)\sqrt{2\beta\Delta_\star}}{1-\sqrt{\chi}}\,\chi^{T/2}, \qquad \chi = 1 - \mu\eta(1-q^2).$$

No uniqueness of the minimizer, isolatedness of stationary points, or convexity of $\mathcal{L}_\star$ or of U_t is required.

Proof. By the L^2 triangle inequality and Theorem C.12, $\sqrt{\mathbb{E}_t[\|u_t\|^2]} \le \|g_{\star,t}\| + \sqrt{\mathrm{MSE}(u_t)} \le (1+q)\|g_{\star,t}\|$. Smoothness and boundedness from below give the self-bound $\|g_\star(\theta)\|^2 \le 2\beta(\mathcal{L}_\star(\theta) - \mathcal{L}_\star^{\inf})$ (apply the descent inequality to the step $\theta - g_\star(\theta)/\beta$). Hence, by Jensen's inequality and the proof of Theorem C.11,

$$\mathbb{E}[\|\theta_{t+1} - \theta_t\|] = \eta\,\mathbb{E}[\|u_t\|] \le \eta(1+q)\sqrt{\mathbb{E}[\|g_{\star,t}\|^2]} \le \eta(1+q)\sqrt{2\beta D_t} \le \eta(1+q)\sqrt{2\beta\Delta_\star}\,\chi^{t/2}.$$

The right-hand side is summable, so $\mathbb{E}[\sum_t \|\theta_{t+1} - \theta_t\|] < \infty$ by Tonelli's theorem; thus the path length is finite almost surely, (θ_t) is almost surely Cauchy, and it converges to some θ_∞. Since $\mathcal{L}_\star(\theta_t) - \mathcal{L}_\star^{\inf} \geq 0$ converges to $\mathcal{L}_\star(\theta_\infty) - \mathcal{L}_\star^{\inf}$ by continuity while $D_t \to 0$, Fatou's lemma gives $\mathbb{E}[\mathcal{L}_\star(\theta_\infty) - \mathcal{L}_\star^{\inf}] = 0$. Summing the displayed estimate from T onward proves the tail bound (for $\chi = 0$ with the usual convention). □

Without Assumption 4, Theorem C.11 still gives $\sum_t \mathbb{E}[\|u_t\|^2] \leq (1+q)^2 \sum_t \mathbb{E}[\|g_{\star,t}\|^2] < \infty$, so the parameter jumps $\|\theta_{t+1} - \theta_t\| = \eta\|u_t\|$ vanish almost surely. If, in addition, the iterates lie almost surely in a compact set in which $\mathcal{L}_\star$ has finitely many stationary points, then θ_t converges almost surely to one of them, though not necessarily to a global minimizer: every cluster point of the bounded sequence (θ_t) is stationary by continuity, since $g_{\star,t} \to 0$, and a bounded sequence with vanishing jumps and finitely many cluster points converges, as otherwise it would cross between disjoint neighborhoods of two cluster points infinitely often and produce a cluster point outside them.

Instantiating the constants. Suppose the target-gradient covariance satisfies $\Sigma_\star \preceq (\sigma_0^2 + \sigma_1^2\|g_{\star,t}\|^2)I$ at every iterate, and $\dim \mathcal{D}(U_t) \leq d_{\text{eff}}$ almost surely (Theorem C.6(ii) gives $d_{\text{eff}} = \sum_p \min\{|G_p|, n-1\}$ for Group-Wise Subset Update and $d_{\text{eff}} = n - 1$ for Global Subset Update). Then $\mathcal{V}(U_t) = \mathbb{E}_t[\text{tr}(\text{Proj}_{\mathcal{D}(U_t)} \Sigma_\star)]/m \leq d_{\text{eff}}(\sigma_0^2 + \sigma_1^2\|g_{\star,t}\|^2)/m$, so the noise part of Assumption 3 holds with $\omega = d_{\text{eff}}\sigma_1^2/m$ and $\varsigma_t^2 = d_{\text{eff}}\sigma_0^2/m$; the floor b_t accounts for the training–target support mismatch and σ_0 for a target-noise floor. Neither is an implementation parameter. In the relative-noise regime $\sigma_0 = 0$, $b_t = 0$, the condition $q < 1$ of Theorem C.11 reads

$$m > \frac{4 d_{\text{eff}} \sigma_1^2}{(1 - \sqrt{\rho})^2},$$

a sufficient certification threshold (not a necessary sample-complexity bound) that makes the tradeoff of Section 2.3 quantitative: refining the partition improves the coverage ρ while enlarging d_{eff}, and hence the target batch size needed to certify the finer set. The zero-floor regime is genuinely stronger than bounded variance: it requires compatible training directions and vanishing exposed target noise near target-stationary points, in the spirit of the strong-growth and interpolation conditions of SGD theory (Vaswani et al., 2019), and it should not be asserted merely because a model is large.

Idealizations. The guarantees of this section concern the exact projected update of Eq.(2) taken as the SGD step. Our experiments pass u_t to AdamW (Theorem C.1) and select with the first-order top-k rule of Section 2.2.1, and set inclusion orders bias and exposed variance (Theorem C.6) but not trajectories pointwise. In our experiments the target batch is resampled at every step from a fixed finite target set (Section 4). Conditioning on that set, i.e., including it in $\mathcal{F}_0$ and replacing $\mathbb{P}_\star$ by its empirical distribution, leaves every result of this section intact for the empirical target loss, since an independent resample is conditionally unbiased for its gradient; increasing m then reduces the sampling noise relative to that empirical objective without adding population information. Transferring the guarantees to the population target loss additionally requires a uniform bound on the gap between the empirical and the population target gradients along the trajectory, which is a separate statistical question.

D Efficient realization of Dr. Post-Training under memory constraints

This section provides the full technical development of the system-level optimizations summarized in Section 3. We start by revisiting the memory footprint of standard training (Appendix D.1), which grounds the later discussions in three aspects. First, in Appendix D.2, we address the computational overhead of data-regularized updates with a carefully designed tensor lifetime schedule that can efficiently reuse the intermediate variables in standard forward and backward passes with negligible additional memory cost; we further extend the schedule to general partitions that may split parameters within a single layer, and provide a two-pass alternative when the memory budget cannot accommodate the one-pass schedule. Second, in Appendix D.3, we investigate the computational costs of the key per-sample scoring step incurred by the data-regularized updates, develop several implementations with different compute–memory trade-offs together with their exact complexity derivations, and identify the most efficient choice for different regimes. Third, in Appendix D.4, we show that the resulting design remains compatible with modern memory-saving training techniques,

including LoRA, MeSO, activation checkpointing, and gradient accumulation. Finally, Appendix D.5 discusses how the framework extends beyond linear layers, including an efficient scoring algorithm for embedding layers and the treatment of normalization layers.

Notation. In Appendices D.2 to D.4, we focus primarily on layer-aligned partitions $\mathcal{G} = \{G_p\}_{p=1}^{P}$ such that each G_p contains parameters of layer(s); we then consider in Appendix D.2.3 general partitions such that each group might contain a part of each layer and can also span across multiple layers. For the ease of presentation, we overload the notation $^{(p)}$ and $^{(l)}$, where the former always refers to group-wise quantities, and the latter refers to layer-wise quantities.

D.1 Preliminary: Memory footprint in standard training

Here, we provide a quick overview of the memory footprint in a standard training step that consists of a forward and a backward pass, where we simplify the transformer model into a stack of L MLP linear layers for discussion convenience.[2] Let the model with d total parameters $\theta_t = (\theta_t^{(1)}, \dots, \theta_t^{(L)}) \in \mathbb{R}^d$, with each layer $\theta_t^{(l)}$ having d/L parameters and activations of dimension $\sqrt{d/L}$ for simplicity. We use both the vector form $\theta_t^{(l)} \in \mathbb{R}^{d/L}$ and the matrix form $W_t^{(l)} \in \mathbb{R}^{\sqrt{d/L}\times\sqrt{d/L}}$ interchangeably, related by $\theta_t^{(l)} = \mathrm{vec}(W_t^{(l)})$; the same applies to weight gradients in matrix form $G^{(l)} \in \mathbb{R}^{\sqrt{d/L}\times\sqrt{d/L}}$ with $g^{(l)} = \mathrm{vec}(G^{(l)})$.

D.1.1 Forward and backward pass

Given a training batch $B_t = \{z_i\}_{i=1}^{n}$ of sequences of length T, a standard training step first performs a forward pass to evaluate the batch loss $\ell := \sum_{i=1}^{n} \ell_i$ with $\ell_i := \ell(\theta_t; z_i)$, caching intermediate activations, then a backward pass to compute the batch gradient $\hat{g}_{\mathrm{tr}} = (\hat{g}_{\mathrm{tr}}^{(1)}, \dots, \hat{g}_{\mathrm{tr}}^{(L)})$. Figure 6 illustrates both passes at a single layer l along with the resulting memory footprint. For simplicity, we omit the dimension of sequence length in the illustration.

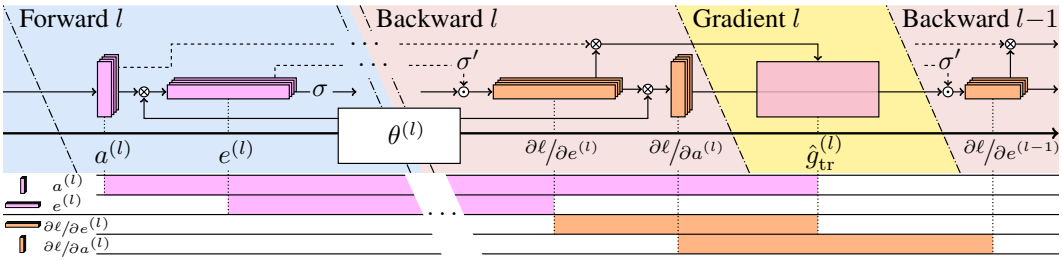

Figure 6: **Top:** Forward pass and backward pass (activation and weight gradient) computation graph: both $\longrightarrow$ and $\dashrightarrow$ denote *computational dependency*; $\dashrightarrow$ indicates that the dependency runs across the boundary between forward and backward. **Bottom:** Memory footprint: a tensor can be released once no remaining operation depends on it.

Forward pass. Each layer l computes the output (pre-activation) of the layer $e_{i,\tau}^{(l)} = W_t^{(l)} a_{i,\tau}^{(l)}$ from the input activation $a_{i,\tau}^{(l)} \in \mathbb{R}^{\sqrt{d/L}}$ for each sample $i \in [n]$ and token $\tau \in [\mathsf{T}]$, then applies an activation function σ: $a_{i,\tau}^{(l+1)} = \sigma(e_{i,\tau}^{(l)})$. In practice, all $n\mathsf{T}$ vectors are batched as columns of an activation matrix, so each layer reduces to a single matrix multiplication. All activations and pre-activations $\{a_{i,\tau}^{(l)}, e_{i,\tau}^{(l)}\}_{i,\tau}$ are *cached* for later use.

Backward pass. The activation gradient propagates backward layer by layer. At layer l, the pre-activation gradient $\partial\ell/\partial e_{i,\tau}^{(l)}$ arrives for each sample i and token τ. The activation gradient

[2] Throughout this section, "layer" refers to an individual linear module (e.g., `nn.Linear` in PyTorch), not a full transformer block. A single transformer block contains multiple linear modules (query, key, value, output projections, and MLP layers). Data regularization targets these linear modules, which account for the vast majority of transformer parameters; the treatment of embedding layers and other non-linear modules is discussed in Appendix D.5.

propagates to layer $l-1$ via $\partial\ell/\partial e_{i,\tau}^{(l-1)} = \sigma'(e_{i,\tau}^{(l-1)}) \odot \partial\ell/\partial a_{i,\tau}^{(l)}$ where $\partial\ell/\partial a_{i,\tau}^{(l)} = W_t^{(l)\top}\partial\ell/\partial e_{i,\tau}^{(l)}$ and $\odot$ denotes the element-wise product. The batch weight gradient is then computed as $\hat{g}_{\text{tr}}^{(l)} = \frac{1}{n}\sum_{i=1}^{n}\sum_{\tau=1}^{\mathsf{T}}(\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)} \in \mathbb{R}^{d/L}$, where $\otimes$ denotes the Kronecker product; for column vectors b and a we identify $b \otimes a$ with the rank-one matrix $ba^\top$ (its matrix form), so that $g_i^{(l)} = \sum_\tau(\partial\ell/\partial e_{i,\tau}^{(l)})(a_{i,\tau}^{(l)})^\top$ in matrix form, and we switch between the two forms freely as above. For later reference, we write the per-sample weight gradient as $g_i^{(l)} := \sum_{\tau=1}^{\mathsf{T}}(\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$, so that $\hat{g}_{\text{tr}}^{(l)} = \frac{1}{n}\sum_{i=1}^{n} g_i^{(l)}$. In practice, however, standard backpropagation does not explicitly form each $g_i^{(l)}$. Instead, it stacks all token-level activations and pre-activation gradients across the batch and evaluates the above double sum with a single batched matrix multiplication, which directly returns the aggregated batch gradient $\hat{g}_{\text{tr}}^{(l)}$. Thus, standard training computes the *batch gradient* without materializing separate *per-sample weight gradients* in memory.

D.1.2 Memory footprint of key tensors during training

During each step of the standard training update, a tensor can be released as soon as no remaining computation depends on it. As the goal is to compute $\hat{g}_{\text{tr}}$, a key observation is that during backward pass at layer $l-1$, both $a^{(l)}$ and $\partial\ell/\partial e^{(l)}$ become free once the batch gradient $\hat{g}_{\text{tr}}^{(l)}$ is assembled (Figure 6). Moreover, $\partial\ell/\partial e^{(l+1)}$ is released before $\partial\ell/\partial e^{(l)}$ is allocated, so its buffer is reused and the memory footprint of pre-activation gradients remains constant across layers.

Algorithm D.1 presents the pseudocode of one standard training step with memory management, which recovers the Full-Training Update. For simplicity, we hide the computation of $a_i^{(l)}$, $e_i^{(l)}$, their gradient, and also their dependency on each other in the backward function call. The creation and release of key tensors of interest are annotated with memory cost ($+$ allocation, $-$ release) to be compared with later algorithms. We note that when the training data is $B_t^\star$ instead of B_t, we recover Target-Only Update.

Algorithm D.1: Standard Training Update

Data: Model θ_t, training data $\{z_i\}_{i=1}^n$, learning rate η_t
Result: Updated model parameters θ_{t+1}

$(\ell, \{a_i^{(l)}, e_i^{(l)}\}_{l,i}) \leftarrow$ `Forward(`θ_t, $\{z_i\}_{i=1}^n$`)` // $+O(2n\mathsf{T}\sqrt{dL})$
$u_t \leftarrow 0 \in \mathbb{R}^d$ // $+O(d)$
for $l = L, \ldots, 1$ **do**
  `/* Allocate` $\{\partial\ell/\partial e_i^{(l)}\}_{i=1}^n$`, release` $\{e_i^{(l)}\}_{i=1}^n$`; memory remains constant */`
  $\{\partial\ell/\partial e_i^{(l)}\}_{i=1}^n \leftarrow$ `Backward(`ℓ, $\theta_t^{(l)}$`)` // $+O(n\mathsf{T}\sqrt{d/L}) - O(n\mathsf{T}\sqrt{d/L})$
  $u_t^{(l)} \leftarrow \frac{1}{n}\sum_{i=1}^{n}\sum_{\tau=1}^{\mathsf{T}}(\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$
  `Release(`$a_i^{(l)}$, $\partial\ell/\partial e_i^{(l)}$`)` for all $i \in [n]$ // $-O(2n\mathsf{T}\sqrt{d/L})$
$\theta_{t+1} \leftarrow \theta_t - \eta_t u_t$
`Release(`u_t`)` // $-O(d)$
return θ_{t+1}

D.2 Customized tensor lifetime scheduling for efficient data-regularized update

We now turn to a main system challenge introduced by the additional computational dependencies in data-regularized updates. Recall that, unlike standard training, our method must first determine the final update direction by solving a subset-selection problem based on per-sample scores, and only then assemble the corresponding gradient update. This creates additional computational dependencies on intermediate per-sample quantities that are not preserved in standard backpropagation.

Naive implementations. A naive implementation is a *two-pass* approach: use one forward-backward pass to compute the per-sample scores and determine the selected subset, and then run

a second forward-backward pass to compute the final gradient update using only the selected subset.[3] This approach is simple and memory-efficient because each pass follows the standard training schedule, but it substantially increases runtime by effectively duplicating the backward computation.

Another naive implementation is a *retain-graph* approach: retain the computation graph (e.g., by calling `retain_graph=True` in `PyTorch`) during backpropagation and keep all intermediate tensors alive so that the same computation graph can be reused for both scoring and gradient assembly. While this avoids the second pass, it is impractical at LLM scale, since retaining the full graph together with the per-sample quantities needed by our method leads to prohibitive memory overhead.

D.2.1 CUSTOMIZED TENSOR LIFETIME SCHEDULING

We develop a customized tensor lifetime scheduling to obtain the data-regularized gradient update within *one pass* while avoiding significant memory overhead.

For convenience, we consider *layer-aligned* partition in the rest of this subsubsection, where each group contains one or more layers. We discuss the extension to general partitions, where each group might contain partial parameters of a layer, in Appendix D.2.3 below.

A starting trick: merging training and target batch. We start by addressing a problem that the training gradients and target gradients are typically calculated in separate passes, which prevents effective tensor lifetime scheduling in one pass. We address this problem by realizing that these gradients can be obtained from a *merged batch* with both training and target samples. Specifically, suppose a training step uses a training batch $B_t = \{z_i\}_{i=1}^n$ and a target batch $B_t^\star = \{z_j^\star\}_{j=1}^m$. Rather than processing them in separate passes, we concatenate them into one batch of size $N = n + m$ and define the merged loss

$$\ell \coloneqq \sum_{i=1}^{n} \ell(\theta_t; z_i) + \sum_{j=1}^{m} \ell(\theta_t; z_j^\star).$$

Because different samples do not share activations, a single forward-backward pass on this merged loss produces all *per-sample* backward signals for both the training samples and the target samples. From these, we can extract the quantities needed for constructing the data-regularized update direction.

Remark D.1. *Doing forward and backward pass on the merged-batch loss $\ell \coloneqq \sum_{i=1}^n \ell_i + \sum_{j=1}^m \ell_j^\star$ is an existing trick in the literature (Pandya et al., 2025; Wang et al., 2024a). We note that this extends to non-loss-gradient target signals for an arbitrary θ_t-differentiable target signal $f(\{z_j^\star\}_{j=1}^m; \theta_t)$ (cf. Section 2.4): backward pass on the merged batch with $\ell \coloneqq \sum_{i=1}^n \ell_i + f(\{z_j^\star\}_{j=1}^m; \theta_t)$ yields both the per-sample training gradients and the target gradient $\nabla_\theta f$ in a single pass; however, when f depends on the training samples z_i, the gradient information is mixed and separate passes are required.*

The computational dependencies in data-regularized update. Next, given the merged batch where we can obtain the per-sample quantities from one pass, we carefully examine the computational dependencies for obtaining the data-regularized updates. For both Global Subset Update and Group-Wise Subset Update, the algorithm can be decomposed into three parts:

1. **Scoring:** compute the score of each training sample against the target signal (at group level);
2. **Subset selection:** determine the selected subset S_t (or $S_{t,p}$ for each group G_p);
3. **Update assembly:** assemble the final update direction by averaging the gradients over the selected subset.

The key issue is that both *scoring* and *update assembly* depend on per-sample quantities, while standard training only materializes the aggregated batch gradient $\hat{g}_{\text{tr}}$. In particular, the per-sample gradient $g_i^{(l)} = \sum_{\tau=1}^{\mathsf{T}} (\partial \ell / \partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$ is never explicitly stored in standard backpropagation. Yet for our method, it enters twice: first in the score computation, and later in the assembly of the selected subset. This is exactly the source of the extra dependencies that need to be handled.

A crucial observation is that both scoring and update assembly ultimately depend on the same local information at each layer, namely the pair $(a^{(l)}, \partial \ell / \partial e^{(l)})$. Once this pair is available, one can either

[3] For example, an existing online data selection method (Wang et al., 2024a) adopts a two-pass implementation. See their implementation in https://github.com/Jiachen-T-Wang/GREATS/ and an in-depth discussion later in this subsection.

materialize the corresponding per-sample gradients $g_i^{(l)}$ explicitly, or compute the required scores and selected averages directly from it. This suggests that the fundamental scheduling question is not whether to preserve the entire computation graph, but rather how to retain this minimal local information long enough for all downstream computations to be completed.

Tensor lifetime scheduling. Given the computational dependencies, we now introduce the proposed tensor lifetime scheduling.

The key idea is to *swap* the forward-cached tensor $e^{(l)}$ with the backward-generated tensor $\partial \ell / \partial e^{(l)}$ at each layer. Recall from Appendix D.1 that in standard training the forward pass caches $(a^{(l)}, e^{(l)})$, and during backward, $e^{(l)}$ is consumed to produce $\partial \ell / \partial e^{(l)}$. After this point, $e^{(l)}$ is no longer needed. Therefore, instead of releasing everything immediately after the layer gradient is computed, we retain the pair $(a^{(l)}, \partial \ell / \partial e^{(l)})$ for later scoring and update assembly, while discarding $e^{(l)}$. Since $\partial \ell / \partial e^{(l)}$ has the same shape as $e^{(l)}$, this replacement preserves the per-layer memory footprint up to lower-order bookkeeping costs.

Applying this swap iteratively across layers yields a one-pass schedule with the following structure:

1. run one forward pass on the merged batch and cache $(a^{(l)}, e^{(l)})$ as in standard training;
2. during backward, replace each cached $e^{(l)}$ with $\partial \ell / \partial e^{(l)}$, so that after the backward pass the retained tensors are $(a^{(l)}, \partial \ell / \partial e^{(l)})$ for all layers;
3. use these retained pairs to compute scores, determine the selected subset(s), and assemble the final update direction;
4. release the retained tensors once all groups depending on them have been resolved.

This schedule differs fundamentally from `retain_graph=True`: we do *not* keep the full autograd graph alive, but only the minimal local tensors needed for later computation. At the same time, unlike the two-pass implementation, we do *not* rerun the backward pass. The result is a one-pass implementation that stays close to the peak memory cost of standard training while avoiding a near-doubling of runtime.

When can a group be resolved? The remaining question is when a group can be finalized and its tensors released. This depends on the partition granularity. For a general partition $\mathcal{G} = \{G_p\}_{p=1}^{P}$, the subset $S_{t,p}$ for group G_p cannot be determined until all score contributions from the layers intersecting G_p have been accumulated. Only after that can the corresponding update $u_t^{(p)}$ be assembled. Therefore, a group can be released only after all of its constituent layers have finished both scoring and update assembly.

This reveals a direct connection between the statistical design and the systems design: finer partitions not only give more flexible update rules, but also allow earlier release of tensors and hence a memory footprint closer to standard training.[4]

D.2.2 Case study: Group granularity

We illustrate the above scheduling with two extreme cases.

Global Subset Update. For Global Subset Update, all parameters share a single global subset S_t. Consequently, the score of each training sample must aggregate contributions from *all* layers before S_t can be determined. This means the retained pairs $(a^{(l)}, \partial \ell / \partial e^{(l)})$ must remain alive across the entire model until scoring is complete. After the global subset S_t is determined, the final update is assembled by revisiting each retained layer and averaging the per-sample gradients of the selected samples (See Algorithm D.2). Compared with standard training, the peak memory remains close, but the high-memory period now spans almost the entire backward-and-selection phase (Middle row in Figure 7).

[4] It is worth noting that the peak memory cost remains similar regardless of the partition granularity, since the peak memory is achieved right after the forward pass. However, finer granularity becomes helpful when considering other memory-saving techniques, such as the activation checkpointing discussed in Appendix D.4.

Algorithm D.2: Global Subset Update

Data: Model θ_t, training data $\{z_i\}_{i=1}^n$, target data $\{z_j^\star\}_{j=1}^m$, learning rate η_t
Result: Updated model parameters θ_{t+1}

$(\ell, \{a_i^{(l)}, e_i^{(l)}\}_{l,i} \cup \{a_j^{\star(l)}, e_j^{\star(l)}\}_{l,j}) \leftarrow$ `Forward`$(\theta_t, \{z_i\}_{i=1}^n \cup \{z_j^\star\}_{j=1}^m)$ // $+O(2N\mathsf{T}\sqrt{dL})$
for $l = L, \ldots, 1$ **do** // Retain all pairs via swapping
  $\{\partial\ell/\partial e_i^{(l)}\}_{i=1}^n \cup \{\partial\ell/\partial e_j^{\star(l)}\}_{j=1}^m \leftarrow$ `Backward`$(\ell, \theta_t^{(l)})$
/* Post-hoc: scoring from retained activations */
for $l = L, \ldots, 1$ **do** // Peak: $2N\mathsf{T}\sqrt{dL}$
  $g_i^{(l)} \leftarrow \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$ for each $i \in [n]$ // $+O(nd/L)$
  $\hat{g}_\star^{(l)} \leftarrow \frac{1}{m} \sum_{j=1}^m \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{j,\tau}^{\star(l)}) \otimes a_{j,\tau}^{\star(l)}$ // $+O(d/L)$
  `Release`$(a_j^{\star(l)}, \partial\ell/\partial e_j^{\star(l)})$ for all $j \in [m]$ // $-O(2m\mathsf{T}\sqrt{d/L})$
  $s_i^{(l)} \leftarrow \langle \hat{g}_\star^{(l)}, g_i^{(l)} \rangle$ for each $i \in [n]$
  `Release`$(g_i^{(l)}, \hat{g}_\star^{(l)})$ for all $i \in [n]$ // $-O((n+1)d/L)$
$s_i \leftarrow \sum_{l=1}^L s_i^{(l)}$ for all $i \in [n]$ // Global scores
$S_t \leftarrow$ `Select-S`$(\{s_i\}_{i=1}^n, k)$
/* Post-hoc: assemble gradients from retained activations */
$u_t \leftarrow 0 \in \mathbb{R}^d$ // $+O(d)$
for $l = L, \ldots, 1$ **do**
  $u_t^{(l)} \leftarrow \frac{1}{k} \sum_{i \in S_t} \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$
  `Release`$(a_i^{(l)}, \partial\ell/\partial e_i^{(l)})$ for all $i \in [n]$ // $-O(2n\mathsf{T}\sqrt{d/L})$
$\theta_{t+1} \leftarrow \theta_t - \eta_t u_t$
`Release`(u_t) // $-O(d)$
return θ_{t+1}

Layer-Wise Subset Update. At the other extreme, for Layer-Wise Subset Update, each layer forms its own group. In this case, the score computation, subset selection, and update assembly for layer l depend only on the local tensors of that same layer. Therefore, once the backward pass reaches layer l, we can immediately compute the layer-wise scores, determine its subset $S_{t,l}$, assemble the update $u_t^{(l)}$, and release all retained tensors for that layer before moving to layer $l-1$ (See Algorithm D.3). As a result, the tensor lifetime schedule almost exactly matches that of standard training, and the memory footprint is closest to the standard baseline (Bottom row in Figure 7).

Algorithm D.3: Layer-Wise Subset Update

Data: Model θ_t, training data $\{z_i\}_{i=1}^n$, target data $\{z_j^\star\}_{j=1}^m$, learning rate η_t
Result: Updated model parameters θ_{t+1}

$(\ell, \{a_i^{(l)}, e_i^{(l)}\}_{l,i} \cup \{a_j^{\star(l)}, e_j^{\star(l)}\}_{l,j}) \leftarrow$ `Forward`$(\theta_t, \{z_i\}_{i=1}^n \cup \{z_j^\star\}_{j=1}^m)$ // $+O(2N\mathsf{T}\sqrt{dL})$
$u_t \leftarrow 0 \in \mathbb{R}^d$ // $+O(d)$
for $l = L, \ldots, 1$ **do**
  $\{\partial\ell/\partial e_i^{(l)}\}_{i=1}^n \cup \{\partial\ell/\partial e_j^{\star(l)}\}_{j=1}^m \leftarrow$ `Backward`$(\ell, \theta_t^{(l)})$
  $g_i^{(l)} \leftarrow \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$ for each $i \in [n]$ // $+O(nd/L)$
  $\hat{g}_\star^{(l)} \leftarrow \frac{1}{m} \sum_{j=1}^m \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{j,\tau}^{\star(l)}) \otimes a_{j,\tau}^{\star(l)}$ // $+O(d/L)$
  $s_i^{(l)} \leftarrow \langle \hat{g}_\star^{(l)}, g_i^{(l)} \rangle$ for each $i \in [n]$
  $S_{t,l} \leftarrow$ `Select-S`$(\{s_i^{(l)}\}_{i=1}^n, k)$
  $u_t^{(l)} \leftarrow \frac{1}{k} \sum_{i \in S_{t,l}} g_i^{(l)}$
  `Release`$(g_i^{(l)}, \hat{g}_\star^{(l)})$ for all $i \in [n]$ // $-O((n+1)d/L)$
  `Release`$(a_i^{(l)}, \partial\ell/\partial e_i^{(l)}, a_j^{\star(l)}, \partial\ell/\partial e_j^{\star(l)})$ for all $i \in [n], j \in [m]$ // $-O(2N\mathsf{T}\sqrt{d/L})$
$\theta_{t+1} \leftarrow \theta_t - \eta_t u_t$
`Release`(u_t) // $-O(d)$
return θ_{t+1}

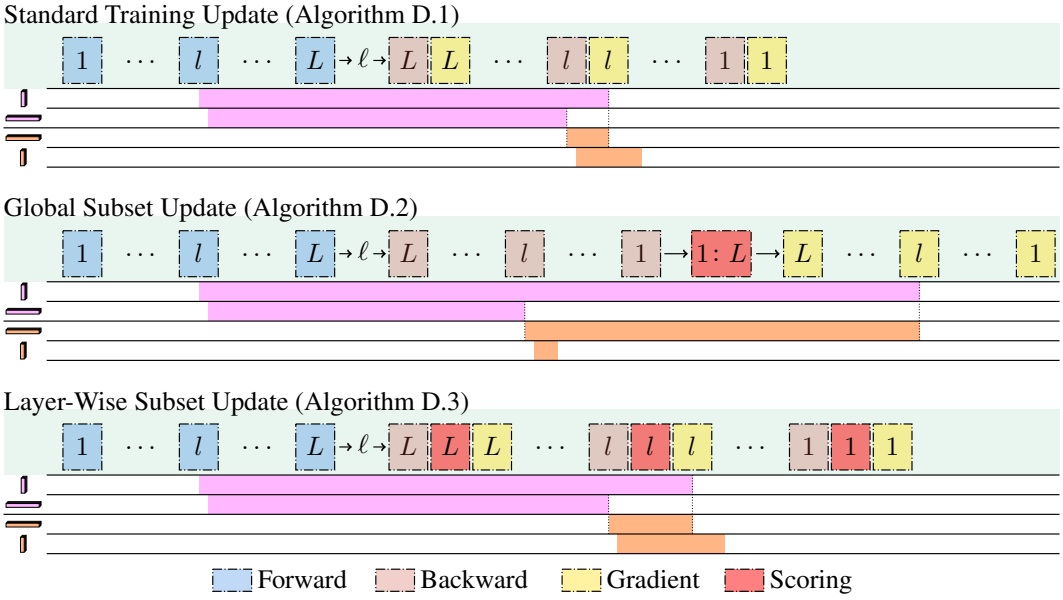

Figure 7: Memory footprint comparison. **Top:** Standard Training Update (Algorithm D.1). **Middle:** Global Subset Update (Algorithm D.2). **Bottom:** Layer-Wise Subset Update (Algorithm D.3). All three methods achieve comparable peak memory per layer, but Global Subset Update must retain both $a^{(l)}$ and $\partial\ell/\partial e^{(l)}$ for all l throughout the backward pass until the global subset is determined, while standard training and Layer-Wise Subset Update release them on-the-fly.

Intermediate granularities. More generally, if each group contains a small number of consecutive layers, then a group's tensors only need to be retained until the backward pass finishes those layers. This gives a smooth interpolation between the two extremes above. In particular, smaller groups allow earlier release and lower sustained memory usage, whereas larger groups delay release and make the schedule resemble Global Subset Update.

Remark D.2. *One benefit of more aggressive memory release (as in Layer-Wise Subset Update) is that, in practice, when layer sizes are not uniform, later in the backward pass, encountering a larger layer provides enough free memory to process it.*

In summary, the proposed customized tensor lifetime schedule resolves a core system challenge of data-regularized updates: it avoids the runtime overhead of a two-pass implementation and the prohibitive memory cost of retaining the full graph, while enabling a practical one-pass realization under the memory constraints of modern LLM training.

D.2.3 General partition

The one-pass tensor lifetime schedule above focuses on layer-aligned partitions, where each group G_p is a union of whole layers. Here, we discuss the additional complications that arise for arbitrary partitions whose groups may split parameters within a single layer.

Layer-by-layer discipline. After the backward pass, all retained pairs $(a^{(l)}, \partial\ell/\partial e^{(l)})$ are available. To stay within the memory budget, score computation and gradient assembly must still proceed layer by layer. At each layer l, we compute per-layer score contributions and accumulate them into the scores for each group. A group G_p becomes resolvable once all layers intersecting G_p have been visited, at which point $S_{t,p}$ is determined. The update direction $u^{(p)}$ is then assembled from the per-sample gradients of the selected samples at layers intersecting G_p, and the retained quantities at layer l are released once all groups intersecting that layer have been resolved.

Non-layer-aligned partitions. The layer-by-layer discipline introduces a tension between memory and recomputation for partitions that do not align with the layer structure. When a group G_p spans multiple layers, $S_{t,p}$ is not known until all layers in the group have been scored. If the scoring strategy materializes per-sample gradients $g_i^{(l)}$ at each layer, these cannot in general be retained across the group's full span without exceeding the memory budget; they must be discarded after scoring, and recomputed from the retained pairs $(a^{(l)}, \partial\ell/\partial e^{(l)})$ when $u_t^{(p)}$ is assembled.

- *Scoring.* We now show how to compute the per-group scores $s_i^{(p)}$ during the backward pass. A crucial observation is that the score decomposes linearly over *parameter space*, i.e., $s_i^{(p)} = \sum_{j \in G_p} g_{i,j} \cdot \hat{g}_{\star,j}$ where $g_{i,j}$ denotes the jth entry of g_i, similarly for $\hat{g}_{\star,j}$. Hence, computing $s_i^{(p)}$ reduces to computing the *per-parameter* score $s_{i,j} \coloneqq g_{i,j} \cdot \hat{g}_{\star,j}$ for each $i \in [n]$ and $j \in [d]$. This is realized via layer-wise accumulation: for each p, we initialize $s_i^{(p)} \leftarrow 0$. Then during the backward pass at layer l, we first materialize the per-sample gradient $g_i^{(l)}$ and the average target gradient $\hat{g}_\star^{(l)}$, and obtain all per-parameter scores $s_{i,j}$ by computing $\hat{g}_\star^{(l)} \odot g_i^{(l)} \in \mathbb{R}^{d/L}$ for all j belonging to layer l. Finally, we accumulate $s_i^{(p)} \leftarrow s_i^{(p)} + \sum_{j \in G_p} s_{i,j}$, and continue the backward pass to layer $l+1$.

  Remark D.3. *We highlight that the scoring methods discussed in Appendix D.3, except for Direct, therefore can not handle general partition, as they do not materialize $\hat{g}_\star^{(l)}$ and $g_i^{(l)}$ explicitly.*

- *Update assembly.* Recall that for layer-aligned partitions, we can break down the construction of $u_t^{(p)}$ into layers. Similarly, in the case of general partitions, we again follow the layer-by-layer discipline, where for each layer, we first identify groups G_p that have non-empty intersections with it, and for each of these groups, what are the corresponding parameters in the intersection. Then, we materialize all the per-sample gradients $\{g_i^{(l)}\}_{i=1}^n$ of the entire layer, and resolve for each group by first computing the average over samples $i \in S_{t,p}$ with sub-vectors of $g_i^{(l)}$ that corresponds to parameters that belongs to the group.

D.2.4 TWO-PASS IMPLEMENTATION

Recall from Appendix D.1 that in standard training, both $a^{(l)}$ and $\partial \ell / \partial e^{(l)}$ can be released as soon as the batch gradient $\hat{g}_{\text{tr}}^{(l)}$ is assembled, that is, once the update direction at layer l is decided. For Group-Wise Subset Update, however, the update direction at each group p depends on $S_{t,p}$, and $S_{t,p}$ can only be determined after scoring all samples across every layer whose parameters intersect G_p. Standard training releases the per-sample quantities needed for gradient assembly before $S_{t,p}$ is known. For Layer-Wise Subset Update, each group corresponds to a single layer, so $S_{t,p}$ can be determined after scoring all samples within that layer, and per-sample quantities can be released immediately. For coarser partitions, however, a two-pass approach trades extra computation for standard memory.

Separate scoring and gradient passes. One solution is to use two passes. A first *scoring pass* accumulates per-group scores $s_i^{(p)}$ layer by layer during a standard forward-backward pass as described in Appendix D.2.3. After $S_{t,p}$ is determined, a second *gradient pass* re-runs forward-backward on the union $\bigcup_p S_{t,p}$: at each layer l, the gradient for each group p whose parameters intersect that layer is assembled from the samples in $S_{t,p}$, which again follow the same strategy described in Appendix D.2.3 in the case of general partitions.

As a concrete instance, consider Global Subset Update, where all parameters form a single group so that $S_{t,p} = S_t$ is global. The per-sample score decomposes as $s_i = \sum_{l=1}^{L} s_i^{(l)}$, so the scoring pass accumulates per-layer contributions into one global score per sample. The gradient pass simplifies to a forward-backward on the k selected samples, at an additional cost of recomputation. GREATS (Wang et al., 2024a) adopts this two-pass design; Algorithm D.4 presents the full pseudocode.

Algorithm D.4: Global Subset Update (two-pass)

Data: Model θ_t, training data $\{z_i\}_{i=1}^n$, target data $\{z_j^\star\}_{j=1}^m$, learning rate η_t
Result: Updated model parameters θ_{t+1}

/* Scoring pass */
$(\ell, \{a_i^{(l)}, e_i^{(l)}\}_{l,i} \cup \{a_j^{\star(l)}, e_j^{\star(l)}\}_{l,j}) \leftarrow \texttt{Forward}(\theta_t, \{z_i\}_{i=1}^n \cup \{z_j^\star\}_{j=1}^m)$ // $+O(2N\mathsf{T}\sqrt{dL})$
for $l = L, \ldots, 1$ **do**
  $\{\partial \ell / \partial e_i^{(l)}\}_{i=1}^n \cup \{\partial \ell / \partial e_j^{\star(l)}\}_{j=1}^m \leftarrow \texttt{Backward}(\ell, \theta_t^{(l)})$
  $s_i^{(l)} \leftarrow \texttt{Score}(a_i^{(l)}, \partial \ell / \partial e_i^{(l)}, \{a_j^{\star(l)}, \partial \ell / \partial e_j^{\star(l)}\}_{j=1}^m)$ for each $i \in [n]$
   // Appendix D.3
  $\texttt{Release}(a_i^{(l)}, \partial \ell / \partial e_i^{(l)}, a_j^{\star(l)}, \partial \ell / \partial e_j^{\star(l)})$ for all i, j // $-O(2N\mathsf{T}\sqrt{d/L})$
$s_i \leftarrow \sum_{l=1}^{L} s_i^{(l)}$ for all $i \in [n]$ // Global scores
$S_t \leftarrow \texttt{Select-S}(\{s_i\}_{i=1}^n, k)$
/* Gradient pass on subset S_t */
$(\ell_{S_t}, \{a_i^{(l)}\}_{l,i}) \leftarrow \texttt{Forward}(\theta_t, \{z_i\}_{i \in S_t})$ // $+O(k\mathsf{T}\sqrt{dL})$
for $l = L, \ldots, 1$ **do**
  $\{\partial \ell_{S_t} / \partial e_i^{(l)}\}_{i \in S_t} \leftarrow \texttt{Backward}(\ell_{S_t}, \theta_t^{(l)})$ // $-O(k\mathsf{T}\sqrt{d/L}) + O(k\mathsf{T}\sqrt{d/L})$
  $u_t^{(l)} \leftarrow \frac{1}{k} \sum_{i \in S_t} \sum_{\tau=1}^{\mathsf{T}} (\partial \ell_{S_t} / \partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$ // $+O(d/L)$
  $\texttt{Release}(a_i^{(l)}, \partial \ell_{S_t} / \partial e_i^{(l)})$ for all $i \in S_t$ // $-O(2k\mathsf{T}\sqrt{d/L})$
  $\theta_{t+1}^{(l)} \leftarrow \theta_t^{(l)} - \eta_t u_t^{(l)}$
  $\texttt{Release}(u_t^{(l)})$ // $-O(d/L)$
return θ_{t+1}

Compatibility with memory-saving techniques. The two-pass design is compatible with both activation checkpointing and gradient accumulation, since each pass independently preserves the standard memory profile. For activation checkpointing, when the partition does not align with the checkpointing schedule (i.e., some group G_p spans multiple checkpoint segments), the scoring pass computes per-group scores under standard checkpointing, releasing activations at segment boundaries, and the gradient pass performs a checkpointed forward-backward over $\bigcup_p S_{t,p}$. For

gradient accumulation with rules that depend on global batch statistics (e.g., top-k), the scoring pass processes all micro-batches to determine $S_{t,p}$ for every group, and the gradient pass assembles gradients from $\bigcup_p S_{t,p}$, itself split into micro-batches if needed.

D.3 Efficient implementations of per-sample scoring

We next study efficient implementations of per-sample scoring, which constitutes a major computational overhead introduced by data-regularized updates. While the tensor lifetime schedule in Appendix D.2 resolves the memory-management challenge of preserving the necessary intermediate quantities within one pass, the overall efficiency of the method still depends critically on how the per-sample scores are computed.

In this subsection, we revisit two existing methods for score computation and introduce two new variants. We then provide a systematic complexity analysis in terms of the key training hyperparameters. This analysis reveals that the relative advantages of different implementations depend strongly on the hyperparameter regime, and that this regime-dependent comparison has not been fully characterized in the existing literature. Among these methods, one of our proposed variants—an approximate scoring method—achieves the best computational efficiency across all regimes.

Throughout this subsection, we focus on the score restricted to one particular layer, $s_i^{(l)}$, since when a parameter group spans multiple layers, the total score is simply the sum of the scores from those layers. For each scoring method, we directly elaborate on the exact computational and memory complexity. To this end, we let $w = \sqrt{d/L}$ denote the layer width for brevity, $N = n + m$, write $\kappa \equiv \kappa^{(l)}$ for the per-layer compressed dimension of Compressed Scoring, and count every addition and multiplication as one operation; memory is measured in scalar entries under fully batched execution. All quantities carry a layer index $^{(l)}$ that we suppress in the complexity derivations for brevity.

Table 5: Computational overhead of scoring per layer for n training samples and m target samples, where $N = n + m$, T denotes the sequence length, and κ is the compressed gradient dimension ($\kappa \ll d/L$). All costs listed below represent additional overhead beyond standard backpropagation. We assume fully batched execution in calculating the complexity; micro-batching over samples or tokens can trade parallelism for lower memory, which further favors the proposed PIP method.

Scoring method	**FLOPs per layer**	**Memory per layer (entries)**
Direct	$2N\mathsf{T}(d/L) + n(d/L)$	$(n+1)(d/L)$
GIP	$4nm\mathsf{T}^2\sqrt{d/L}$	$2nm\mathsf{T}^2$
PIP	$2N\mathsf{T}(d/L) + n\mathsf{T}\sqrt{d/L}$	$d/L + n\mathsf{T}\sqrt{d/L}$
Compressed	$O(N\mathsf{T}\kappa) + (2n+m)\kappa$	$(n+1)\kappa$

D.3.1 Existing methods: Direct and Ghost Inner Product (GIP)

We first revisit two existing methods, *Direct* and *Ghost Inner Product* (GIP).

Direct. Direct is a naive approach for the score computation. It directly materializes each per-sample gradient $g_i^{(l)} = \sum_\tau (\partial \ell / \partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$ and the target gradient $\hat{g}_\star^{(l)}$ first, then computes the score as $s_i^{(l)} = \langle \hat{g}_\star^{(l)}, g_i^{(l)} \rangle$. Existing literature criticizes this naive approach for materializing $O(n)$ per-sample gradient vectors and incurring high computational costs, which led to the GIP introduced below (Wang et al., 2024a). However, we demonstrate that the comparison between Direct and GIP is more nuanced than the literature suggested when considering the sequence length T and the number of target samples m. Direct's cost decomposes into three steps:

1. *Materialize n training gradient matrices* $g_i = \sum_{\tau=1}^{\mathsf{T}} (\partial \ell / \partial e_{i,\tau}) \otimes a_{i,\tau}$. Each Kronecker product costs w^2 multiplications; summing T outer products element-wise adds $(\mathsf{T}-1)w^2$ additions. Per sample: $(2\mathsf{T}-1)w^2$. Total: $n(2\mathsf{T}-1)w^2$.
2. *Materialize* $\hat{g}_\star = \frac{1}{m}\sum_j g_j^\star$. Same Kronecker-product accumulation for m target samples: $m(2\mathsf{T}-1)w^2$. Summing the m matrices: $(m-1)w^2$ additions. Scaling by $1/m$: w^2 multiplications. Total: $m(2\mathsf{T}-1)w^2 + (m-1)w^2 + w^2 = 2m\mathsf{T}w^2$.

3. *Score.* Each Frobenius inner product $s_i = \langle \hat{g}_\star, g_i \rangle$ (the Euclidean inner product of the vectorized matrices) costs w^2 multiplications + (w^2-1) additions = $(2w^2-1)$ per sample. Total: $n(2w^2-1)$.

Summing the three steps yields total FLOPs of $n(2\mathsf{T}-1)w^2 + 2m\mathsf{T}w^2 + n(2w^2-1) = 2N\mathsf{T}w^2 + n(w^2-1)$, and a memory footprint consisting of the n gradient matrices $g_i \in \mathbb{R}^{w\times w}$ plus $\hat{g}_\star$, totaling $(n+1)w^2$ entries.

Ghost Inner Product (GIP) (Wang et al., 2024a). GIP avoids the full gradient materialization (thus the name "Ghost") by evaluating the same inner product in an alternative contraction order from $a^{(l)}$ and $\partial\ell/\partial e^{(l)}$: contracting over the model dimension instead of the token dimension yields $\mathsf{T}\times\mathsf{T}$ cross-correlations. Treating $\partial\ell/\partial e^{(l)}, a^{(l)} \in \mathbb{R}^{\sqrt{d/L}\times\mathsf{T}}$ as matrices whose columns are the per-token vectors, the score is

$$s_i^{(l)} = \frac{1}{m}\sum_{j=1}^{m}\left\langle \left(\frac{\partial\ell}{\partial e_i^{(l)}}\right)^\top \frac{\partial\ell}{\partial e_j^{\star(l)}}, (a_i^{(l)})^\top a_j^{\star(l)} \right\rangle,$$

where each factor is a $\mathsf{T}\times\mathsf{T}$ matrix product computed over all nm training-target pairs. While this avoids gradient materialization, it introduces quadratic dependence on T and linear dependence on m. Concretely, for each training-target pair (i,j), forming the activation-gradient cross-correlation $(\partial\ell/\partial e_i)^\top(\partial\ell/\partial e_j^\star) \in \mathbb{R}^{\mathsf{T}\times\mathsf{T}}$ and the activation cross-correlation $(a_i)^\top a_j^\star \in \mathbb{R}^{\mathsf{T}\times\mathsf{T}}$ each costs $\mathsf{T}^2(2w-1)$ (each entry is a dot product of w-vectors: w multiplications + $(w-1)$ additions, giving $\mathsf{T}^2(2w-1)$ per matrix), so two such matrices cost $2\mathsf{T}^2(2w-1)$; the subsequent Frobenius inner product over the two $\mathsf{T}\times\mathsf{T}$ matrices is element-wise multiply and sum, costing T^2 multiplications + (T^2-1) additions = $(2\mathsf{T}^2-1)$. Per pair: $2\mathsf{T}^2(2w-1) + (2\mathsf{T}^2-1) = 4\mathsf{T}^2w - 1$. Accumulating m pairs per training sample adds $(m-1)$ additions and 1 multiplication (the $1/m$ scaling), giving $m(4\mathsf{T}^2w-1) + m = 4m\mathsf{T}^2w$ per training sample. Across n training samples, the total FLOPs is $4nm\mathsf{T}^2w$, and the memory consists of two $\mathsf{T}\times\mathsf{T}$ cross-correlation matrices per (i,j) pair, batched over all pairs, giving $2nm\mathsf{T}^2$ entries.

Comparison between Direct and GIP. As shown in Table 5, GIP is competitive only when m and T are relatively small. By comparing the leading terms in the computational complexity, GIP is preferable to Direct only when $m\mathsf{T} \lesssim \sqrt{d/L}/2$. A similar trend holds for the memory cost comparison as well.

D.3.2 Per-Token Inner Product (PIP)

The comparison shows that GIP suffers from long context (large T) while Direct suffers from large parameter size per layer (large d/L). We now introduce a third method, *Per-Token Inner Product* (PIP), that interpolates between these two methods and provides a more favorable computation cost trade-off in an important regime.

Observing that the GIP differs from Direct by swapping the order of some linear operators (sum and inner product) in calculating the score $s_i^{(l)}$, our proposed PIP applies a different order. Let $\hat{G}_\star^{(l)} \in \mathbb{R}^{\sqrt{d/L}\times\sqrt{d/L}}$ denote the matrix form of the *aggregated* target gradient $\hat{g}_\star^{(l)}$. Then the score can be written as

$$s_i^{(l)} = \sum_{\tau=1}^{\mathsf{T}}\left(\frac{\partial\ell}{\partial e_{i,\tau}^{(l)}}\right)^\top \hat{G}_\star^{(l)} a_{i,\tau}^{(l)}.$$

This expression factors the computation into a matrix-vector product $\hat{G}_\star^{(l)} a_{i,\tau}^{(l)} \in \mathbb{R}^{\sqrt{d/L}}$, followed by a dot product with $\partial\ell/\partial e_{i,\tau}^{(l)}$. Importantly, this avoids materializing the full per-sample training gradient $g_i^{(l)}$: the score is accumulated directly from token-level quantities that are already available under the tensor lifetime schedule. Concretely, PIP has two steps:

1. *Materialize* $\hat{G}_\star$, identical to Step 2 of Direct, with cost $2m\mathsf{T}w^2$.
2. *Score.* For each training token (i,τ): a matrix-vector product $\hat{G}_\star a_{i,\tau} \in \mathbb{R}^w$ costs $w(2w-1)$ (each of w entries is a dot product of w-vectors), then a dot product with $\partial\ell/\partial e_{i,\tau}$ costs $(2w-1)$. Per token: $w(2w-1) + (2w-1) = (w+1)(2w-1) = 2w^2 + w - 1$. Accumulating T tokens into s_i: $\mathsf{T}(2w^2+w-1) + (\mathsf{T}-1) = 2\mathsf{T}w^2 + \mathsf{T}w - 1$ per sample. Total: $n(2\mathsf{T}w^2 + \mathsf{T}w - 1)$.

Summing yields total FLOPs of $2m\mathsf{T}w^2 + n(2\mathsf{T}w^2 + \mathsf{T}w - 1) = 2N\mathsf{T}w^2 + n(\mathsf{T}w - 1)$, and a memory footprint consisting of $\hat{G}_\star \in \mathbb{R}^{w\times w}$ plus the batched intermediates $\hat{G}_\star A \in \mathbb{R}^{w\times n\mathsf{T}}$, giving $w^2 + n\mathsf{T}w$ entries.

Comparison with existing methods. As shown in Table 5, the complexities of PIP remain linear in the sequence length T, and therefore avoid the quadratic dependence on T that makes GIP unattractive for long-context training. Compared with Direct, PIP has the same dominant FLOPs term $2N\mathsf{T}(d/L)$, but replaces the lower-order term $n(d/L)$ with $n\mathsf{T}\sqrt{d/L}$. Therefore, when $\mathsf{T} \lesssim \sqrt{d/L}$, PIP is more efficient than Direct in both FLOPs and memory. Intuitively, in this regime, it is cheaper to work directly with token-level quantities than to first materialize n full per-sample gradients. When T becomes large, however, this advantage disappears in the FLOP count: the additional $n\mathsf{T}\sqrt{d/L}$ term dominates, and Direct becomes preferable, although in practice the gap stays small because a fused PIP kernel keeps that intermediate in registers while Direct's $n(d/L)$ term is materialized in memory (Table 4(b)). Thus, PIP occupies an intermediate regime between GIP and Direct: it improves substantially over GIP for long sequences, and improves over Direct when the context length is moderate relative to the per-layer hidden dimension.

D.3.3 Compressed Scoring

All three methods above compute the score exactly, and therefore their cost necessarily depends on the gradient dimension d/L. We now introduce a fourth method, *Compressed Scoring*, which reduces this cost by approximately computing the score in a low-dimensional compressed space.

Let $\Pi^{(l)} : \mathbb{R}^{d/L} \to \mathbb{R}^\kappa$ be a compression map with $\kappa \ll d/L$. Instead of directly computing the full per-sample training gradient $g_i^{(l)}$ and target gradient $\hat{g}_\star^{(l)}$, we first compute their compressed versions

$$\widetilde{g}_i^{(l)} \coloneqq \Pi^{(l)}(g_i^{(l)}) \in \mathbb{R}^\kappa, \qquad \widetilde{g}_\star^{(l)} \coloneqq \Pi^{(l)}(\hat{g}_\star^{(l)}) \in \mathbb{R}^\kappa,$$

and then approximate the score by

$$s_i^{(l)} \approx \langle \widetilde{g}_\star^{(l)}, \widetilde{g}_i^{(l)} \rangle.$$

The key point is that $\widetilde{g}_i^{(l)}$ and $\widetilde{g}_\star^{(l)}$ can be computed directly from the retained token-level quantities $(a^{(l)}, \partial\ell/\partial e^{(l)})$, without ever materializing the full d/L-dimensional gradients. In particular, since $g_i^{(l)} = \sum_{\tau=1}^{\mathsf{T}} (\partial\ell/\partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)}$, state-of-the-art gradient compression methods in data attribution literature (Hu et al., 2025a; Choe et al., 2025) can exploit this outer-product structure and apply the projection $\Pi^{(l)}$ in a factorized manner; we elaborate on this factorized projection machinery in Appendix D.3.4 below. As a result, the compressed per-sample gradient can be obtained from the same retained local tensors used throughout Appendix D.2, but with cost scaling only in the compressed dimension κ. Concretely, Compressed Scoring decomposes into three steps:

1. *Compress all N per-sample gradients* to $\widetilde{g}_i \in \mathbb{R}^{\kappa^{(l)}}$ via the projection Π. The per-token cost depends on the Kronecker structure of Π (see Appendix D.3.4); the total is $O(N\mathsf{T}\kappa^{(l)})$.
2. *Aggregate target gradient* $\widetilde{g}_\star = \frac{1}{m}\sum_j \widetilde{g}_j^\star$. Summing m vectors of dimension $\kappa^{(l)}$: $(m-1)\kappa^{(l)}$ additions. Scaling by $1/m$: $\kappa^{(l)}$ multiplications. Total: $m\kappa^{(l)}$.
3. *Score.* Each inner product $s_i = \langle \widetilde{g}_\star, \widetilde{g}_i \rangle$ costs $\kappa^{(l)}$ multiplications + $(\kappa^{(l)} - 1)$ additions = $(2\kappa^{(l)} - 1)$ per sample. Total: $n(2\kappa^{(l)} - 1)$.

Summing yields total FLOPs of $O(N\mathsf{T}\kappa^{(l)}) + (2n + m)\kappa^{(l)}$, and a memory footprint of $(n+1)\kappa^{(l)}$ (the n compressed training gradients plus $\widetilde{g}_\star$).

Comparison with exact methods. As shown in Table 5, Compressed Scoring has per-layer cost $O(N\mathsf{T}\kappa) + (2n + m)\kappa$ and memory cost $(n+1)\kappa$, which removes the dependence on the original gradient dimension d/L from both FLOPs and memory. Since $\kappa \ll d/L$, this makes Compressed Scoring strictly more efficient than all three exact methods across all practical regimes in terms of the leading dimension dependence, and Table 4(b) confirms that it is the cheapest method at every T and m we test.

While the gain in efficiency of Compressed Scoring comes at the price of approximation, empirically, we find the approximate scores remain a reliable surrogate for ranking or filtering training samples (Hu

et al., 2025a; 2026). In our implementation, both backends are available: the system benchmarks use exact GIP scoring, the backend of the question-answering experiments (Appendix E.4), Compressed Scoring with $\kappa^{(l)} = 64 \times 64$ is the choice for long contexts and many target examples, where its cost advantage is largest (Table 4(b)), and under LoRA, whose per-sample gradients are already small, we use exact PIP. Appendices E.1.3 and E.2.5 compare the two backends.

D.3.4 EFFICIENT PER-SAMPLE GRADIENT PROJECTION

We now provide a self-contained discussion on the state-of-the-art per-sample gradient compression (Hu et al., 2025a) that underlies Compressed Scoring above; the same compression machinery is also used in the MeSO integration in Appendix D.4.1.

Factorized projection. State-of-the-art gradient compressors (Hu et al., 2025a; Choe et al., 2025) consider a projection operator $\Pi \in \mathbb{R}^{\kappa \times d}$ from dimension $d := d_{\text{in}} d_{\text{out}}$ to dimension $\kappa \ll d$ with a two-stage structure:

$$\Pi = P_{\text{final}} \left(P_{\text{in}} \otimes P_{\text{out}} \right),$$

where $P_{\text{in}} \in \mathbb{R}^{\kappa_{\text{in}} \times d_{\text{in}}}$ and $P_{\text{out}} \in \mathbb{R}^{\kappa_{\text{out}} \times d_{\text{out}}}$ are the first-stage projections, $P_{\text{in}} \otimes P_{\text{out}}$ denotes their Kronecker product, and $P_{\text{final}} \in \mathbb{R}^{\kappa \times \kappa_{\text{in}} \kappa_{\text{out}}}$ is a small second-stage projection.

Remarkably, for any input vectors that admit such a factorized structure, the factorized projection can be computed efficiently without materializing the full $\mathbb{R}^{\kappa \times d}$ representation of Π, and can directly operate on the input factors in $\mathbb{R}^{d_{\text{in}}}$ and $\mathbb{R}^{d_{\text{out}}}$, enabling efficient forward and back-projection. *Forward projection.* For any input with outer-product structure $b \otimes a$ where $a \in \mathbb{R}^{d_{\text{in}}}$ and $b \in \mathbb{R}^{d_{\text{out}}}$, the forward projection evaluates as

$$\Pi \operatorname{vec}(b \otimes a) = P_{\text{final}} \operatorname{vec}\left((P_{\text{out}} b) \otimes (P_{\text{in}} a) \right),$$

requiring only two small matrix-vector products ($P_{\text{in}} a \in \mathbb{R}^{\kappa_{\text{in}}}$ and $P_{\text{out}} b \in \mathbb{R}^{\kappa_{\text{out}}}$) followed by the second-stage projection, without forming $\operatorname{vec}(b \otimes a) \in \mathbb{R}^d$. By linearity, this extends to sums of outer products: for $X = \sum_\tau b_\tau \otimes a_\tau$,

$$\widetilde{X} = \Pi \operatorname{vec}(X) = P_{\text{final}} \operatorname{vec}\left(\sum_\tau (P_{\text{out}} b_\tau) \otimes (P_{\text{in}} a_\tau) \right),$$

where each term contributes a small outer product in $\mathbb{R}^{\kappa_{\text{out}} \times \kappa_{\text{in}}}$. *Backward projection.* For any vector $\widetilde{x} \in \mathbb{R}^\kappa$, the transpose map $\Pi^\top \widetilde{x} \in \mathbb{R}^d$ is equally efficient. Let $X' \in \mathbb{R}^{\kappa_{\text{out}} \times \kappa_{\text{in}}}$ be the matrix obtained by reshaping $(P_{\text{final}})^\top \widetilde{x}$, and let Mat denote the inverse of vec, which reshapes a vector in $\mathbb{R}^d$ into a $d_{\text{out}} \times d_{\text{in}}$ matrix. The Kronecker structure gives

$$\operatorname{Mat}(\Pi^\top \widetilde{x}) = (P_{\text{out}})^\top X' P_{\text{in}},$$

a pair of small matrix multiplications, without forming the full matrix $\Pi^\top \in \mathbb{R}^{d \times \kappa}$.

Per-sample gradient compression. We now show how to apply this compressor to compress linear layers' per-sample gradients. *Notation.* For generality, we do not assume that each linear layer is square: specifically, the weight at layer l is $W^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l)} \times d_{\text{in}}^{(l)}}$ with $d^{(l)} := d_{\text{in}}^{(l)} d_{\text{out}}^{(l)}$, related to the vectorized form by $\theta^{(l)} = \operatorname{vec}(W^{(l)})$. Per-training-sample weight gradients are correspondingly denoted $G_i^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l)} \times d_{\text{in}}^{(l)}}$, so that $g_i^{(l)} = \operatorname{vec}(G_i^{(l)})$. Similarly for $G_j^\star$ for per-target-sample gradients. Let $\langle \cdot, \cdot \rangle$ denote the Frobenius inner product, i.e., the Euclidean inner product of the vectorized forms. *Efficient per-sample gradient compression.* Recall that the per-sample weight gradient for sample z_i at layer l is a sum of outer products over tokens: $g_i^{(l)} = \sum_{\tau=1}^{\mathsf{T}} (\partial \ell / \partial e_{i,\tau}^{(l)}) \otimes a_{i,\tau}^{(l)} \in \mathbb{R}^{d^{(l)}}$. Consider a compressor $\Pi^{(l)} := P_{\text{final}}^{(l)} (P_{\text{in}}^{(l)} \otimes P_{\text{out}}^{(l)})$, with $P_{\text{in}}^{(l)} \in \mathbb{R}^{\kappa_{\text{in}}^{(l)} \times d_{\text{in}}^{(l)}}$ and $P_{\text{out}}^{(l)} \in \mathbb{R}^{\kappa_{\text{out}}^{(l)} \times d_{\text{out}}^{(l)}}$, and $P_{\text{final}}^{(l)} \in \mathbb{R}^{\kappa^{(l)} \times \kappa_{\text{in}}^{(l)} \kappa_{\text{out}}^{(l)}}$. Applying the forward projection gives the compressed per-sample gradient

$$\widetilde{g}_i^{(l)} = P_{\text{final}}^{(l)} \operatorname{vec}\left(\sum_{\tau=1}^{\mathsf{T}} \left(P_{\text{out}}^{(l)} \frac{\partial \ell}{\partial e_{i,\tau}^{(l)}} \right) \otimes \left(P_{\text{in}}^{(l)} a_{i,\tau}^{(l)} \right) \right) \in \mathbb{R}^{\kappa^{(l)}}.$$

We note that this can be computed directly from the cached activations and activation gradients, without materializing $g_i^{(l)}$. With careful choice of $P_{\text{in}}^{(l)}$, $P_{\text{out}}^{(l)}$, and $P_{\text{final}}^{(l)}$, the per-sample memory is

$O(\kappa^{(l)})$ instead of $O(d^{(l)})$, and the per-token computational cost is $O(\kappa^{(l)})$ (Hu et al., 2025a), so compressing all n samples across T tokens and L layers costs $O(n\mathsf{T}\sum_l \kappa^{(l)})$ in total. Later, we write this operation as $\Pi^{(l)}(a_i^{(l)}, \partial\ell/\partial e_i^{(l)}, \{a_j^{\star(l)}, \partial\ell/\partial e_j^{\star(l)}\})$ in Algorithm D.5 to emphasize that $\Pi^{(l)}$ operates on factorized inputs without materializing the full gradient.

D.4 Compatibility with memory-saving techniques in modern infrastructure

Modern LLM training pipelines employ a range of memory-saving techniques to mitigate the tight memory constraints. Since our methods introduce additional per-sample computational dependencies beyond standard training, a key system question is whether the proposed methods remain compatible with these techniques, including parameter-efficient training, activation checkpointing, and gradient accumulation.

In this subsection, we show that the proposed methods, especially Layer-Wise Subset Update, can be largely integrated with common memory-saving strategies given our system optimizations.

D.4.1 Compatibility with parameter-efficient training

We first consider two widely-used parameter-efficient training methods, LoRA (Hu et al., 2022) and MeSO (Zhao et al., 2024). Both reduce memory consumption by restricting or compressing the trainable update space, and both are naturally compatible with our framework.

LoRA. Recall from Appendix B.1 that LoRA reparametrizes each linear layer as $W_t^{(l)} + B_t^{(l)} A_t^{(l)}$, where $B_t^{(l)} \in \mathbb{R}^{\sqrt{d/L}\times r}$ and $A_t^{(l)} \in \mathbb{R}^{r\times\sqrt{d/L}}$ with rank $r \ll \sqrt{d/L}$, and only the adapter parameters are updated. Since the two LoRA adapters $A_t^{(l)}$ and $B_t^{(l)}$ are themselves linear maps, the per-sample gradients admit the same outer-product factorization as full-parameter gradients: writing $h_{i,\tau}^{(l)} := A_t^{(l)} a_{i,\tau}^{(l)} \in \mathbb{R}^r$ for the adapter's intermediate activation (and absorbing the LoRA scaling factor),

$$G_i^{(A,l)} = \sum_{\tau=1}^{\mathsf{T}} \frac{\partial\ell}{\partial h_{i,\tau}^{(l)}} \otimes a_{i,\tau}^{(l)} \in \mathbb{R}^{r\times d_{\text{in}}^{(l)}}, \qquad G_i^{(B,l)} = \sum_{\tau=1}^{\mathsf{T}} \frac{\partial\ell}{\partial e_{i,\tau}^{(l)}} \otimes h_{i,\tau}^{(l)} \in \mathbb{R}^{d_{\text{out}}^{(l)}\times r}.$$

The layer-wise scoring $s_i^{(l)} = \langle \hat{G}_\star^{(l)}, G_i^{(l)} \rangle$ and gradient computation proceed identically to the full-parameter case, but with the adapter dimension $r(d_{\text{in}}^{(l)} + d_{\text{out}}^{(l)})$ replacing the full dimension $d_{\text{in}}^{(l)} d_{\text{out}}^{(l)}$. This yields proportional reductions in both scoring and gradient computation cost. No modification to the Layer-Wise Subset Update algorithm (Algorithm D.3) is needed; LoRA simply changes the gradient representation that the algorithm operates on.

MeSO. MeSO methods reduce memory in a different way: instead of restricting the trainable parameters, they maintain optimizer states and updates in a compressed subspace (Appendix B.2). This is also compatible with our framework, since data regularization only changes how the update direction is *chosen*, not how the resulting gradient is *applied*. In particular, any scoring method from Appendix D.3 (including the exact methods) can be used to determine $S_{t,p}$, and the optimizer then updates in the projected subspace using only the samples in $S_{t,p}$.

There is also a tighter integration between MeSO and the Compressed Scoring method in Appendix D.3.3. Since both rely on low-dimensional gradient representations, the same compressed per-sample gradients can be used both for score computation and for the optimizer update, avoiding redundant computation. This makes MeSO especially well aligned with our framework: it reduces the memory cost of optimization itself, while also supporting efficient approximate scoring in the same compressed space. We elaborate on the design choices below.

Compression. We adapt Appendix D.3.4 for the computation of the compression required by MeSO. Note that the compressed gradients are trivial to compute by directly applying the forward projection in the factorized form. The tricky part is that when computing the momentum states transfer during subspace refreshing, some of the input does not have the factorized structure. Taking the first moment $\hat{m}_{\text{old}}$ as an example, we see that the transformation requires computing $\Pi_{\text{new}}\Pi_{\text{old}}^\top \hat{m}_{\text{old}}$. We see that for the backward projection, we have an efficient implementation to compute $\Pi_{\text{old}}^\top \hat{m}_{\text{old}}$. However, for the forward projection, it is unclear how to forward project this general input $\Pi_{\text{old}}^\top \hat{m}_{\text{old}}$ under Π_{new}.

However, a simple application of the same trick (formally known as the *vec-trick*), as discussed in the backward projection, resolves this problem and provides a way to compute the forward projection on general input. We omit the details here for simplicity.

Scoring. For exact scoring, this is quite straightforward: we first compute the scores using one of the exact scoring methods discussed in Appendix D.3, discard all the intermediate quantities, solve for $S_{t,p}$, and finally compute the compressed gradients for $S_{t,p}$. Since the exact scoring and compressed gradient computations generally do not have overlapping computations, we do not need to consider retaining any intermediate quantities to avoid recomputation, simplifying the memory management issue.

On the other hand, for approximate scoring, we see that the compressed per-sample gradients that MeSO computes for parameter updates can also be used for compressed scoring (Appendix D.3.3) as an additional optimization, so that the *same* compressed gradients $\widetilde{g}_i^{(l)}, \widetilde{g}_\star^{(l)}$ drive both alignment scoring ($s_i^{(l)} = \langle \widetilde{g}_\star^{(l)}, \widetilde{g}_i^{(l)} \rangle$) and the optimizer update, avoiding redundant computation.

However, in this case, we note that managing memory for the compressed per-sample gradients needed by the optimizer update and compressed scoring is again non-trivial: when there is a group G_p that contains many layers, materializing all the per-sample compressed gradients across all layers may still be memory-intensive and infeasible under a tight memory budget.

Memory management for compressed scoring. We next discuss the memory-management aspect for MeSO with compressed gradients reused. Specifically, assume that we materialize all the $O(n)$ compressed per-sample gradients $\widetilde{g}_i^{(l)}$ and also $\widetilde{g}_\star^{(l)}$ during the backward pass. Once $\widetilde{g}_i^{(l)}$ is computed, the activations $a_i^{(l)}$ and activation gradients $\partial \ell / \partial e_i^{(l)}$ can be released immediately, since the optimizer update only requires averaging the compressed gradients in $S_{t,l}$. This replaces $O(2n\mathsf{T}\sqrt{d/L})$ memory (for $a_i^{(l)}$ and $\partial \ell / \partial e_i^{(l)}$) with $O(n\kappa^{(l)})$ (for $\widetilde{g}_i^{(l)}$), a net savings when $\kappa^{(l)} < \mathsf{T}\sqrt{d/L}$, which holds in typical settings. Hence, it is generally possible to directly retain all the compressed gradients to avoid recomputation of the per-sample compressed gradients for the MeSO updates.

For illustration, the pseudocode for this strategy in the case of Layer-Wise Subset Update is provided in Algorithm D.5, where the compressed gradients are further released at each layer once $S_{t,l}$ is determined, so only one layer's worth of per-sample compressed gradients is held at a time.

Algorithm D.5: Layer-Wise Subset Update with MeSO

Data: Model θ_t, training data $\{z_i\}_{i=1}^n$, target data $\{z_j^\star\}_{j=1}^m$, learning rate η_t, projections Π
Result: Updated model parameters θ_{t+1}

$(\ell, \{a_i^{(l)}\}_{l,i}, \{a_j^{\star(l)}\}_{l,j}) \leftarrow$ `Forward(`θ_t, $\{z_i\}_{i=1}^n \cup \{z_j^\star\}_{j=1}^m$`)`
for $l = L, \dots, 1$ **do**
  $\{\partial \ell / \partial e_i^{(l)}\}_{i=1}^n, \{\partial \ell / \partial e_j^{\star(l)}\}_{j=1}^m \leftarrow$ `Backward(`ℓ, $\theta_t^{(l)}$`)`
  $\widetilde{g}_i^{(l)} \leftarrow \Pi(a_i^{(l)}, \partial \ell / \partial e_i^{(l)}) \in \mathbb{R}^{\kappa^{(l)}}$ for all $i \in [n]$ `// Compress training`
  $\widetilde{g}_j^{\star(l)} \leftarrow \Pi(a_j^{\star(l)}, \partial \ell / \partial e_j^{\star(l)}) \in \mathbb{R}^{\kappa^{(l)}}$ for all $j \in [m]$ `// Compress target`
  `Release(`$a_i^{(l)}$, $\partial \ell / \partial e_i^{(l)}$, $a_j^{\star(l)}$, $\partial \ell / \partial e_j^{\star(l)}$`)` for all i, j
  $\widetilde{g}_\star^{(l)} \leftarrow \frac{1}{m} \sum_{j=1}^m \widetilde{g}_j^{\star(l)}$
  $s_i^{(l)} \leftarrow \langle \widetilde{g}_\star^{(l)}, \widetilde{g}_i^{(l)} \rangle$ for each $i \in [n]$ `// Score via compressed gradients`
  $S_{t,l} \leftarrow$ `Select-S(`$\{s_i^{(l)}\}_{i=1}^n$, k`)`
  $\widetilde{u}_t^{(l)} \leftarrow \frac{1}{k} \sum_{i \in S_{t,l}} \widetilde{g}_i^{(l)}$ `// Aggregate selected`
  `Release(`$\widetilde{g}_i^{(l)}$`)` for all i
  $\theta_{t+1}^{(l)} \leftarrow \theta_t^{(l)} - \eta_t \cdot (\Pi^\top \widetilde{u}_t^{(l)})$ `// Back-project and update`
  `Release(`$\widetilde{u}_t^{(l)}$`)`
return θ_{t+1}

D.4.2 COMPATIBILITY WITH MEMORY-SAVING SCHEDULING TECHNIQUES

We now turn to two techniques that reduce memory by changing the temporal structure of training: activation checkpointing and gradient accumulation. Unlike LoRA and MeSO, these methods affect *when* intermediate tensors are available, and therefore interact more directly with our tensor lifetime schedule.

Activation checkpointing. Activation checkpointing reduces memory by discarding some forward activations and recomputing them on demand during backward. Our one-pass tensor lifetime schedule is compatible with checkpointing whenever the parameter groups align with the checkpointing schedule, so that all computations associated with a group can be resolved within a checkpoint segment. Layer-Wise Subset Update satisfies this condition naturally, since each layer forms its own group and can therefore be resolved within its own backward step.

When a group spans multiple checkpoint segments, however, the one-pass schedule can no longer preserve all required local tensors across segment boundaries without defeating the purpose of checkpointing. In this case, one must revert to the two-pass implementation discussed in Appendix D.2.4: the first pass computes scores and determines the subsets, and the second pass assembles the update (see Appendix D.2.4). Thus, the compatibility with checkpointing depends on whether the partition granularity is aligned with the memory-saving schedule.

Gradient accumulation. Gradient accumulation reduces peak memory by splitting a large batch into micro-batches that are processed sequentially, and the activations and pre-activation gradients associated with an earlier micro-batch are typically released before the next one is processed for saving memory. This conflicts with our one-pass implementation, which relies on retaining the pairs $(a_i^{(l)}, \partial \ell / \partial e_i^{(l)})$ for all samples in the full batch in order to compute scores and assemble the final update. When the rule for determining $S_{t,p}$ decomposes across micro-batches, however, this issue resolves: each micro-batch can be scored, solved, and assembled independently during its own forward and backward pass, and the resulting updates can then be accumulated across micro-batches. Thresholding is one such example. In our RL experiments each micro-batch is scored, filtered, and normalized by its own retained item count, and the micro-batch gradients are then averaged with the standard accumulation weights; the update is therefore the average of per-micro-batch retained means rather than the mean over all retained samples of the mini-batch, mirroring the per-micro-batch normalization that standard training uses in the same pipeline. In contrast, rules such as top-k depend on statistics computed over the entire batch, so the micro-batches cannot be resolved independently. In this case, we again resort to the two-pass implementation described in Appendix D.2.4.

D.5 BEYOND LINEAR LAYERS

The system design above develops scoring and selection exclusively for linear layers (e.g., `nn.Linear` in `PyTorch`), which account for the vast majority of trainable parameters in standard transformer architectures. A transformer model, however, also contains other trainable modules—token embeddings (`nn.Embedding`) and normalization layers (`nn.LayerNorm`, `nn.RMSNorm`)—whose treatment we now discuss.

D.5.1 EMBEDDING LAYERS

The token embedding layer maps each input token index $x_{i,\tau} \in \mathcal{V}$ to a dense vector via a lookup: $\mathrm{output}_{i,\tau} = W[x_{i,\tau}]$, where $W \in \mathbb{R}^{V \times d_{\mathrm{emb}}}$ is the embedding matrix, $\mathcal{V}$ is the vocabulary, and d_{emb} is the embedding dimension. Unlike a linear layer, whose per-sample gradient is a sum of dense outer products, the embedding gradient is *sparse*: writing $\delta_{i,\tau} \coloneqq \partial \ell_i / \partial \mathrm{output}_{i,\tau} \in \mathbb{R}^{d_{\mathrm{emb}}}$ for the activation gradient at position τ, the per-sample weight gradient is

$$G_i = \sum_{\tau=1}^{\mathsf{T}} e_{x_{i,\tau}} \otimes \delta_{i,\tau} \in \mathbb{R}^{V \times d_{\mathrm{emb}}},$$

where $e_{x_{i,\tau}} \in \mathbb{R}^{|\mathcal{V}|}$ is the one-hot indicator for token $x_{i,\tau}$. Row v of G_i equals $\sum_{\tau:\, x_{i,\tau}=v} \delta_{i,\tau}$, and all other rows are zero.

Efficient scoring via lookup. This sparsity enables a scoring algorithm that avoids materializing any per-sample gradient matrix. The target embedding gradient is

$$\hat{G}_\star = \frac{1}{m}\sum_{j=1}^{m}\sum_{\tau=1}^{\mathsf{T}} e_{x^\star_{j,\tau}} \otimes \delta^\star_{j,\tau} \in \mathbb{R}^{V\times d_{\text{emb}}},$$

computable in $O(m\mathsf{T}d_{\text{emb}})$ via `index_add` (scatter-accumulate over token indices). The per-sample alignment score then simplifies as follows:

$$s_i^{(\text{emb})} = \langle G_i, \hat{G}_\star\rangle = \sum_{v\in\mathcal{V}}\langle G_i[v], \hat{G}_\star[v]\rangle = \sum_{\tau=1}^{\mathsf{T}}\langle \delta_{i,\tau}, \hat{G}_\star[x_{i,\tau}]\rangle,$$

where the last equality follows from $G_i[v] \neq 0$ only for $v \in \{x_{i,\tau}\}_\tau$. Each term is a table lookup of $\hat{G}_\star$ at index $x_{i,\tau}$ followed by a dot product with $\delta_{i,\tau}$, both $O(d_{\text{emb}})$ operations. Summing over T tokens gives $O(\mathsf{T}d_{\text{emb}})$ per sample, and $O(n\mathsf{T}d_{\text{emb}})$ across all n training samples. Including the cost of constructing $\hat{G}_\star$, the total is $O(N\mathsf{T}d_{\text{emb}})$ FLOPs with $O(VD)$ memory for storing $\hat{G}_\star$.

Comparison with linear layer scoring. The embedding scoring algorithm is the natural analogue of the Per-Token Inner Product (PIP, Appendix D.3.2) for linear layers. For a linear layer, PIP evaluates $s_i^{(l)} = \sum_\tau (\partial\ell/\partial e_{i,\tau}^{(l)})^\top \hat{G}_\star^{(l)} a_{i,\tau}^{(l)}$, which requires a matrix-vector product $\hat{G}_\star^{(l)} a_{i,\tau}^{(l)} \in \mathbb{R}^{\sqrt{d/L}}$ at each token. For the embedding layer, the "input" is a one-hot vector $e_{x_{i,\tau}}$, so the matrix-vector product $\hat{G}_\star\, e_{x_{i,\tau}} = \hat{G}_\star[x_{i,\tau}]$ reduces to a single row lookup, eliminating the $O(d_{\text{emb}})$ factor from the matrix-vector multiply. In practice, the activation gradients $\delta_{i,\tau}$ are already available during backpropagation (they are the upstream gradient flowing into the embedding layer), so scoring adds only the table-lookup and dot-product cost per token.

Gradient assembly. After selection, the weight gradient for the selected subset S is assembled as $G^{(\text{emb})} = \frac{1}{|S|}\sum_{i\in S} G_i$, which is a sparse accumulation computable in $O(|S|\mathsf{T}d_{\text{emb}})$ via `index_add`, matching the cost of standard embedding gradient computation on a batch of size $|S|$.

D.5.2 Normalization layers

Normalization layers (e.g., `LayerNorm` (Ba et al., 2016), `RMSNorm` (Zhang & Sennrich, 2019)) contain a small number of trainable parameters—a scale vector $\gamma \in \mathbb{R}^{d_{\text{emb}}}$ and optionally a bias $b \in \mathbb{R}^{d_{\text{emb}}}$—contributing $O(d_{\text{emb}})$ parameters per normalization layer versus $O(d_{\text{emb}}^2)$ for each linear layer. These parameters are not scored or selected; they receive the gradient of the training batch only, i.e., they are trained under Full-Training Update ($U = \{\hat{g}_{\text{tr}}\}$), identically across the compared methods. In the merged-batch one-pass schedule (our SFT experiments) a custom backward restricts their weight gradient to the n training samples of the merged batch, excluding the m target samples; in the RL pipelines the target gradient is computed in a separate pass, so the target samples never enter their gradient. In a typical transformer, the total parameter count of all normalization layers is negligible relative to that of the linear layers: in QWEN3-1.7B-BASE they amount to about 1.2×10^5 of 1.7×10^9 trainable parameters (below 0.01%), and in the LoRA settings every trainable parameter belongs to an adapter of a hooked linear layer.

Summary. Every trainable parameter in our experiments is handled by exactly one of two rules: it is either *hooked*, i.e., scored and updated from its selected subset, or it is updated with the plain training-batch gradient (Full-Training Update); our implementation enumerates the trainable parameters at setup so that none is left unaccounted for. The hooked modules are all `nn.Linear` modules of the transformer blocks and, in the full-parameter and MeSO settings, the embedding matrix (scored with the lookup-based algorithm above) and the `lm_head`. In QWEN3-1.7B-BASE the two are weight-tied; we treat the two uses as separate groups, each selecting independently as any other pair of groups, and sum their selected gradients into the shared matrix; the feasible set of the shared weight is then the sum of the two group feasible sets, which contains the single-subset set, so the bias ordering of Theorem C.6, which uses only this inclusion and not the per-group separability of Appendix C.3, is preserved, and under Global Subset Update both uses share one subset so the update reduces to the standard tied gradient. In the LoRA settings every trainable parameter is an adapter of a hooked linear layer, and the embedding and normalization layers are frozen. Updated with the training-batch gradient are the normalization layers (Appendix D.5.2) and the value head of

the RLHF policy, which is not scored and is updated with every rollout (Appendix E.2). Together these two classes account for all trainable parameters, and the target samples never enter the gradient of an unhooked parameter.

E Omitted details from Section 4

In this section, we provide all the omitted details for all experiment settings, including SFT, RLHF, and RLVR. We note that all reported results in the main text are averaged over 5 random seeds with error bars given as standard errors of the mean, and evaluation is performed on held-out test sets disjoint from the data used for regularization or training. Unless stated otherwise, every curated arm uses exact scoring (PIP, or GIP in the question-answering setting; the two return the same inner products); compressed scoring appears only in the ablations of Appendices E.1.3 and E.2.5.

Compared methods. Our experiments ask how a small target set should guide updates supported by a much larger general training pool. We therefore compare Full-Training Update, i.e., standard training on the general pool, with the two main data-regularized updates, Global Subset Update and Layer-Wise Subset Update, under the same models, data, optimizers, schedules, and step budgets; the two curated updates share the selection rule (top-k in SFT, negative score filtering in RL), the scoring backend, and all hyperparameters, and differ only in the partition.

Scope: We study post-training from the data-centric side: given a small target set, which examples of a much larger general pool an update should draw on, at which granularity, and why. The compared Methods therefore differ only in how they use the pool: Full-Training Update uses it as is, Global and Layer-Wise Subset Update reweight it under one shared or many per-layer rankings, and Target-Only Update uses none of it and marks the pool-free end of the spectrum. We do not cover every way to utilize the scarce target data. Alternatives that change the training schedule rather than the use of the pool, such as mixing the target examples into the pool, are orthogonal to data regularization and can follow any of the arms above; combining them with data-regularized updates is a promising direction that we leave to future work.

Table 6 lists the model, fine-tuning method, scoring backend, selection rule, and target set of every setting, with pointers to its results; the details follow in the respective subsections. As baselines we take the state-of-the-art online selection method of each paradigm: GREATS (Wang et al., 2024a), which outperforms loss-based (Loshchilov & Hutter, 2015), gradient-norm (Katharopoulos & Fleuret, 2018), reference-model (Mindermann et al., 2022), and embedding-similarity (Reimers & Gurevych, 2019) selection on the general/target pairs we reuse, and IIF (Hu et al., 2025b) in RLHF. Global Subset Update with top-k is the first-order variant of GREATS; in the question-answering settings, GREATS' redundancy-aware greedy construction selected nearly the same subsets as top-k, so all curated arms use first-order alignment scores (Appendix C.2).[5]

Computing resources. Our experiments are conducted on a shared academic GPU cluster. SFT, RLHF, and benchmark experiments run on nodes with four `A40` GPUs (48 GB each), and RLVR experiments on nodes with eight `H200` GPUs (141 GB each).

E.1 Supervised fine-tuning

We now provide the missing details of the SFT experiments in Section 4.1, following its two stages: the question-answering experiments with Llama-3.2-1B (Section 4.1.1), including the case study, and the reasoning and instruction-following experiments with Qwen3-1.7B-Base (Section 4.1.2).

E.1.1 Question answering with Llama-3.2-1B

Datasets. We consider four general/target pairs:

1. `alpaca`[6] (Taori et al., 2023) (CC BY-NC 4.0 license)/`samsum`[7] (Gliwa et al., 2019) (CC BY-NC-ND 4.0 license). Single-turn instruction-following/dialogue summarization. We randomly sample 40% of the `alpaca` training pool per seed ($\sim$20.8k examples).

[5] GREATS itself validates this term through the fidelity of its Taylor approximation rather than through a downstream comparison with first-order selection.

[6] https://huggingface.co/datasets/tatsu-lab/alpaca

[7] https://huggingface.co/datasets/knkarthick/samsum

Table 6: Overview of the experimental settings: model and trainer, fine-tuning method, scoring backend, selection rule, target set, and where the results are reported (main text; appendix).

	Question answering Section 4.1.1; Appendix E.1.1	**Reasoning and IF** Section 4.1.2; Appendix E.1.4	**RLHF** Section 4.2; Appendix E.2	**RLVR** Section 4.3; Appendix E.3
Model	LLAMA-3.2-1B	QWEN3-1.7B-BASE	GPT-NEO-2.7B	QWEN3-1.7B-BASE
Trainer	SFT	SFT	PPO (`TRL`)	GRPO (`Verl`)
Fine-tuning	Full, LoRA, MeSO	Full parameter	LoRA	Full parameter
Scoring	Exact (GIP), compressed	Exact (PIP)	Exact (PIP), compressed	Exact (PIP)
Selection	Top-k, $k = 4$ of 8	Top-k, $k \in \{2, 4, 6\}$ of 8	Negative score filtering	Negative score filtering
Target set	16 held-out examples	32 held-out examples (16–128 in the ablation)	Self-referencing rollouts, or 1024 held-out prompts; reward or PPO loss as signal	Training prompts; verifiable reward as signal
Target gradient	$m = 1$ example, resampled every step	$m = 1$ example, resampled every step	256 prompts per PPO step, captured once per rollout batch and reused across its mini-batches	64 prompts $\times$ 8 rollouts, refreshed every fourth round and reused in between
Additional baselines	GREATS, Random, Target-Only	Random	IIF	–
Main text	Table 2	Table 3	Figure 2	Figure 3
Appendix	Tables 7, 9 and 10 and Figs. 8 and 9	Tables 11 and 12	Figures 10 and 11 and Table 13	Table 14

2. `less-mix` (Xia et al., 2024a)/`tydiqa`[8] (Clark et al., 2020) (Apache-2.0 license). Heterogeneous instruction mixture/multilingual extractive QA. We randomly sample 0.5% of `less-mix` per seed ($\sim$9.8k examples).
3. `triviaqa`[9] (Joshi et al., 2017)/`nq_open`[10] (Kwiatkowski et al., 2019) (CC BY-SA 3.0 license). Closed-book TriviaQA QA pairs (`rc.nocontext`, $\sim$ 138k examples)/closed-book Natural Questions evaluation. We randomly sample 5% of the `triviaqa` training pool per seed ($\sim$6.9k examples).
4. `less-mix` (Xia et al., 2024a)/`squad`[11] (Rajpurkar et al., 2016) (CC BY-SA 4.0 license). Same training pool as the `tydiqa` setting, with the target task replaced by SQuAD reading-comprehension answers without context. 0.5% sampling rate as above.

For each target task we hold out 16 samples for data regularization and 500 held-out samples for in-training perplexity tracking, on which we also report the final-checkpoint downstream metric (Rouge-L for `samsum`; F1 for `tydiqa`, `nq_open`, and `squad`). LLAMA-3.2-1B ships without a chat template, so we install an `open-instruct`-style fallback ($\langle$user$\rangle$/$\langle$assistant$\rangle$ plaintext markers) before tokenization, with the loss computed only on assistant-content tokens.

Training configurations. All models are trained with standard supervised fine-tuning using next-token prediction on formatted instruction–response pairs. Across the four settings, we report results for (i) full-parameter fine-tuning, (ii) LoRA fine-tuning, and (iii) MeSO; All three fine-tuning methods are reported on all four settings (Appendix E.1.3). All methods use the same number of optimization steps within each dataset pair and the same effective batch size. We use AdamW (Loshchilov & Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, weight decay 0, a linear learning-rate schedule with 3% warmup, batch size 8, maximum sequence length 512, and train for 1 epoch. Training uses `bfloat16` precision and FlashAttention-2. For the learning rate, we follow Grattafiori et al. (2024), where we use 10^{-5} for full-parameter fine-tuning, 10^{-4} for LoRA (a $10\times$ scaling (Schulman & Lab, 2025)), and 5×10^{-5} for MeSO.

The number of training steps is determined by the 1-epoch budget on the sampled training pool: $\sim$2600 for `alpaca`/`samsum`, $\sim$1225 for `less-mix`/`tydiqa` and `less-mix`/`squad`, and $\sim$865 for `triviaqa`/`nq_open`. We log evaluation perplexity at $\sim$100 equally spaced steps across the trajectory (80 for `triviaqa`/`nq_open`).

For LoRA, we use rank $r = 8$, scaling factor $\alpha = 16$, dropout 0.1, and apply adapters to all linear layers (the full Q/K/V/O attention projections and the gate/up/down MLP projections). For

[8] https://huggingface.co/datasets/google-research-datasets/tydiqa
[9] https://huggingface.co/datasets/mandarjoshi/trivia_qa
[10] https://huggingface.co/datasets/google-research-datasets/nq_open
[11] https://huggingface.co/datasets/rajpurkar/squad

MeSO, we use the gradient compressor described in Appendix D.3.4 with a Gaussian projector for both $P_{\text{out}}^{(l)}$ and $P_{\text{in}}^{(l)}$ of dimension 512 (i.e., the total per-layer compression dimension is $\kappa^{(l)} = 512 \times 512$), and no second-stage projector ($P_{\text{final}}^{(l)} = I$), refreshing the subspace every 200 steps; the tied input/output embedding matrix and the normalization weights are updated by standard AdamW without compression.

Data regularization. Data regularization is guided by a small held-out target set of 16 samples. At each training step, we form a merged batch of $n = 8$ training samples and $m = 1$ target sample (resampled per step from the 16 samples held-out target pool), and score the candidate training examples either exactly (GIP, Appendix D.3.1) or with compressed scoring (Appendix D.3.3, $\kappa^{(l)} = 64 \times 64$); both backends are run for every curated arm under all three fine-tuning methods (Appendix E.1.3), and the main-text results use exact scoring.

Both curated updates in Section 4.1.1 solve for S_t (or $S_{t,l}$ in the case of Layer-Wise Subset Update) by the top-k strategy with a retaining rate of 50%, i.e., $k = n/2 = 4$ of the $n = 8$ candidate training examples per step. Global Subset Update with top-k selection is the first-order variant of the online data selection baseline GREATS (Wang et al., 2024a), which introduced the first two general/target pairs and whose greedy construction additionally accounts for the redundancy penalty (Appendix C.2); we use the same top-k rule for both curated arms so that they differ only in partition granularity. Target-Only Update trains on the 16 target examples alone, with the learning rate and schedule of Full-Training, and is reported under two protocols: with the step budget of the other arms (*same budget*) and stopped after 50 steps (*early stop*, the protocol of Table 2). For a given seed these are exactly the target examples the curated arms score against.

Results. Table 2 in the main text reports the LoRA runs with exact scoring, Figure 8 shows their evaluation perplexity on the 500 held-out examples over training, and Table 7 reports the final evaluation perplexity of the three main update rules and of Target-Only Update under every fine-tuning method.

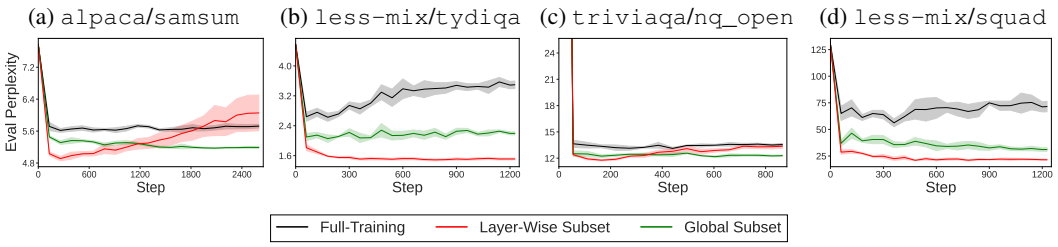

Figure 8: Question-answering SFT (LoRA, exact scoring): evaluation perplexity on the 500 held-out examples over training, mean over 5 seeds with standard-error bands.

Two points stand out. First, the ordering Layer-Wise < Global < Full-Training in perplexity holds on nearly every setting/fine-tuning pair: Global Subset Update ends below Full-Training on all pairs but one, and Layer-Wise Subset Update ends below Global Subset Update on all pairs but two. The gaps are widest under the broad `less-mix` pool, whose general distribution is most misaligned with the targets, and narrowest on `triviaqa/nq_open`, whose general pool is already close to the target. Second, no method is tuned: every arm runs with the learning rate, schedule, and step budget of Full-Training, although their training dynamics differ. In Figure 8, Layer-Wise Subset Update falls below the other arms within the first few hundred steps on every setting, and on `alpaca/samsum` and `triviaqa/nq_open` under LoRA it then overfits: its held-out perplexity reaches its minimum, below the minimum of every other arm, after about a fifth of the budget and climbs for the rest of training until it ends above Global Subset Update, while its perplexity on the 16 target examples keeps falling. These two LoRA pairs are two of the three exceptions to the ordering above; under full-parameter fine-tuning the same arm ends close to its minimum and the ordering is intact. The third exception is `triviaqa/nq_open` under MeSO, where every arm ends well above its counterparts under the other two fine-tuning methods and Full-Training ends lowest, the price of the compressed optimizer update at the larger step size. The least regularized arm thus converges fastest and would benefit most from a shorter budget or a smaller step size, so the gains we report for it are conservative.

Target-Only Update. Under the shared budget, one pass over the pool and hence hundreds of passes over the target examples, the pool-free arm behaves as Section 2.3 predicts for a small target

set: its held-out perplexity is minimal within the first few dozen steps and diverges as the examples are memorized, ending far above every other arm on every target but `less-mix/tydiqa` (Table 7). Nothing regularizes an arm without a pool, hence the early-stopped protocol, an oracle stop near that minimum that favors Target-Only Update; even so it ends above both curated updates on three of the four targets. The downstream metric adds what the loss does not isolate, the style of the answer. On `samsum` and `tydiqa` the content is in the prompt and the references are short summaries or spans, whereas the pools answer in paragraphs, so style decides the metric; the target examples fix it within the first steps and greedy decoding keeps it after the perplexity has diverged, so Target-Only Update is ahead of or level with the best pool-based arm under LoRA and full-parameter fine-tuning (under MeSO, whose compressed update learns more slowly, the early stop is too short on `tydiqa`), and the curated updates recover part of the gap by keeping the shortest pool examples. On the closed-book targets the metric depends on knowledge that the target examples cannot supply and that memorizing them erodes, so it never comes first under either protocol: style is what a handful of examples can teach, knowledge is not.

Table 7: Final evaluation perplexity on the 500 held-out examples of the three main update rules and Target-Only Update in the question-answering experiments.

Setting	Fine-tuning	Full-Training	Global Subset	Layer-Wise Subset	Target-Only (same budget)	Target-Only (early stop)
`alpaca/samsum`	Full parameter	5.67 ±0.02	5.30 ±0.02	**4.80 ±0.03**	62.42 ±5.16	21.36 ±5.34
	LoRA	5.72 ±0.05	**5.19 ±0.02**	6.06 ±0.45	289.66 ±21.36	12.21 ±0.66
	MeSO	6.69 ±0.08	6.53 ±0.09	**6.20 ±0.15**	89.51 ±8.35	64.01 ±11.58
`less-mix/tydiqa`	Full parameter	3.30 ±0.03	2.31 ±0.04	**1.72 ±0.03**	1.68 ±0.04	1.78 ±0.03
	LoRA	3.50 ±0.10	2.19 ±0.04	**1.51 ±0.03**	1.85 ±0.06	1.75 ±0.07
	MeSO	2.83 ±0.06	2.55 ±0.38	**1.63 ±0.02**	1.98 ±0.07	2.84 ±0.15
`triviaqa/nq_open`	Full parameter	13.40 ±0.06	12.37 ±0.07	**11.98 ±0.09**	77.55 ±8.35	36.98 ±3.01
	LoRA	13.55 ±0.13	**12.29 ±0.09**	13.36 ±0.25	168.40 ±24.93	32.92 ±2.51
	MeSO	**18.91 ±0.18**	20.19 ±0.22	19.40 ±0.37	246.82 ±39.75	160.66 ±21.85
`less-mix/squad`	Full parameter	81.78 ±3.40	42.53 ±2.29	**27.06 ±1.23**	109.41 ±5.82	55.45 ±6.05
	LoRA	71.49 ±4.90	31.11 ±1.69	**21.47 ±0.57**	463.25 ±84.54	48.10 ±5.45
	MeSO	69.99 ±5.42	48.25 ±5.47	**35.27 ±1.38**	434.70 ±76.23	231.68 ±13.29

E.1.2 CASE STUDY

We record per-layer scores during a Full-Training Update across all four general/target pairs. At each training step, we separately score the same batch of $n = 8$ samples as in Layer-Wise Subset Update and Global Subset Update; to remove score compression as a confounder, the recorded scores are exact PIP scores (Appendix D.3.2). We compute

(i) **mean absolute score** per layer (magnitude), and

(ii) **Spearman rank correlation** ρ_S between each layer's per-sample scores and the subset's global ranking.

We aggregate the results by *transformer blocks* and *layer types*: each transformer block contains seven linear layers, namely four attention projections (query (Q), key (K), value (V), and output (O)) and three MLP projections (gate, up, and down).

The first column of Figure 9 shows the per-layer scores on the `alpaca/samsum` setting. The layers disagree about which candidates to keep (bottom panel): the ranking of `down_proj` agrees with the global ranking ($\rho_S = 0.88$ at its peak), whereas the rankings of the Q and K projections are nearly uncorrelated with it ($\rho_S = 0.12$ and 0.11 averaged over blocks, and at most 0.24 at any block), so the two groups of layers prefer different samples. Which of them the shared ranking follows is decided by magnitude (top panel): `down_proj` at block 1 produces scores $32\times$ larger than the second-largest layer type at that block, the V projection, and the K projection, which shares V's input–output shape (512×2048), is smaller still by more than an order of magnitude, so the sum over layers follows `down_proj`. Under Global Subset Update these 32 layers (out of 112 block-level linear layers) therefore train on samples chosen for `down_proj`; Layer-Wise Subset Update lets each layer follow its own ranking.

The remaining columns of Figure 9 extend the score analysis to the three QA settings. The same qualitative pattern holds across all four settings: Q and K projections remain weakly correlated with the global subset ranking ($\rho_S \leq 0.22$ averaged over blocks), while a single `down_proj` layer at an early block (block 0 or 1) produces scores $22\times$–$38\times$ larger than the second-largest layer type at the same block, again the V projection, and thereby decides the shared ranking ($\rho_S = 0.90$–0.94 at its

peak). The disagreement between layers is thus not specific to any particular general/target pair or training-pool composition. Whether rescaling the layers' scores suffices to repair the shared ranking is tested in Appendix E.1.3.

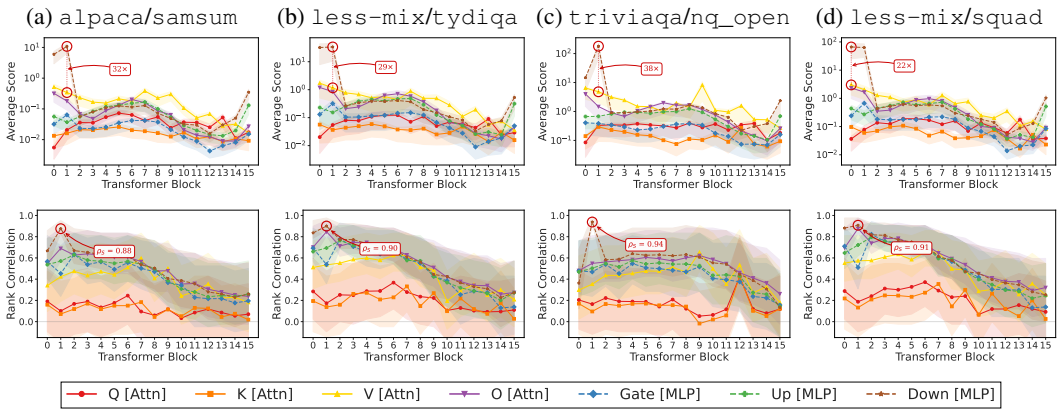

Figure 9: Case study on the four general/target pairs (columns): per-layer alignment scores recorded during Full-Training Update, aggregated by transformer block and layer type; lines are means over training steps and seeds, bands the 25th–75th percentiles. **Top:** mean absolute score (log scale); the annotation marks the `down_proj` peak and its ratio to the second-largest layer type at that block. **Bottom:** Spearman rank correlation ρ_S between each layer's per-sample scores and the shared ranking of Global Subset Update; the annotation marks ρ_S at the `down_proj` peak.

E.1.3 Ablations on the question-answering setting

Table 2 compares the two endpoints of the group-wise family, the global partition and the per-layer partition, under LoRA with exact scoring. We conduct five ablations on the same four question-answering settings: the Random Subset controls, the intermediate partitions of Section 2.2.2, a layer-normalized variant of Global Subset Update, the fine-tuning method, and the scoring backend. Unless stated otherwise, every arm follows the protocol of Appendix E.1.1 unchanged: the same pools, target sets of 16 examples, $n = 8$, $m = 1$, $k = n/2$, learning rate, schedule, and step budget, evaluated at the final checkpoint on the 500 held-out examples over 5 seeds. Table 9 reports every partition under both scoring backends for LoRA, full-parameter fine-tuning, and MeSO, and Table 10 the controls.

- *Random Subset controls.* Random Subset Update draws a uniformly random subset of $k = 4$ candidates at every step and trains on it through the same update path as the curated arm it mirrors, so the two differ only in how the subset is chosen: Global Random Subset draws one subset per step that all layers share, and Layer-Wise Random Subset draws an independent subset for every layer.
- *Partition granularity.* Table 8 lists the four partitions and their group counts, which depend on which layers are scored. Under LoRA the scored layers are the 224 adapter factors, the A and B matrices of each of the 112 projections, so Layer-Wise Subset Update selects a separate subset for the two factors of one projection, and the coarser partitions group all factors of a transformer block or of an attention/MLP sublayer. Under full-parameter fine-tuning and MeSO the scored layers are the 112 projections together with the token embedding and the output head, which lie outside the transformer blocks and are singleton groups under every partition except the global one. Each group sums the scores of its layers and keeps its own top-k subset through the same single-pass update as the layer-wise arm (Section 2.2.2).
- *Layer-normalized scores.* The case study (Appendix E.1.2) admits two readings of why the shared ranking fails: a few layers dominate it by the magnitude of their scores, or the layers' rankings disagree. The first reading suggests a cheaper fix than the group-wise relaxation, so we test a layer-normalized variant of Global Subset Update that divides each layer's score vector by its mean absolute value before the vectors are summed into the shared score, so that no layer can dominate the shared ranking by magnitude alone. Under the second reading, rescaling only changes

which layers decide the shared ranking and still keeps one subset for all layers, so it cannot serve layers whose rankings disagree, whereas the group-wise update follows each layer's own ranking. This variant and the controls are scored with compressed scoring, so Table 10 lists the compressed-scoring rows of the two endpoints next to them.

- *Fine-tuning methods.* Every arm, including Target-Only Update, is run with full-parameter fine-tuning, LoRA, and MeSO on all four settings, with the learning rates of Appendix E.1.1; the fine-tuning method changes which parameters are updated and how, whereas scoring and selection are unchanged.
- *Scoring backend.* Every curated arm is scored both exactly and with compressed scoring at $\kappa^{(l)} = 64 \times 64$, under all three fine-tuning methods.

Table 8: Partitions of the question-answering ablation and their number of groups P on LLAMA-3.2-1B under each fine-tuning method. Under LoRA the scored layers are the adapter factors, so the two factors of one projection are separate layers; under full-parameter fine-tuning and MeSO the token embedding and the output head are singleton groups under every partition except the global one.

Partition	**One group per**	**Number of groups P**	
		LoRA (224 adapter factors)	Full parameter, MeSO (112 projections, embedding, output head)
Global Subset	all scored layers	1	1
Block-Wise Subset	transformer block	16 (14 factors each)	18 (16 blocks, 2 singletons)
Sublayer-Wise Subset	attention or MLP sublayer	32 (8 or 6 factors each)	34 (32 sublayers, 2 singletons)
Layer-Wise Subset	scored layer	224	114

Results. Tables 9 and 10 support five observations, one per ablation.

- *Random Subset controls.* Discarding half of every batch at random does not reproduce the gains: both controls stay within a couple of points of Full-Training on every setting and fine-tuning method, above it as often as below, whether the subset is shared or drawn per layer. The improvements of the curated arms come from the steering by the target scores, not from the smaller effective batch.
- *Partition granularity.* With exact scoring, wherever the arms separate beyond their standard errors the order is global $<$ block $<$ sublayer $<$ layer, most clearly under the misaligned `less-mix` pool; on the closed-book settings the four partitions lie within a point or two of one another, with the layer-wise endpoint still on top under full-parameter fine-tuning and LoRA. This matches Theorem 2.2: refining the partition can only lower the bias terms.
- *Layer-normalized scores.* Rescaling the layers' scores does not close the gap: layer normalization moves Global Subset Update by several points at most, closing part but never most of its distance to the layer-wise endpoint on the misaligned settings, and changes the closed-book settings little. This supports the second reading above: rebalancing the magnitudes changes which layers decide the shared ranking, but one shared subset still cannot serve layers whose rankings disagree.
- *Fine-tuning methods.* The ordering does not depend on the fine-tuning method: under full-parameter fine-tuning as under LoRA, Layer-Wise Subset Update with exact scoring is the best pool-based arm on every setting, and Global Subset Update improves over Full-Training on `less-mix`/`tydiqa` and stays level with it elsewhere. Under MeSO the picture is mixed (Table 9(c)): Layer-Wise Subset Update with exact scoring remains the best arm on `alpaca`/`samsum` and `less-mix`/`tydiqa`, but on the two closed-book settings no curated arm improves over Full-Training, and every MeSO arm trails its full-parameter counterpart there. MeSO applies each layer's update through a 512×512 sketch of its gradient, so a curated subset steers a noisier update, and the downstream cost is largest where the gains of curation are smallest; we report MeSO as one of the fine-tuning methods the framework composes with, not as a recommendation.
- *Scoring backend.* The finer the partition, the more it depends on exact scores. Global Subset Update is insensitive to the backend (exact and compressed agree within one to two standard errors everywhere), whereas Layer-Wise Subset Update loses up to several points under compressed scoring on the misaligned settings, most under full-parameter fine-tuning on `less-mix`/`tydiqa`, where the loss reaches double digits, and little elsewhere, with the intermediate partitions in between. The cause is per-layer sketch noise rather than bias: on 26 identical `alpaca`/`samsum` batches, compressed per-layer scores correlate with exact ones at 0.89 (top-4 overlap 0.77) at 64×64 and 0.99 (0.89) at 512×512, and summing the layers as Global Subset Update does

Table 9: Partition granularity and scoring backend on the question-answering settings under the three fine-tuning methods: downstream metric (%) at the final checkpoint. Exact scoring is GIP, compressed scoring uses $\kappa^{(l)} = 64 \times 64$, and Full-Training and Target-Only Update do not score; the exact-scoring Full-Training, Global Subset, and Layer-Wise Subset rows of Table 9(a) are those of Table 2.

(a) LoRA.

Method	alpaca/samsum		less-mix/tydiqa		triviaqa/nq_open		less-mix/squad	
	Exact	Compressed	Exact	Compressed	Exact	Compressed	Exact	Compressed
Full-Training	22.4 ±0.3		4.9 ±0.3		12.7 ±0.2		9.0 ±0.2	
Global Subset	22.7 ±0.5	22.3 ±0.2	13.7 ±1.5	12.8 ±1.5	13.5 ±0.3	**13.8 ±0.2**	9.1 ±0.2	9.0 ±0.2
Block-Wise Subset	25.9 ±0.8	23.0 ±0.8	33.5 ±2.6	25.8 ±3.0	13.7 ±0.2	13.2 ±0.4	8.6 ±0.2	8.3 ±0.4
Sublayer-Wise Subset	28.3 ±1.1	24.7 ±1.1	42.2 ±1.1	29.0 ±2.4	13.2 ±0.2	13.2 ±0.3	8.8 ±0.2	8.4 ±0.4
Layer-Wise Subset	**29.3 ±1.4**	**29.2 ±0.7**	**48.3 ±1.7**	**46.9 ±0.6**	**14.0 ±0.3**	13.3 ±0.4	**10.0 ±0.4**	**9.5 ±0.3**
Target-Only (same budget)	30.9 ±0.3		51.8 ±3.7		7.0 ±0.5		5.5 ±0.4	
Target-Only (early stop)	32.8 ±0.6		51.1 ±2.9		9.3 ±0.4		8.7 ±0.4	

(b) Full-parameter fine-tuning.

Method	alpaca/samsum		less-mix/tydiqa		triviaqa/nq_open		less-mix/squad	
	Exact	Compressed	Exact	Compressed	Exact	Compressed	Exact	Compressed
Full-Training	20.4 ±0.2		3.5 ±0.2		13.2 ±0.1		8.4 ±0.1	
Global Subset	20.9 ±0.7	20.1 ±0.7	8.8 ±1.6	6.4 ±0.8	13.6 ±0.3	13.1 ±0.2	8.5 ±0.2	8.4 ±0.1
Block-Wise Subset	24.6 ±1.2	23.7 ±0.8	18.7 ±1.7	11.5 ±0.6	13.7 ±0.3	13.7 ±0.2	8.5 ±0.1	8.6 ±0.1
Sublayer-Wise Subset	26.3 ±0.9	23.6 ±1.0	22.6 ±1.4	**16.9 ±1.8**	14.1 ±0.2	13.5 ±0.1	8.6 ±0.2	8.4 ±0.1
Layer-Wise Subset	**26.9 ±0.6**	**24.1 ±1.0**	**32.7 ±1.8**	16.3 ±2.2	**14.2 ±0.3**	**13.8 ±0.2**	**9.0 ±0.2**	**8.9 ±0.1**
Target-Only (same budget)	30.8 ±0.3		56.2 ±2.2		8.2 ±0.7		8.6 ±0.3	
Target-Only (early stop)	32.1 ±0.3		49.1 ±1.8		10.1 ±0.5		8.8 ±0.3	

(c) MeSO.

Method	alpaca/samsum		less-mix/tydiqa		triviaqa/nq_open		less-mix/squad	
	Exact	Compressed	Exact	Compressed	Exact	Compressed	Exact	Compressed
Full-Training	19.3 ±0.7		4.2 ±0.2		**11.7 ±0.3**		**7.4 ±0.2**	
Global Subset	20.1 ±0.9	19.6 ±0.6	11.1 ±4.0	8.9 ±2.3	9.7 ±0.5	9.0 ±1.7	6.7 ±0.2	6.9 ±0.2
Block-Wise Subset	25.0 ±0.4	22.1 ±0.5	22.4 ±1.6	15.6 ±3.2	10.1 ±0.4	10.4 ±0.6	7.1 ±0.3	6.9 ±0.2
Sublayer-Wise Subset	24.2 ±0.6	**24.4 ±0.6**	30.6 ±2.0	17.8 ±2.2	10.7 ±0.3	10.7 ±0.6	6.9 ±0.2	6.9 ±0.2
Layer-Wise Subset	**25.8 ±0.6**	**24.4 ±1.3**	**32.7 ±2.8**	**28.5 ±1.5**	10.9 ±0.5	10.4 ±0.2	6.8 ±0.3	6.7 ±0.1
Target-Only (same budget)	31.7 ±0.4		36.0 ±1.8		4.0 ±0.4		4.1 ±0.4	
Target-Only (early stop)	30.5 ±0.4		21.8 ±1.5		3.3 ±0.4		2.8 ±0.4	

Table 10: Random Subset and layer-normalized controls under the three fine-tuning methods: downstream metric (%) at the final checkpoint. The controls do not score; the Global Subset, layer-normalized, and Layer-Wise Subset rows use compressed scoring, and the two endpoints repeat the compressed columns of Table 9.

Method	alpaca/samsum			less-mix/tydiqa			triviaqa/nq_open			less-mix/squad		
	Full param.	LoRA	MeSO	Full param.	LoRA	MeSO	Full param.	LoRA	MeSO	Full param.	LoRA	MeSO
Full-Training	20.4 ±0.2	22.4 ±0.3	19.3 ±0.7	3.5 ±0.2	4.9 ±0.3	4.2 ±0.2	13.2 ±0.1	12.7 ±0.2	**11.7 ±0.3**	8.4 ±0.1	9.0 ±0.2	**7.4 ±0.2**
Global Random Subset	20.1 ±0.4	22.2 ±0.3	18.3 ±0.7	3.5 ±0.1	3.9 ±0.2	6.6 ±2.6	12.8 ±0.2	13.0 ±0.3	10.4 ±0.5	8.1 ±0.2	8.3 ±0.1	6.7 ±0.3
Layer-Wise Random Subset	20.6 ±0.1	21.9 ±0.3	18.0 ±0.7	3.2 ±0.1	6.0 ±0.2	4.5 ±0.4	13.1 ±0.2	12.7 ±0.3	10.2 ±0.4	8.2 ±0.1	8.9 ±0.2	7.1 ±0.2
Global Subset	20.1 ±0.7	22.3 ±0.2	19.6 ±0.6	6.4 ±0.8	12.8 ±1.5	8.9 ±2.3	13.1 ±0.2	**13.8 ±0.2**	9.0 ±1.7	8.4 ±0.1	9.0 ±0.2	6.9 ±0.2
Global Subset, layer-normalized scores	22.0 ±0.5	24.0 ±0.5	19.7 ±0.5	9.5 ±1.1	10.5 ±0.9	15.6 ±3.7	13.5 ±0.3	13.4 ±0.4	10.3 ±0.2	8.6 ±0.1	8.9 ±0.2	7.0 ±0.3
Layer-Wise Subset	**24.1 ±1.0**	**29.2 ±0.7**	**24.4 ±1.3**	**16.3 ±2.2**	**46.9 ±0.6**	**28.5 ±1.5**	**13.8 ±0.2**	13.3 ±0.4	10.4 ±0.2	**8.9 ±0.1**	**9.5 ±0.3**	6.7 ±0.1
Target-Only (same budget)	30.8 ±0.3	30.9 ±0.3	31.7 ±0.4	56.2 ±2.2	51.8 ±3.7	36.0 ±1.8	8.2 ±0.7	7.0 ±0.5	4.0 ±0.4	8.6 ±0.3	5.5 ±0.4	4.1 ±0.4
Target-Only (early stop)	32.1 ±0.3	32.8 ±0.6	30.5 ±0.4	49.1 ±1.8	51.1 ±2.9	21.8 ±1.5	10.1 ±0.5	9.3 ±0.4	3.3 ±0.4	8.8 ±0.3	8.7 ±0.4	2.8 ±0.4

raises the 64×64 agreement to 0.96 (0.87). Exact scoring is also affordable in this setting: at $\mathsf{T} = 512$ and $m = 1$, GIP is the cheapest exact method and costs at most 34 ms per step more than compressed scoring (Table 4(b)), hence its use in the main text; the advantage of compressed scoring grows with T and m. Assessed jointly, the two backends are not interchangeable: exact GIP costs 3–5 points of step time more than compressed scoring at $n = 8$, $\mathsf{T} = 512$ (Table 15 against the compressed totals quoted in Appendix E.4) and none of the accuracy, whereas compressed scoring returns those points at no accuracy cost for Global Subset Update but at an accuracy cost for Layer-Wise Subset Update that reaches double digits under full-parameter fine-tuning on the misaligned pool. Compressed scoring is therefore the choice only when T or m makes exact scoring the dominant cost (Table 4(b)), and then preferably with Global Subset Update or a 512×512 sketch, which restores the per-layer ranking agreement above; all accuracy results in this paper, and the benchmarks of Section 4.4, use exact scoring.

E.1.4 Reasoning and instruction following with Qwen3-1.7B-Base

Datasets. Next, we consider reasoning and instruction following tasks, where we sample three 32,000-example pools uniformly from `Dolci-Instruct-SFT`[12] (Olmo Team, 2025), restricted to plain alternating user/assistant conversations of at most 16K characters and excluding the Tool Use and Hardcoded Data domains: an *instruction* pool (Chat, Precise IF, Other, Multilingual, Safety), a *reasoning* pool (Math, Coding, Reasoning, Science), and a *mixed* pool over all nine domains. A fourth pool of 32,000 examples is sampled from the `Tulu 3` SFT mixture[13] (Lambert et al., 2025), in which Precise-IF data makes up about 3% and math about 36% of the examples. Every pool is decontaminated against all benchmark and target prompts.

Targets and benchmarks. Each target set has 128 examples; the main-text runs use the first 32 of the set shuffled with the training seed, and the target-set size is varied below. The benchmarks are separate datasets that share the target skill, scored by generation from the final checkpoint (greedy decoding, at most 2048 new tokens): `IFEval` (Zhou et al., 2023b) and `IFBench` (Pyatkin et al., 2025) (both prompt-level strict accuracy) for the Precise-IF target, and `MATH500` (Lightman et al., 2024) and `GSM8K` (Cobbe et al., 2021) for the Math targets, whose final answers are checked against the reference with Math-Verify (Kydlíček, 2025). The three targets are: (i) *Precise IF (`Dolci`)*: single-turn examples from the Precise-IF source of `Dolci-Instruct-SFT`, removed from every pool by id; (ii) *Math (`Dolci`)*: held-out `Dolci` examples from the pools' own math sources (`Tulu 3 Persona MATH`, `Algebra`, and `GSM`), so that the target set has the pool's math style; (iii) *Math (curated)*: `MATH`-train (Hendrycks et al., 2021) problems re-solved by the post-trained Qwen3-32B in non-thinking mode (8 samples at temperature 0.7; the shortest response whose final answer passes the same Math-Verify check against the reference is kept, and the 4 unsolved problems are dropped, leaving 124 examples). *Curated* thus means re-solved and verified, mimicking the situation where one can curate a small, high-quality target set; since `MATH`-train is disjoint from `MATH500` and `GSM8K`, it does not contaminate the benchmarks.

Training configurations. All arms use full-parameter fine-tuning of Qwen3-1.7B-Base (Qwen Team, 2025), following prior work (Kopiczko et al., 2026; Nayak et al., 2026; Shen et al., 2026), for one epoch (4000 steps) with batch size 8, maximum sequence length 2048, AdamW at learning rate 10^{-5} with a linear schedule and 3% warmup, no weight decay, `bfloat16` precision and FlashAttention-2. The training loss is the mean over examples of per-example token-mean losses, so that per-sample gradients are gradients of per-example losses, and the embedding row of the chat end-of-turn token `<|im_end|>`, which is untrained in the base model, is initialised from `<|endoftext|>` before fine-tuning so that generations terminate. Every arm is run with 5 seeds, and we report the mean $\pm$ standard error over seeds.

Data regularization. The protocol is that of the question-answering setting (Appendix E.1.1), with $n = 8$ candidates and $m = 1$ target example per step, except that the target example is drawn from the target sets above, scores are exact PIP scores (Appendix D.3.2) in every run, with no compressed-scoring arm, and top-k selection is run at $k \in \{2, 4, 6\}$, with $k = 4$ in the main text. Full-Training ignores the target set, so its arm is shared between the two Math targets on the reasoning pool. The curated updates are compared against Full-Training only: the question here is how much of a pool to use at a given mismatch, which the pool-free Target-Only Update does not address (Appendix E.1.1).

[12] https://huggingface.co/datasets/allenai/Dolci-Instruct-SFT
[13] https://huggingface.co/datasets/allenai/tulu-3-sft-mixture

Table 11: Precise-IF settings of the reasoning and instruction-following experiment (greedy decoding): every arm at $k \in \{2, 4, 6\}$ of the $n = 8$ candidates, in one block per target-set size in $\{128, 64, 32, 16\}$, where every arm uses that many leading examples of the seed-shuffled 128-example target set; Full-Training does not depend on the target set and is given once in the first row. `IFEval` and `IFBench`: prompt-level strict accuracy. Best curated arm per column within each target-size block in bold.

Target size	Method	k	`Dolci` instruction/Precise IF (`Dolci`)		`Dolci` mixed/Precise IF (`Dolci`)		`Tulu 3`/Precise IF (`Dolci`)	
			`IFEval`	`IFBench`	`IFEval`	`IFBench`	`IFEval`	`IFBench`
–	Full-Training	8	42.7 ±0.6	17.9 ±0.2	40.9 ±0.6	17.9 ±0.3	32.5 ±0.2	16.2 ±0.3
128	Global Subset	2	42.7 ±0.4	18.7 ±0.4	42.6 ±0.6	**16.9 ±0.5**	35.0 ±0.3	**17.0 ±0.3**
		4	44.0 ±0.7	18.8 ±0.3	41.3 ±0.4	16.5 ±0.3	35.3 ±0.4	16.0 ±0.4
		6	**44.1 ±0.4**	18.3 ±0.5	42.3 ±0.3	16.1 ±0.2	35.6 ±0.2	16.1 ±0.6
	Layer-Wise Subset	2	42.6 ±0.1	**19.9 ±0.7**	42.3 ±0.4	16.7 ±0.1	35.1 ±0.3	16.6 ±0.4
		4	43.3 ±0.5	18.1 ±0.5	42.3 ±0.3	16.2 ±0.3	**36.2 ±0.6**	16.3 ±0.5
		6	43.8 ±0.4	18.0 ±0.4	**42.7 ±0.2**	16.3 ±0.1	35.3 ±0.5	16.7 ±0.3
64	Global Subset	2	43.0 ±0.8	18.9 ±0.4	41.1 ±0.5	16.3 ±0.4	33.7 ±0.7	16.8 ±0.3
		4	43.5 ±0.7	18.7 ±0.6	41.8 ±0.4	16.3 ±0.5	34.6 ±0.4	16.7 ±0.2
		6	**44.1 ±0.3**	17.9 ±0.4	**42.4 ±0.3**	16.2 ±0.3	34.9 ±0.6	16.4 ±0.2
	Layer-Wise Subset	2	43.6 ±0.5	18.7 ±0.4	41.1 ±0.7	**16.7 ±0.4**	35.3 ±0.6	**17.1 ±0.5**
		4	43.8 ±0.4	**19.3 ±0.5**	42.1 ±0.3	16.0 ±0.3	**35.8 ±0.6**	**17.1 ±0.5**
		6	**44.1 ±0.3**	18.2 ±0.3	41.7 ±0.5	16.1 ±0.5	35.7 ±0.3	16.5 ±0.2
32	Global Subset	2	42.9 ±0.5	18.1 ±0.3	41.5 ±0.4	16.6 ±0.4	34.8 ±0.5	16.6 ±0.4
		4	**44.5 ±0.2**	18.5 ±0.3	41.8 ±0.3	16.5 ±0.4	35.3 ±0.3	16.0 ±0.3
		6	43.7 ±0.3	17.8 ±0.2	41.8 ±0.4	16.2 ±0.3	34.5 ±0.3	16.1 ±0.2
	Layer-Wise Subset	2	43.4 ±0.1	**19.7 ±0.8**	41.7 ±0.4	**17.1 ±0.2**	35.6 ±0.7	**17.1 ±0.8**
		4	44.1 ±0.4	18.7 ±0.5	41.7 ±0.4	**17.1 ±0.3**	35.3 ±0.4	16.4 ±0.9
		6	43.8 ±0.3	18.0 ±0.6	**42.2 ±0.3**	15.8 ±0.2	**36.1 ±0.4**	16.5 ±0.5
16	Global Subset	2	42.7 ±0.3	18.5 ±0.6	41.4 ±0.6	16.3 ±0.5	35.0 ±0.5	16.2 ±0.5
		4	43.7 ±0.3	18.7 ±0.6	41.5 ±0.3	16.5 ±0.2	34.9 ±0.6	15.9 ±0.5
		6	43.4 ±0.5	18.8 ±0.6	**42.0 ±0.5**	15.4 ±0.3	35.1 ±0.4	15.8 ±0.3
	Layer-Wise Subset	2	42.7 ±0.5	**19.7 ±0.5**	40.1 ±0.4	**17.5 ±0.6**	34.5 ±0.4	**17.2 ±0.6**
		4	43.4 ±0.4	19.6 ±0.4	40.7 ±0.2	16.7 ±0.4	35.0 ±0.4	**17.2 ±0.5**
		6	**43.8 ±0.3**	18.7 ±0.3	41.7 ±0.4	16.5 ±0.4	**35.8 ±0.2**	16.9 ±0.5

Results. Table 3 reports both curated updates at a single shared configuration, $k = n/2$ with 32 target examples, used for all seven settings and both methods. Tables 11 and 12 sweep $k \in \{n/4, n/2, 3n/4\}$ and the target-set size: every curated arm is rerun with 128, 64, 32, and 16 target examples, obtained for each seed by shuffling the 128 examples with the training seed and keeping a prefix, so that the sets are nested within a seed; Full-Training does not depend on the target set and is shared. The qualitative picture of Section 4.1.2 holds at every k and every target-set size down to 16 examples: selection helps most where the target skill is diluted in the pool, with the largest gains on `Tulu 3`; on the Precise-IF target both curated methods match or beat Full-Training on `IFEval` in every setting; and on the Math targets the gains are largest on `Tulu 3` and, with the Math (curated) target, for Layer-Wise Subset Update, which improves both `MATH500` and `GSM8K`. `IFEval`'s constraint templates appear verbatim in `Dolci` Precise-IF prompts (the prompts themselves are decontaminated), whereas `IFBench` uses new constraint types, so the two benchmarks separate fitting the target's constraint types from generalizing beyond them.

In a smaller set of additional runs, we observe that Random Subset Update at $k = n/2$ on the `Dolci` pools stays at or below Full-Training on `IFEval` and within a point of it elsewhere, so the gains of the curated arms over Full-Training are not an effect of training on fewer examples; the `IFBench` decrease on the mixed pool, which random subsets share, reflects the smaller number of trained examples rather than the selection.

E.2 Reinforcement learning from human feedback

We now provide the missing details of the RLHF experiments in Section 4.2, including details on datasets, libraries, training configurations, and data regularization.

E.2.1 Libraries and datasets

We implement PPO-based RLHF on top of `TRL`[14] (von Werra et al., 2020) (Apache-2.0 license, version 0.26.1), using its value-head policy model and re-implementing its PPO loop in our trainer

[14] https://huggingface.co/docs/trl/index

Table 12: Math settings of the reasoning and instruction-following experiment (greedy decoding), same layout as Table 11: the three settings with the Math (Dolci) target, which uses the pools' own math sources, and the reasoning pool with the Math (curated) target, MATH-train problems re-solved and verified by QWEN3-32B. Full-Training on the reasoning pool is shared between its two targets. Best curated arm per column within each target-size block in bold.

Target size	Method	k	Dolci reasoning/Math (Dolci)		Dolci mixed/Math (Dolci)		Tulu 3/Math (Dolci)		Dolci reasoning/Math (curated)	
			MATH500	GSM8K	MATH500	GSM8K	MATH500	GSM8K	MATH500	GSM8K
–	Full-Training	8	63.6 ±0.5	79.9 ±0.2	64.2 ±0.3	78.8 ±0.1	57.1 ±0.5	78.8 ±0.2	63.6 ±0.5	79.9 ±0.2
128	Global Subset	2	63.4 ±0.4	80.3 ±0.3	**63.6 ±0.6**	**80.0 ±0.1**	60.0 ±0.6	79.0 ±0.3	63.1 ±0.7	79.7 ±0.3
		4	62.8 ±0.5	80.8 ±0.2	**63.6 ±0.7**	79.7 ±0.1	58.6 ±0.5	79.6 ±0.2	64.0 ±0.5	80.5 ±0.2
		6	**64.0 ±0.7**	**80.9 ±0.3**	63.4 ±0.6	**80.0 ±0.2**	57.3 ±0.8	79.5 ±0.5	64.0 ±0.7	80.3 ±0.2
	Layer-Wise Subset	2	60.9 ±0.7	79.0 ±0.5	**63.6 ±0.4**	79.9 ±0.2	**60.5 ±0.5**	79.5 ±0.3	65.0 ±0.8	80.8 ±0.4
		4	63.1 ±0.3	80.1 ±0.4	63.0 ±0.8	79.7 ±0.2	59.2 ±0.4	79.4 ±0.3	**65.8 ±0.3**	**80.9 ±0.5**
		6	**64.0 ±0.6**	80.4 ±0.2	63.4 ±0.3	79.6 ±0.1	59.2 ±0.5	**80.0 ±0.2**	63.5 ±0.5	80.8 ±0.3
64	Global Subset	2	62.6 ±0.6	80.4 ±0.2	63.8 ±0.5	79.9 ±0.4	**60.2 ±0.6**	79.0 ±0.3	63.3 ±0.2	80.0 ±0.3
		4	64.0 ±0.3	80.4 ±0.1	63.4 ±0.5	**80.0 ±0.3**	59.3 ±0.5	79.4 ±0.2	64.8 ±0.5	80.2 ±0.3
		6	**64.2 ±0.6**	**80.6 ±0.2**	**64.3 ±0.5**	79.4 ±0.3	57.0 ±0.6	79.5 ±0.2	64.3 ±0.4	80.3 ±0.2
	Layer-Wise Subset	2	61.8 ±0.7	78.8 ±0.4	62.7 ±0.3	79.7 ±0.2	59.4 ±0.8	79.7 ±0.4	**65.6 ±0.6**	**81.0 ±0.2**
		4	62.9 ±0.2	79.7 ±0.4	62.6 ±0.6	79.6 ±0.2	59.2 ±0.1	**79.9 ±0.1**	65.1 ±0.2	80.8 ±0.3
		6	63.8 ±0.6	**80.6 ±0.1**	62.8 ±0.4	79.3 ±0.2	58.6 ±0.3	79.4 ±0.3	64.2 ±0.3	80.7 ±0.3
32	Global Subset	2	64.2 ±0.3	80.2 ±0.2	63.9 ±0.5	**80.4 ±0.2**	60.2 ±0.5	79.4 ±0.3	62.9 ±0.2	79.8 ±0.4
		4	**64.4 ±0.3**	80.7 ±0.2	**64.2 ±0.4**	**80.4 ±0.3**	58.3 ±0.4	79.4 ±0.2	64.0 ±0.2	79.9 ±0.1
		6	63.8 ±0.4	**80.9 ±0.2**	63.7 ±0.1	**80.4 ±0.1**	56.4 ±0.7	79.3 ±0.3	63.9 ±0.4	79.9 ±0.2
	Layer-Wise Subset	2	62.1 ±0.8	79.4 ±0.5	63.5 ±0.7	80.2 ±0.4	60.4 ±0.4	79.4 ±0.2	**66.0 ±0.5**	80.3 ±0.4
		4	62.8 ±0.2	80.3 ±0.2	63.9 ±0.4	**80.4 ±0.3**	**60.6 ±0.6**	79.8 ±0.3	65.1 ±0.5	**81.0 ±0.2**
		6	63.6 ±0.3	80.6 ±0.3	63.1 ±0.4	79.8 ±0.4	58.8 ±0.3	**80.0 ±0.2**	64.6 ±0.7	80.9 ±0.2
16	Global Subset	2	63.8 ±0.4	80.1 ±0.1	**64.4 ±0.3**	**80.2 ±0.2**	60.0 ±0.6	78.8 ±0.5	62.5 ±0.3	80.0 ±0.1
		4	63.4 ±0.3	80.7 ±0.2	63.5 ±0.3	79.9 ±0.3	59.4 ±0.5	79.8 ±0.3	63.1 ±0.2	79.9 ±0.2
		6	**64.8 ±0.6**	**81.0 ±0.1**	64.0 ±0.4	79.8 ±0.3	57.6 ±0.2	78.8 ±0.4	64.4 ±0.5	80.2 ±0.2
	Layer-Wise Subset	2	62.6 ±0.8	79.7 ±0.3	63.8 ±0.6	80.0 ±0.3	**61.2 ±0.6**	79.7 ±0.3	64.7 ±0.4	81.0 ±0.5
		4	64.0 ±0.8	80.4 ±0.3	63.3 ±0.3	80.0 ±0.2	60.0 ±0.5	80.0 ±0.4	**65.4 ±0.7**	**81.1 ±0.2**
		6	63.4 ±0.5	80.6 ±0.3	63.3 ±0.5	79.9 ±0.3	59.4 ±0.5	**80.3 ±0.2**	64.8 ±0.7	**81.1 ±0.2**

so that per-sample scores can be computed within each mini-batch, and sample prompts from real-toxicity-prompts[15] (Gehman et al., 2020) (Apache-2.0 license). For rewards, we use the LFTW R4 Target toxicity detector[16] (Vidgen et al., 2021), and we evaluate toxicity using da-electra-hatespeech-detection[17]. We note that this experimental setup largely follows the official example provided from TRL (Hugging Face, 2023)[18] (whcih is also adopted by Hu et al. (2025b)), but with a newer TRL version. Reward model, evaluation classifier, decoding, and continuation length follow the TRL detoxification example (Hugging Face, 2023) and IIF (Hu et al., 2025b). The evaluation classifier is independent of the reward model, so the toxicity row of Figure 2 is not the signal the policy is optimized against, while the reward row reports that signal itself. Hu et al. (2025b) score 400 Wiki-Toxic prompts after training; we score 500 held-out real-toxicity-prompts prompts at every PPO step, which yields the curves of Figure 2.

E.2.2 TRAINING CONFIGURATIONS

Training prompts are the first 80% (unshuffled) of the 35,109 real-toxicity-prompts entries with prompt toxicity above 0.3, i.e., 28,087 prompts, each truncated to a random length of 10–15 tokens; the remaining 20% is split in half, one half providing the held-out target prompts of Section 4.2 and the other the evaluation prompts, so the two never overlap. For each prompt, the policy samples a continuation of up to 30 tokens (temperature 1, no top-k or top-p truncation), scored by the reward model; PPO updates maximize reward while penalizing deviation from a frozen reference policy via an adaptive KL penalty. One epoch over the training prompts with rollout batch size 256 gives 110 PPO steps. We use 4 PPO epochs with mini-batch size 4, learning rate 10^{-5} (policy) and 5×10^{-4} (value head), a linear schedule with 3% warmup, and no weight decay. LoRA is applied to the query and value projections with rank 16, $\alpha = 32$, and no dropout. Curation acts on the LoRA parameters; the value head is not scored and is updated with every rollout of the mini-batch (for IIF, with the pre-filtered rollout batch). The PPO surrogate uses clipping range 0.2 for both policy and value, value function coefficient 0.1, discount factor $\gamma = 1.0$, and generalized advantage estimation (GAE) with $\lambda = 0.95$. The KL coefficient is initialized at 0.02 and adapted by TRL's adaptive KL controller

[15] https://huggingface.co/datasets/allenai/real-toxicity-prompts

[16] https://huggingface.co/facebook/roberta-hate-speech-dynabench-r4-target

[17] https://huggingface.co/DaNLP/da-electra-hatespeech-detection

[18] https://huggingface.co/docs/trl/v0.4.7/en/detoxifying_a_lm

(target 70, horizon 10,000) using the k_1 estimator; as in `TRL`, the remaining mini-batches of a rollout are skipped once the mini-batch policy KL exceeds 0.45 (1.5 times a target of 0.3). Evaluation is performed every PPO step by sampling continuations (temperature 1, top-k 50, no top-p truncation) for 500 held-out prompts with prompt toxicity above 0.5 (batch size 256) and reporting the mean predicted toxicity.

E.2.3 DATA REGULARIZATION

In this setting, scores are exact PIP scores (Appendix D.3.2); Appendix E.2.5 repeats the same configurations with compressed scoring. Two types of "target set" are considered: (i) the same rollout batch (self-referencing), or (ii) new rollouts from the current policy on a batch of 256 prompts drawn at each step from a held-out target set of 1024 prompts (disjoint from the evaluation prompts, see above). Furthermore, we consider two target loss functions for the target signals: (i) the *expected reward*, estimated on the target rollouts $\{(x_i, y_i)\}_{i=1}^{m}$ by the REINFORCE surrogate $-\frac{1}{m}\sum_i \tilde{R}_i \log \pi_\theta(y_i \mid x_i)$, where $\tilde{R}_i$ is the raw reward standardized over the target batch (no KL penalty, value baseline, or clipping) and $\log \pi_\theta(y_i \mid x_i)$ is the sequence log-probability, i.e., the sum over response tokens, and (ii) the *PPO training loss*, i.e., the clipped surrogate plus the value loss that the policy is trained with. In both cases the target gradient is captured once per rollout batch at the parameters that generated the rollouts, so for the training-loss target the importance ratio equals one and clipping is inactive: it reduces to the token-level policy gradient with GAE advantages plus the value-loss term.

Finally, we solve S_t (or $S_{t,l}$ in the case of Layer-Wise Subset Update) via *negative score filtering*, where at each PPO step, samples (i.e., rollouts) with negative scores are dropped from the update. The retained rollouts of a group are averaged, i.e., the group receives the gradient of the mini-batch loss restricted to its retained samples, so the effective update is normalized by the number of retained rollouts rather than by the mini-batch size; a group whose rollouts all score negative receives a zero gradient at that step (for IIF, an empty filtered rollout batch skips the PPO update), and the value head is updated with every rollout as described above. In every configuration the retained rollouts are trained with the full PPO objective, including the KL penalty; the target loss enters only through the alignment scores.

E.2.4 TOXICITY UNDER DIFFERENT JUDGES

The evaluation toxicity in Figure 2 is scored by `da-electra-hatespeech-detection`, the judge of the prior work we follow (Hu et al., 2025b). To show that the comparison does not hinge on this judge, we regenerate continuations from the final checkpoint of every run on the same 500 evaluation prompts, with the same decoding as the in-training evaluation, and score them with four classifiers: the paper judge (`DaNLP`), the reward model (`LFTW`), and two English toxicity classifiers trained on the Jigsaw toxic-comment data, `roberta_toxicity_classifier`[19] (`s-nlp`) and `toxic-bert`[20]. Figure 10 reports the mean toxicity probability over 5 seeds, with the untuned GPT-NEO-2.7B as a reference. The picture of Figure 2 holds under every judge: with the expected-reward target, every curated update ends far below standard PPO under all four judges (Global and Layer-Wise Subset Updates by roughly an order of magnitude under every judge, IIF by a factor of about four under the paper judge and by an order of magnitude under the two English judges), and with the training-loss target they remain close to standard PPO. Under the English judges, the three curated updates are within noise of one another at the final checkpoint.

E.2.5 COMPRESSED SCORING

The RLHF results of Section 4.2 use exact PIP scoring (Appendix D.3.2), which is cheap under LoRA, where the per-sample adapter gradients are small. Here we ask whether the RL comparison is robust to the scoring approximation, since the target signal is itself a noisy reward estimate and negative score filtering keys on the sign of every score: we repeated every curated configuration of Figure 2 with compressed scoring (Appendix D.3.3) at $\kappa^{(l)} = 64 \times 64$ under the same seeds and hyperparameters; standard PPO does no scoring and is shared. With LoRA rank $r = 16$, the rank-16

[19] https://huggingface.co/s-nlp/roberta_toxicity_classifier
[20] https://huggingface.co/unitary/toxic-bert

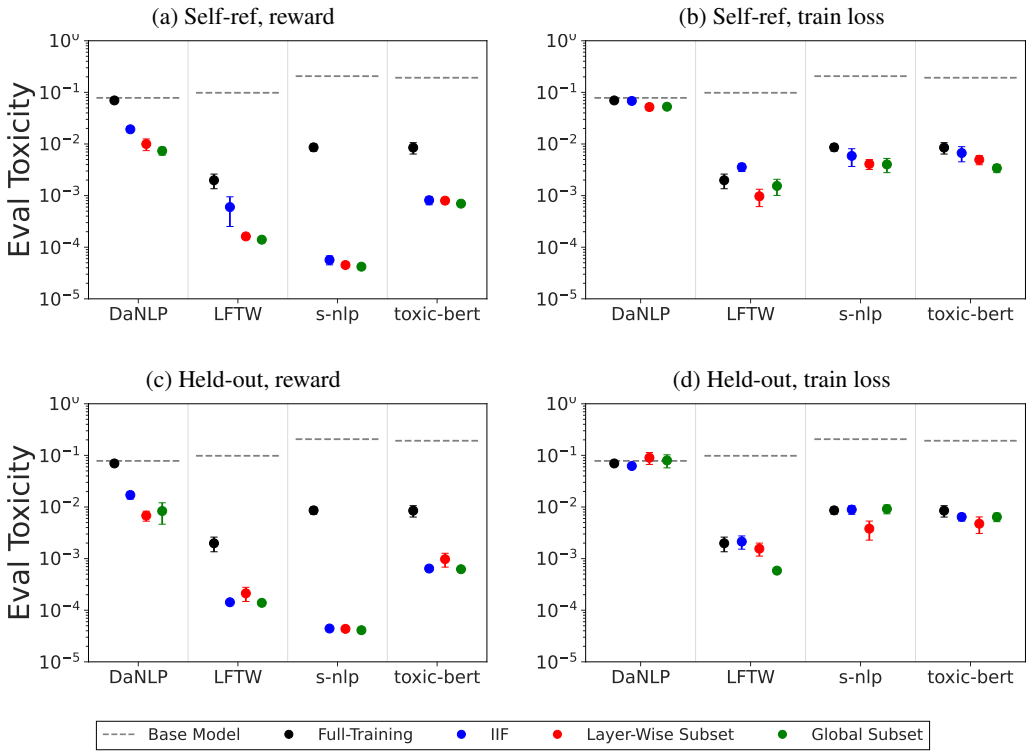

Figure 10: RLHF toxicity (mean probability, log scale) of the final checkpoints under four judges: the paper judge (`DaNLP`), the reward model (`LFTW`), and two English classifiers (`s-nlp`, `toxic-bert`); dashed lines mark the untuned base model. Panels follow Figure 2.

Table 13: Final-checkpoint toxicity under four judges (mean toxicity probability over 500 evaluation prompts, in %) for exact PIP and compressed scoring. Standard PPO does no scoring and is shared.

Target	Method	DaNLP		LFTW		s-nlp		toxic-bert	
		Exact	Compressed	Exact	Compressed	Exact	Compressed	Exact	Compressed
Full-Training (standard PPO)		6.98 ± 0.45		0.20 ± 0.06		0.86 ± 0.13		0.85 ± 0.21	
Self-ref., reward	IIF	1.93 ± 0.25	2.77 ± 0.41	0.06 ± 0.03	0.02 ± 0.00	0.01 ± 0.00	0.01 ± 0.00	0.08 ± 0.01	0.07 ± 0.00
	Global Subset	0.74 ± 0.13	1.22 ± 0.26	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.07 ± 0.00	0.06 ± 0.00
	Layer-Wise Subset	1.00 ± 0.25	0.72 ± 0.18	0.02 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.08 ± 0.01	0.06 ± 0.00
Self-ref., train loss	IIF	6.82 ± 0.79	5.98 ± 0.40	0.36 ± 0.06	0.12 ± 0.04	0.59 ± 0.22	0.45 ± 0.07	0.67 ± 0.22	0.44 ± 0.08
	Global Subset	5.28 ± 0.28	6.20 ± 0.68	0.15 ± 0.05	0.21 ± 0.04	0.40 ± 0.12	1.05 ± 0.15	0.34 ± 0.06	0.68 ± 0.09
	Layer-Wise Subset	5.20 ± 0.76	7.38 ± 1.80	0.10 ± 0.04	0.15 ± 0.04	0.41 ± 0.09	0.50 ± 0.07	0.50 ± 0.10	0.65 ± 0.13
Held-out, reward	IIF	1.71 ± 0.28	1.80 ± 0.29	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.01 ± 0.00	0.06 ± 0.00	0.06 ± 0.00
	Global Subset	0.84 ± 0.37	1.55 ± 0.25	0.01 ± 0.00	0.01 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.06 ± 0.00	0.06 ± 0.00
	Layer-Wise Subset	0.68 ± 0.15	0.56 ± 0.10	0.02 ± 0.01	0.05 ± 0.02	0.00 ± 0.00	0.01 ± 0.00	0.10 ± 0.03	0.09 ± 0.02
Held-out, train loss	IIF	6.23 ± 0.54	5.40 ± 0.34	0.21 ± 0.06	0.11 ± 0.03	0.89 ± 0.17	0.85 ± 0.15	0.64 ± 0.11	0.63 ± 0.06
	Global Subset	7.99 ± 2.26	7.13 ± 1.82	0.06 ± 0.01	0.24 ± 0.04	0.92 ± 0.18	0.64 ± 0.23	0.64 ± 0.12	0.58 ± 0.13
	Layer-Wise Subset	9.01 ± 2.33	8.89 ± 1.31	0.16 ± 0.04	0.22 ± 0.07	0.38 ± 0.15	0.76 ± 0.23	0.47 ± 0.17	0.67 ± 0.17

side of each adapter factor is already smaller than 64, so its compressor is the identity and only the 2560-dimensional side is projected.

Figure 11 shows the compressed-scoring training curves on the axes of Figure 2, and Table 13 compares the final checkpoints under the four judges of Appendix E.2.4. Compressed scoring reproduces the main comparison: with the expected-reward target every curated update again ends far below standard PPO under all four judges (Layer-Wise Subset Update by an order of magnitude under the paper judge, Global Subset Update and IIF by a factor of two to six, and all three by an order of magnitude under the two English judges), and with the training-loss target the curated updates stay close to standard PPO. With the expected-reward target, final toxicities move by 0.1–0.8 points, within one to two standard errors and in both directions (Layer-Wise Subset Update slightly lower with compressed scoring, Global Subset Update and self-referencing IIF slightly higher), so compressed scoring orders the endpoints as Layer-Wise below Global below IIF, whereas exact scoring leaves Global and Layer-Wise Subset Updates tied within error.

Two finer differences remain. Under the reward target, Layer-Wise and Global Subset Updates reach 3% evaluation toxicity after a similar number of PPO steps with compressed scoring (22–27 on average, against 10–18 with exact scoring and 38–45 for IIF), so the approximation costs some of the early speed-up. Under the training-loss target the backend matters more: the self-referencing Global and Layer-Wise arms, which end below standard PPO with exact scoring (5.3 and 5.2 against 7.0 under the paper judge), end at 6.2 and 7.4 with compressed scoring.

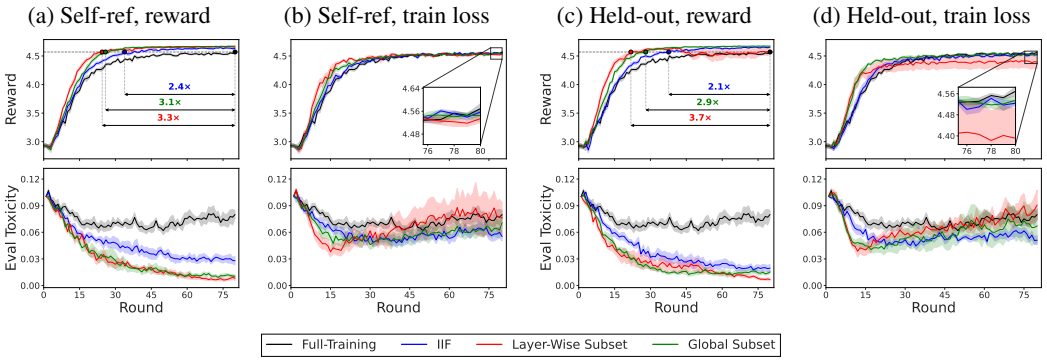

Figure 11: RLHF training curves with compressed scoring, same layout and axes as Figure 2. **Top:** training reward. **Bottom:** evaluation toxicity (mean probability).

E.3 Reinforcement learning with verifiable rewards

We now provide the missing details of the RLVR experiments in Section 4.3: libraries and datasets, training configurations, and data regularization.

E.3.1 LIBRARIES AND DATASETS

We use the `Verl` framework[21] (Sheng et al., 2024) (Apache-2.0 license) to train on the `MATH` dataset[22] (Hendrycks et al., 2021) (MIT license). We note that this experimental setup largely follows the official example provided from `Verl`,[23] with different models and datasets.

E.3.2 TRAINING CONFIGURATIONS

The policy is initialized from the pretrained base model QWEN3-1.7B-BASE (Qwen Team, 2025) rather than its post-trained counterpart and trained with full-parameter updates; each run uses four `H200` GPUs on one node. Each training sample is a math problem; the model generates 8 candidate solutions and receives a verifiable reward from exact-match checking against the ground-truth answer. The policy is optimized using GRPO (Shao et al., 2024) with training batch size 128, mini-batch size 32, micro-batch size 2 per GPU for the curated arms and 4 for standard GRPO, KL coefficient 0.001 (low-variance estimator), learning rate 10^{-6}, maximum prompt length 1024, and maximum response length 2048. Gradient checkpointing is enabled. We train for 80 rounds and evaluate every 3 rounds on about 4,500 `MATH` test problems with greedy decoding and one sample per prompt (`Verl`'s default validation setting); Figure 3 reports this accuracy. All other settings follow the default `Verl` configuration.

E.3.3 DATA REGULARIZATION

We instantiate the framework as in-run cross-validation with exact PIP scores; an RLVR round is dominated by rollout generation, so the scoring backend is immaterial to its cost. Every fourth round, 64 prompts drawn from the training set itself serve as the target batch (no prompts are held aside), each is rolled out 8 times with the current policy, GRPO's own group size, and the REINFORCE surrogate of the expected verifier reward on these 512 target rollouts (as in Appendix E.2) provides the target signal, on top of the 1024 training rollouts (128 prompts $\times$ 8) of every round. As in RLHF, the target gradient is captured in a separate pass, so the target rollouts never enter the gradient of any parameter, and is reused across mini-batch updates; since the target gradient changes slowly along the trajectory, we amortize it over four rounds rather than refreshing it every round, which brings the target cost to 128 target rollouts per round, an eighth of the training rollouts. Negative score filtering then drops the training rollouts with negative alignment scores. Retained rollouts are aggregated with `Verl`'s default token-mean loss, so each layer's update is the policy-gradient loss averaged over the response tokens of the rollouts it retains, and a layer that retains no rollout receives a zero gradient at that round; selection and this normalization act per micro-batch of two rollouts per GPU, and the micro-batch gradients are averaged across micro-batches and data-parallel ranks exactly as in standard GRPO. Scoring and selection act on all linear modules, the embedding matrix, and the tied output head (Appendix D.5).

Results. Table 14 quantifies Figure 3. After the first 20 rounds, Layer-Wise Subset Update leads at every evaluation round and reaches a final accuracy of 63.8%, against 63.0% for both standard GRPO and Global Subset Update. The target rollouts are the price of the target signal, which in RL, unlike SFT, has to be re-sampled from the current policy and is standard in influence-based data selection in RL (Wang et al., 2025); amortized over the four rounds between refreshes, they add an eighth to the rollout budget of every round. This overhead does not account for the gain: continuing standard GRPO past the 80-round budget, it first reaches the round-78 accuracy of Layer-Wise Subset Update at about round 147, after roughly 1.7 times as many rollouts as Layer-Wise Subset Update has generated by round 78, target rollouts included; matched on rollouts instead, i.e., at about 88 rounds, standard GRPO remains at 63.0%.

E.4 SYSTEM EFFICIENCY

This section expands the system benchmarks of Section 4.4 in three directions: the complete per-step sweep, which adds the small-batch configuration $n = 2$, $\mathsf{T} = 1024$ and the two-pass implementation

[21] https://github.com/verl-project/verl

[22] https://github.com/hendrycks/math

[23] https://github.com/verl-project/verl/blob/65eb5a15058d73ef54064092222b7af67d658241/examples/grpo_trainer/run_qwen3-8b.sh

Table 14: RLVR evaluation accuracy (%) of Figure 3 at selected rounds and at the end of training.

	Full-Training	**Global Subset**	**Layer-Wise Subset**
Round 30	61.6 ± 0.2	61.7 ± 0.2	62.6 ± 0.1
Round 60	62.8 ± 0.1	62.7 ± 0.3	63.3 ± 0.3
Final	63.0 ± 0.1	63.0 ± 0.1	63.8 ± 0.2

of Global Subset Update (Global Subset (2P), Appendix D.2.4) to Table 4(a) (Table 15); the same sweep and the peak GPU memory under activation checkpointing (Tables 16 and 17); the scoring-cost sweep of Table 4(b) is discussed last. The synthetic benchmarks run on a single `A40` GPU in `bfloat16` on three models from different architecture families (SMOLLM2-360M, TINYLLAMA-1.1B, LLAMA-3.2-3B) with exact GIP scoring, the backend of the question-answering experiments (Appendix E.1.1), and report timings on fixed-length batches averaged over 20 timed iterations after 10 warm-up iterations.

Per-step breakdown. Table 4(a) and Table 15 split one training step into forward, backward, and optimizer time, and the backward pass into four components: `a.grad` computes the input-activation gradient $\partial \ell / \partial a^{(l)}$ with a single matrix multiplication against the weight matrix (the output gradient $\partial \ell / \partial e^{(l)}$ is propagated from the layer above and costs nothing at this layer); `scoring` computes the per-sample alignment scores; `w.grad` assembles the weight gradient from the selected samples; and `autograd` collects the remaining overhead, namely graph traversal, memory allocation, hook dispatch, and the backward passes of the non-linear modules (normalization layers, activation functions, etc.).

The overhead of the curated updates is the balance of two opposing parts. The extra target example and the scoring add a nearly fixed cost: `Forward` and `a.grad` grow with the $(n+m)/n = 9/8$ larger batch, and `scoring` costs 32–86 ms on the three models. Against this, `w.grad` shrinks, because only the $k = 4$ kept samples enter the weight gradient and the fused kernel reads them in place: by $1.3\times$ on SMOLLM2-360M (27.5 to 21.6 ms), $1.7\times$ on TINYLLAMA-1.1B (75.9 to 44.4 ms), and $2.7\times$ on LLAMA-3.2-3B (338 to 126 ms). On the smallest model the step is launch-bound, so the saving is small and the fixed costs dominate ($+18$–20%); as the model grows the weight gradient becomes compute-bound and the saving grows with it, until on LLAMA-3.2-3B it outweighs the fixed costs and the curated step is on par with Full-Training (-1% to -2%). The relative overhead is therefore configuration-dependent. In the small-batch configuration ($n = 2$, $\mathsf{T} = 1024$) the single target example is half the candidate batch, so `Forward` and `a.grad` grow by up to $1.6\times$ and the fixed costs are amortized over only two candidates; the overhead of Layer-Wise Subset Update is then $+47\%$ on SMOLLM2-360M, $+35\%$ on TINYLLAMA-1.1B, and $+21\%$ on LLAMA-3.2-3B, although the absolute overhead stays within about 180 ms per step. The favourable regime is the one that matters for LLM post-training, namely larger models and larger candidate batches. Finally, the two-pass Global Subset (2P) repeats the forward pass to separate scoring from update assembly, and costs $+33\%$ to $+60\%$ over Full-Training at $n = 8$; it is needed only when a group spans several checkpoint segments (below). With compressed scoring ($\kappa^{(l)} = 64 \times 64$, Appendix D.3.3) in place of GIP, the same steps cost $+15\%$, $+4\%$, and -5% at $n = 8$: the cheaper backend saves 3–5 points of step time in this configuration, at the accuracy cost quantified in Appendix E.1.3, and its advantage grows with T and m (Table 4(b)).

Activation checkpointing. As noted in Section 3, the partition granularity determines the memory footprint and the compatibility with activation checkpointing. Table 16 reports peak GPU memory with and without checkpointing, and Table 17 the per-step breakdown with checkpointing.

Without checkpointing, peak memory is nearly identical across methods, and the difference to Full-Training is the m extra target samples. With checkpointing, the group of Global Subset Update spans many layers across the checkpointing schedule and breaks the one-pass scheduling, so its memory overhead grows visibly (6.5 vs. 4.6 GB for Full-Training on SMOLLM2-360M, 12.4 vs. 10.3 GB on TINYLLAMA-1.1B, and 30.7 vs. 29.9 GB on LLAMA-3.2-3B at $n = 8$, $\mathsf{T} = 512$), whereas Layer-Wise Subset Update stays within 0.3 GB of Full-Training. Global Subset (2P), which separates the scoring and gradient passes (Appendix D.2.4), matches Layer-Wise's footprint at a compute cost of $+38\%$ to $+85\%$ over Full-Training. For the one-pass methods, the compute overhead under checkpointing stays between $+1\%$ and $+18\%$ at $n = 8$, $\mathsf{T} = 512$ (LLAMA-3.2-3B is again at parity with Full-Training), since checkpointing inflates the baseline and the scoring-augmented backward pass roughly in proportion.

Table 15: Per-step component breakdown (ms) on a single `A40` GPU across three models and two batch configurations ($m = 1$, $k = n/2$, exact GIP scoring).

(a) SMOLLM2-360M, $n = 8$, $\mathsf{T} = 512$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	78.7	90.5	89.1	89.0	49.9
Backward	155.2	194.1	191.6	169.7	80.8
`a.grad`	32.6	33.5	33.3	33.4	16.0
`scoring`	—	32.2	32.5	32.1	—
`w.grad`	27.5	21.6	21.2	—	15.4
`autograd`	95.1	106.8	104.5	104.2	49.4
Optimizer	26.6	26.7	26.6	—	26.6
Total	**260**	**311** (+19.5%)	**307** (+18.0%)	**416** (+59.7%)	

(b) SMOLLM2-360M, $n = 2$, $\mathsf{T} = 1024$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	49.4	61.6	62.0	61.8	51.2
Backward	82.2	144.8	143.7	130.9	81.6
`a.grad`	15.9	25.5	25.5	25.5	14.9
`scoring`	—	31.6	32.5	32.3	—
`w.grad`	15.2	12.4	12.2	—	12.9
`autograd`	51.1	75.2	73.5	73.1	53.8
Optimizer	26.6	26.6	26.6	—	28.2
Total	**158**	**233** (+47.3%)	**232** (+46.8%)	**354** (+123.5%)	

(c) TINYLLAMA-1.1B, $n = 8$, $\mathsf{T} = 512$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	134.0	145.7	146.2	146.6	71.1
Backward	278.5	306.6	306.3	261.6	145.8
`a.grad`	78.3	81.2	81.4	81.5	41.7
`scoring`	—	42.0	42.3	42.1	—
`w.grad`	75.9	44.4	44.4	—	39.5
`autograd`	124.3	139.1	138.1	137.9	64.6
Optimizer	80.7	80.7	80.8	—	80.7
Total	**493**	**533** (+8.1%)	**533** (+8.1%)	**706** (+43.1%)	

(d) TINYLLAMA-1.1B, $n = 2$, $\mathsf{T} = 1024$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	71.6	103.8	103.9	104.2	54.5
Backward	147.6	220.5	219.9	195.5	79.6
`a.grad`	41.5	57.0	57.1	57.2	21.6
`scoring`	—	41.4	42.0	41.9	—
`w.grad`	39.2	24.5	24.1	—	21.3
`autograd`	66.9	97.6	96.7	96.5	36.7
Optimizer	80.7	80.7	80.7	—	80.8
Total	**300**	**405** (+35.0%)	**405** (+34.9%)	**515** (+71.6%)	

(e) LLAMA-3.2-3B, $n = 8$, $\mathsf{T} = 512$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	340.0	387.6	387.4	388.8	188.8
Backward	821.0	752.4	758.8	621.5	424.7
`a.grad`	226.2	254.7	252.4	256.2	116.9
`scoring`	—	85.5	85.1	85.1	—
`w.grad`	338.1	126.2	136.0	—	173.5
`autograd`	256.7	286.0	285.3	280.3	134.3
Optimizer	235.5	235.4	235.5	—	235.5
Total	**1397**	**1375** (-1.5%)	**1382** (-1.1%)	**1859** (+33.1%)	

(f) LLAMA-3.2-3B, $n = 2$, $\mathsf{T} = 1024$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	183.5	263.7	263.1	264.3	113.7
Backward	426.8	523.1	526.3	444.9	200.6
`a.grad`	115.5	165.6	163.7	164.4	62.4
`scoring`	—	85.9	85.6	85.5	—
`w.grad`	172.4	70.6	75.0	—	63.0
`autograd`	138.9	201.0	202.0	194.9	75.3
Optimizer	235.6	235.5	235.6	—	235.5
Total	**846**	**1022** (+20.8%)	**1025** (+21.2%)	**1259** (+48.8%)	

Table 16: Peak GPU memory (GB) across three models and two batch configurations ($m = 1$, $k = n/2$, exact GIP scoring) on a single `A40` GPU, with and without activation checkpointing (ckpt).

Model	Setting	$n = 8$, $T = 512$				$n = 2$, $T = 1024$			
		Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)
SMOLLM2-360M	No ckpt	9.4	10.3	10.3	10.3	5.7	7.6	7.6	7.6
	With ckpt	4.6	4.9	6.5	4.9	3.4	3.9	5.0	3.9
TINYLLAMA-1.1B	No ckpt	15.0	16.1	16.1	16.1	10.6	12.9	12.9	12.9
	With ckpt	10.3	10.3	12.4	10.3	10.3	10.3	10.4	10.3
LLAMA-3.2-3B	No ckpt	37.9	40.5	40.5	40.5	29.9	33.0	33.0	33.0
	With ckpt	29.9	29.9	30.7	29.9	29.9	29.9	29.9	29.9

Table 17: Same settings as Table 15 with activation checkpointing.

(a) SMOLLM2-360M, $n = 8$, $\mathsf{T} = 512$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	78.0	88.8	89.4	88.9	82.2
Backward	218.3	264.9	263.4	241.6	155.6
`a.grad`	32.6	33.4	33.3	33.4	19.5
`scoring`	—	31.9	32.3	32.1	—
`w.grad`	27.5	21.5	21.3	—	17.3
`autograd`	158.2	178.2	176.4	176.1	118.8
Optimizer	26.6	26.6	26.6	—	28.5
Total	**323**	**380** (+17.8%)	**379** (+17.5%)	**597** (+84.8%)	

(b) SMOLLM2-360M, $n = 2$, $\mathsf{T} = 1024$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	70.8	73.7	72.4	74.0	79.2
Backward	150.4	203.0	201.5	180.8	163.5
`a.grad`	19.2	26.2	26.2	25.6	17.0
`scoring`	—	32.8	34.2	32.3	—
`w.grad`	17.1	13.6	12.4	—	14.3
`autograd`	114.1	130.3	128.6	123.0	132.2
Optimizer	27.8	26.8	26.8	—	28.3
Total	**249**	**303** (+21.9%)	**301** (+20.8%)	**526** (+111.2%)	

(c) TINYLLAMA-1.1B, $n = 8$, $\mathsf{T} = 512$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	143.4	145.3	145.1	146.0	71.0
Backward	399.1	438.5	436.6	393.6	209.5
`a.grad`	77.9	81.4	81.0	81.7	41.6
`scoring`	—	42.1	42.2	42.3	—
`w.grad`	75.5	44.5	44.5	—	39.4
`autograd`	245.6	270.5	268.9	269.5	128.5
Optimizer	80.7	80.7	80.8	—	80.7
Total	**623**	**665** (+6.6%)	**662** (+6.3%)	**901** (+44.5%)	

(d) TINYLLAMA-1.1B, $n = 2$, $\mathsf{T} = 1024$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	81.8	103.3	103.7	103.8	51.1
Backward	211.5	314.0	313.6	288.6	118.1
`a.grad`	41.3	56.9	57.0	57.0	21.2
`scoring`	—	41.3	41.9	41.5	—
`w.grad`	39.0	24.4	24.1	—	20.9
`autograd`	131.2	191.4	190.5	190.1	76.0
Optimizer	80.7	80.7	80.7	—	81.8
Total	**374**	**498** (+33.1%)	**498** (+33.1%)	**643** (+72.0%)	

(e) LLAMA-3.2-3B, $n = 8$, $\mathsf{T} = 512$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	345.3	386.5	384.2	386.3	181.8
Backward	1108.2	1090.9	1084.7	947.5	581.2
`a.grad`	225.8	261.0	256.2	257.3	115.4
`scoring`	—	86.7	85.4	85.3	—
`w.grad`	337.8	127.6	137.7	—	172.5
`autograd`	544.5	615.6	605.3	604.9	293.2
Optimizer	235.4	235.5	235.5	—	235.4
Total	**1689**	**1713** (+1.4%)	**1704** (+0.9%)	**2332** (+38.1%)	

(f) LLAMA-3.2-3B, $n = 2$, $\mathsf{T} = 1024$.

Component	Full-Training	Layer-Wise Subset	Global Subset (1P)	Global Subset (2P)	
				Score	Grad
Forward	190.4	262.5	261.6	268.2	103.7
Backward	588.4	750.8	745.7	671.3	290.0
`a.grad`	115.7	166.6	165.0	165.6	62.5
`scoring`	—	86.3	86.1	86.1	—
`w.grad`	172.6	70.8	75.7	—	62.8
`autograd`	300.1	427.1	418.9	419.5	164.8
Optimizer	235.4	235.5	235.5	—	235.6
Total	**1014**	**1249** (+23.1%)	**1243** (+22.5%)	**1569** (+54.7%)	

Scoring cost. Table 4(b) reports the standalone cost of the four scoring mechanisms of Appendix D.3 at $n = 8$, sweeping T at fixed $m = 1$ and m at fixed $\mathsf{T} = 512$. The measurements follow the complexity analysis of Table 5. Among the exact methods, GIP is the cheapest only at short sequences with a single target example ($\mathsf{T} = 512$, $m = 1$: 29–73 ms), as its leading term predicts ($m\mathsf{T} < \sqrt{d/L}/2$). Along m, PIP overtakes GIP at $m = 4$ on SMOLLM2-360M (52 vs. 95 ms) and TINYLLAMA-1.1B (121 vs. 138 ms) and at $m = 8$ on LLAMA-3.2-3B (429 vs. 508 ms), consistent with the predicted crossover at $m \gtrsim \sqrt{d/L}/(2\mathsf{T})$. Along T, GIP's quadratic cost takes over quickly: at $\mathsf{T} = 2048$ PIP is already the cheapest exact method on all three models (122, 335, and 961 ms against GIP's 356, 519, and 1001 ms), and by $\mathsf{T} = 8192$ GIP costs 5–13× PIP. The predicted crossover from PIP to Direct at $\mathsf{T} \gtrsim \sqrt{d/L}$ (Appendix D.3.2), in contrast, does not appear within $\mathsf{T} \leq 8192$: PIP stays ahead of Direct on SMOLLM2-360M (463 vs. 575 ms) and LLAMA-3.2-3B (3880 vs. 4091 ms) and ties on TINYLLAMA-1.1B (1351 vs. 1353 ms). The two methods share the leading term $2N\mathsf{T}(d/L)$; PIP's lower-order term $n\mathsf{T}\sqrt{d/L}$ is the $[n, \mathsf{T}, \sqrt{d/L}]$ intermediate, which our fused kernel keeps in registers, whereas Direct's lower-order term $n(d/L)$ materializes n gradient matrices, a memory-bound write and read that no kernel removes. Compressed scoring is the cheapest method at every T and m (25–412 ms), and its cost is independent of m.

Scope of the efficiency claim. The per-step parity of Section 4.4 is a property of the post-training regime, large models and candidate batches, where the skipped half of the weight gradient pays for the target example; with only two candidates per step the target example is half the batch and the overhead is +21–47% (Table 15), and on launch-bound small models the fixed costs show as +8–20% even at $n = 8$. Accuracy and cost also have to be read together across backends: compressed scoring is the cheapest and leaves Global Subset Update unchanged, but costs Layer-Wise Subset Update up to 11 points on the misaligned `less-mix` pool (Appendix E.1.3, which quantifies the tradeoff). Every accuracy result in this paper and the benchmarks of Section 4.4 therefore use exact scoring; compressed scoring is the option for long contexts or many target examples, best paired with Global Subset Update or a larger sketch.