FILTER-THEN-WEIGHT: ONLINE DATA SELECTION AND REWEIGHTING FOR LLM FINE-TUNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Gradient-based data selection scores a sample by how it moves the model. Online LLM fine-tuning complicates this: utility depends on the current parameters, the effective update geometry is set by an adaptive optimizer, and per-example gradients are too large to materialize, so online rules score samples independently and weight the selected batch uniformly. We formulate online selection instead as *subset-level gradient matching*: choose non-negative weights whose composite update aligns with the target under the optimizer-induced geometry. Solved with the coupled greedy solver that gradient matching conventionally uses, this degrades on LLM gradients—the solver refits coefficients while the support is still being built, and the estimation error is comparable to the correlation gaps that decide each choice. Two conditions recover it: coefficient fitting must leave support construction, and the optimizer geometry must enter the alignment term alone, with redundancy measured in the candidates' own metric. FILTER-THEN-WEIGHT meets both. To run it inside the training step we hold per-example gradients as factorized outer products, project the two factors rather than the gradient, and contract the token axis before the feature axes; selection then costs under 9% of a step and carries no quadratic dependence on context length. Across backbones from 0.6B to 8B and two optimizers, it ranks above every online selection baseline we evaluate on both main benchmarks. An ablation shows that the gain comes from filtering and reweighting together.

1 INTRODUCTION

Data selection for large language models (LLMs) aims to curate a representative subset from massive training corpora, enabling models to achieve convergence with reduced data consumption and improved performance. While heuristic metrics—such as representation similarity (Zhang et al., 2018; Hanawa et al., 2020; Xie et al., 2023) sample learning difficulty (Li et al., 2024), and outlier structure (Chhabra et al., 2024)—offer simple filtering criteria, gradient-based methods (Pruthi et al., 2020; Xia et al., 2024) provide a theoretically grounded framework that explicitly quantifies a sample's contribution to model parameter updates (Koh & Liang, 2017; Han & Tsvetkov, 2021; Grosse et al., 2023). However, these methods generally require additional forward and backward passes to compute sample gradients, introducing severe computational overhead that scales linearly with the number of training samples and model parameters.

Gradient-based methods divide into two families (Qin et al., 2024). Influence methods score each sample by the similarity between its gradient and the validation gradient (Koh & Liang, 2017; Pruthi et al., 2020), with instruction-tuning adaptations that accommodate the Adam optimizer and variable instruction lengths (Xia et al., 2024) and an online variant that corrects a sample's influence for what has already been selected (Wang et al., 2024). Gradient matching and coreset construction instead choose a subset whose aggregate gradient approximates a reference (Killamsetty et al., 2021; Mirzasoleiman et al., 2020), at the domain level (Fan et al., 2024), over clustered gradients for task-agnostic coverage (Zhang et al., 2025), or with datamodels for pre-training (Engstrom et al., 2024).

Most of these approaches (Pruthi et al., 2020; Xia et al., 2024), and recent in-context scoring (Zhang et al., 2026), target the offline setting, where the full dataset is available and scores can be computed once. In online fine-tuning (Aljundi et al., 2019) the data arrives sequentially and decisions are made

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.

without full corpus access, so a sample's utility depends on the current parameters and selection becomes a step-wise problem rather than a single-shot one. Averaging gradients over several historical checkpoints, standard offline (Xia et al., 2024; Deng et al., 2024), is also unavailable under streaming latency and storage budgets. Finally, online selection has to respect the update rule the optimizer actually applies, whereas much of this literature (Pruthi et al., 2020; Wang et al., 2024; Fan et al., 2024) assumes a plain SGD step. Concurrent work (Wang et al., 2026) is likewise optimizer-aware and online, but targets pre-training and scores samples individually, so it does not address the interactions among jointly selected samples that subset-level reweighting creates. Appendix A discusses related work in full.

These observations suggest that online data selection should be viewed not as static sample ranking, but as shaping the next training update toward the target task—jointly accounting for both optimizer geometry and sample interactions. Motivated by this view, we propose an optimizer-aware framework for online data selection and reweighting in LLM fine-tuning. Our key idea is to formulate the problem as *subset-level gradient matching*: rather than scoring samples individually (Pruthi et al., 2020; Xia et al., 2024; Wang et al., 2026) or reweighting the whole pool without selecting (Zhao et al., 2026), we optimize a non-negative weighting over a candidate subset so that its composite gradient approximates the target-oriented update under the geometry induced by the optimizer state.

Building on this formulation, we introduce FILTER-THEN-WEIGHT(FW), a subset-level reweighting method designed to remain stable and practical at LLM scale. Two separations are central to its design. First, FW separates support construction from coefficient estimation. Existing coupled solvers refit the coefficients after every greedy addition, allowing estimation noise in projected mini-batch LoRA gradients to alter the residual and propagate into subsequent selections. We instead construct a complementary support with a whitened residual filter and perform a single non-negative least-squares solve only after the support is fixed. Second, we separate optimizer-aware alignment from pairwise redundancy: optimizer state modifies the target-alignment term, whereas redundancy is measured in the shared candidate-gradient geometry, following a second-order expansion of the target loss. Our ablations validate both separations. Across backbones from 0.6B to 8B, FW ranks above every selection baseline we evaluate on our two main benchmarks, and the gain is largest where the candidate pool carries target-relevant signal to exploit. On two further targets the candidate pool sets the ceiling: on one every method lands within about a point, on the other no selector recovers the untrained model, and ours loses the least. Finally, to run FW inside each training step, we retain per-example gradients as factorized outer products, project the two factors rather than the gradient, and contract the token axis before the feature axes. Selection then costs under 9% of a step, and the score-side memory grows linearly rather than quadratically in context length. Our code is available at `https://anonymous.4open.science/r/FWDS-647E`.

2 PROBLEM SETUP

2.1 ONLINE TRAINING DATA SELECTION

We consider a large-scale training corpus, denoted as $\mathcal{D}_{tr} := \{x_i\}$, which has mixed knowledge type and data quality. Rather than minimizing the training loss over $\mathcal{D}_{tr}$, our goal is to optimize the model performance on a *target distribution* that differs from the training one. Let $\mathcal{D}_{tar}$ denote a small validation or target dataset drawn from the downstream task distribution, and let $\mathcal{L}_{\text{tar}}(\theta) := \mathbb{E}_{x_j \sim \mathcal{D}_{tar}}[\ell(\theta, x_j)]$, where ℓ is the loss function. Note that the validation set typically is considerably smaller than the training set, i.e., $|\mathcal{D}_{tar}| \ll |\mathcal{D}_{tr}|$. Given limited validation data, our primary objective is to improve the predictive performance of a fine-tuned model on a specific target task by strategically leveraging the vast but noisy training corpus.

Online and Memory-Constrained Data Selection. Training proceeds in sequential batches. At each iteration t the algorithm sees a candidate pool $\mathcal{S}^{(t)} \subset \mathcal{D}_{\text{tr}}$ and must decide how to use it without persistent access to the full corpus; a bounded buffer of past samples may be revisited, but full replay is not available. Selection decisions are therefore made on-the-fly from the current model state.

Weighted training update. We use continuous weights $w_i \in \mathbb{R}_{\geq 0}$ to score candidate samples. Setting $w_i \in \{0, 1\}$ recovers standard subset selection, while continuous weights subsume both selection and reweighting, allowing each sample's contribution to be adaptively modulated based on its relevance to the target task. Given a candidate mini-batch $\mathcal{S}^{(t)}$, the resulting weighted training gradient at step t is $g_t(w) = \nabla \mathcal{L}_{\text{tr}}^{(t)}(\theta_{t-1}; w) = \sum_{x_i \in \mathcal{S}^{(t)}} w_i \nabla \ell(\theta_{t-1}, x_i)$, and the model parameters

are updated via $\theta_t = \theta_{t-1} - \eta_t P_t(g_t(w))$, where η_t is the learning rate and $\mathcal{P}_t$ is the possibly adaptive operator induced by the optimizer.

2.2 Step-Wise Target Loss Reduction

At each step t, our objective is to choose w to directly minimize the target loss $\mathcal{L}_{\text{tar}}(\theta_t)$ under the actual optimization dynamics used for fine-tuning. For brevity, we write $\nabla \ell_i := \nabla \ell(\theta_{t-1}, x_i)$ and $\nabla \ell_j := \nabla \ell(\theta_{t-1}, x_j)$ for training and validation gradients respectively. To connect the choice of weights w with the optimization of the target loss, following Pruthi et al. (2020), we consider the first-order Taylor expansion of loss around θ_{t-1} as $\mathcal{L}_{\text{tar}}(\theta_t) \approx \mathcal{L}_{\text{tar}}(\theta_{t-1}) + \langle \nabla \mathcal{L}_{\text{tar}}(\theta_{t-1}),\, \theta_t - \theta_{t-1} \rangle = \mathcal{L}_{\text{tar}}(\theta_{t-1}) - \eta_t \langle \nabla \mathcal{L}_{\text{tar}}(\theta_{t-1}),\, \mathcal{P}_t(g_t(w)) \rangle$, using $\theta_t - \theta_{t-1} = -\eta_t \mathcal{P}_t(g_t(w))$.

Under this approximation, minimizing the target loss after one update step is equivalent to maximizing the alignment between the weighted training and the target gradients under the optimizer-induced geometry. Consequently, the weight assignment problem at step t can be formulated as $\max_w \left\langle \nabla \mathcal{L}_{\text{tar}}(\theta_{t-1}),\, \mathcal{P}_t\big(\sum_{x_j \in \mathcal{S}^{(t)}} w_i \nabla \ell_i\big) \right\rangle$. With proofs in Appendix C.5, this formulation highlights that the optimal weights depend not only on the similarity between individual training gradients and the target gradient, but also on the underlying optimizer through $\mathcal{P}_t$. As a result, the data weighting problem is inherently optimizer-aware and evolves dynamically across training steps. We also formulate the problem with second-order Taylor expansion approximation in Appendix C.6.

Optimizers ($\mathcal{P}_t$). We instantiate $\mathcal{P}_t$ with three prevalent optimizers: (i) *SGD* (Robbins & Monro, 1951), where $\mathcal{P}_t(g) = g$; (ii) *Adam* (Kinga et al., 2015), where $\mathcal{P}_t(g) = \frac{\hat{m}_t(g)}{\sqrt{\hat{v}_t(g)} + \epsilon}$ with bias-corrected moment estimates $\hat{m}_t(g) = \frac{\beta_1 m_{t-1} + (1-\beta_1) g}{1-\beta_1^t}$ and $\hat{v}_t(g) = \frac{\beta_2 v_{t-1} + (1-\beta_2) g^2}{1-\beta_2^t}$, decay coefficients $\beta_1, \beta_2 \in [0, 1)$, a small constant $\epsilon > 0$, and element-wise square; and (iii) *AdamW* (Loshchilov & Hutter, 2019), which augments Adam with a decoupled weight decay term, $\mathcal{P}_t(g) = \frac{\hat{m}_t(g)}{\sqrt{\hat{v}_t(g)} + \epsilon} + \lambda \theta_{t-1}$, where $\lambda \geq 0$ is the decay coefficient and θ_{t-1} the current parameters. SGD serves as the linear reference case, under which the selection objective collapses to additive gradient similarity—the assumption prior scoring rules make implicitly (Section 3.1); every experiment we report uses Adam or AdamW.

3 Methodology

We now turn the step-wise objective into an algorithm that runs under the computational constraints of LLM fine-tuning.

3.1 Subset Utility: From Additive Scoring to Gradient Matching

At iteration t, we quantify the utility of a candidate mini-batch $\mathcal{S}^{(t)}$ by its one-step impact on the target loss. Approximating the target objective using a validation mini-batch $\mathcal{S}_{\text{val}}^{(t)}$, the averaged validation gradient is $\nabla \ell_{\text{val}} := \frac{1}{|\mathcal{S}_{\text{val}}^{(t)}|} \sum_{x_j \in \mathcal{S}_{\text{val}}^{(t)}} \nabla \ell_j$. Following the formulation in Section 2.2, selecting and weighting training samples amounts to maximizing the alignment between the weighted training gradient and the validation gradient under the optimizer-induced operator:

$$\max_{w} \left\langle \nabla \ell_{\text{val}}, \mathcal{P}_t\big(g(w)\big) \right\rangle, \qquad g(w) := \textstyle\sum_{x_i \in \mathcal{S}^{(t)}} w_i \nabla \ell_i. \tag{1}$$

Why additive individual scoring fails. When the optimizer is linear ($\mathcal{P}_t(g) = g$) and weights are restricted to $w_i \in \{0, 1\}$ with $\sum_i w_i = k$, Eq. 1 decomposes additively across samples, recovering the gradient-similarity criterion of (Pruthi et al., 2020; Xia et al., 2024): $U(\mathcal{S}^{(t)}; \mathcal{S}_{\text{val}}^{(t)}) = \sum_{x_i \in \mathcal{S}^{(t)}} \langle \nabla \ell_{\text{val}}, \nabla \ell_i \rangle$. Three structural problems emerge once we leave this restricted regime. First, 0-1 selection alone cannot recover the right mixture coefficients under continuous weights. Second, two samples with identical individual scores can be either complementary or redundant depending on their pairwise inner product, and individual scoring cannot distinguish the two cases. Third, when $\mathcal{P}_t$ is nonlinear (as in Adam), the alignment of the weighted sum is not the sum of individual alignments.

Reformulation as gradient matching. We therefore treat selection and weighting as a *subset-level gradient matching* problem: find a small set of candidates and non-negative weights whose

optimizer-transformed composite gradient best matches the target gradient. This yields

$$\min_{w \geq 0} \quad \left\| \nabla \ell_{\text{val}} - \mathcal{P}_t\big(g(w)\big) \right\|_2^2 + \lambda \|w\|_2^2. \tag{2}$$

Although gradient-matching objectives have appeared in offline coreset selection (Killamsetty et al., 2021; Mirzasoleiman et al., 2020), Eq. 2 differs in combining non-negative sample weights with an optimizer-aware mapping $\mathcal{P}_t$. Non-negativity restricts the composite update to the cone of the candidate gradients. Without it, least-squares matching on highly correlated gradients fits the target by cancelling large opposing coefficients (Slawski & Hein, 2013), and a negative weight reverses the descent direction of its own training example. Requiring $w_i \geq 0$ rules that out while still letting unhelpful candidates take zero weight; removing it makes training diverge (Section 4.4).

From alignment to subset-level matching. After dropping the constant $\|\nabla \ell_{\text{val}}\|_2^2$, Eq. 2 equals

$$\min_{w \geq 0} \; -2 \underbrace{\langle \nabla \ell_{\text{val}}, \mathcal{P}_t(g(w)) \rangle}_{\text{target-loss reduction in Eq. 1}} + \left\| \mathcal{P}_t(g(w)) \right\|_2^2 + \lambda \|w\|_2^2. \tag{3}$$

Matching therefore keeps the alignment term that first-order selection uses and adds control of the update magnitude. Under the linearized optimizer mapping of Section 3.2 that second term is quadratic in w, so it is where pairwise interactions enter; the same structure follows from a second-order expansion of the target loss under an isotropic Hessian (Appendix C.3).

3.2 Filter-then-Weight: A Decoupled Two-Stage Algorithm

A standard approximation to Eq. 2 is to couple greedy subset selection with coefficient refitting. For example, GRAD-MATCH applies Orthogonal Matching Pursuit (OMP), re-solving the weights over the selected support whenever a new sample is added (Killamsetty et al., 2021). Classical OMP analyses indicate that stable greedy support recovery requires sufficiently separated dictionary atoms and a signal-to-noise ratio large enough to distinguish their residual correlations (Cai & Wang, 2011; Ding et al., 2012). Neither condition is guaranteed for stochastic, projected LLM gradients: candidate gradients can be highly correlated, while mini-batch estimation and projection errors can be comparable to the correlation gaps that determine each greedy choice. This motivates avoiding repeated coefficient refitting during support construction.

We therefore decouple the two operations into a robust *filtering* stage that identifies a candidate backbone using unit weights, and a precise *weighting* stage that solves for the final coefficients only over the filtered set. The intuition is that filtering with unit weights is a low-frequency operation insensitive to small gradient perturbations, while the weighting stage operates on a much smaller and well-conditioned problem—each stage is exposed to the noise level it can handle. We build on the expansion of Eq. 2 into a precomputed Alignment Vector $\mathbf{b}$ and Gram Matrix $\mathbf{G}$ (detailed in Appendix B.1), which reduces both stages to small-scale operations on these reusable components.

Preparation: normalizing candidate geometry and the target direction. Raw gradients are poorly conditioned for matching. Coordinate scales vary by orders of magnitude across LoRA blocks, a component shared by the whole pool obscures what distinguishes one candidate from another, and Adam rescales the effective update through its optimizer state. The first two are properties of the candidates and the third is a property of the target, so we treat them separately.

For the candidate pool we standardize each coordinate under statistics taken over the current pool $\mathcal{S}^{(t)}$. Writing $\mu^{(t)}, \sigma^{(t)}$ for the coordinatewise mean and standard deviation, $\widehat{\nabla \ell_i}^{(t)} = (\nabla \ell_i^{(t)} - \mu^{(t)})/\sigma^{(t)}$, and candidate redundancy is measured in that space, $\widehat{G}_{ij}^{(t)} = \langle \widehat{\nabla \ell_i}^{(t)}, \widehat{\nabla \ell_j}^{(t)} \rangle$. The scaling stops high-variance coordinates from deciding the similarity on their own; the centering removes what every candidate has in common, which carries no information about which of them to pick.

For the target we additionally account for the optimizer. Freezing Adam's second-moment denominator at the state before step t linearizes the preconditioner as $\mathcal{P}_t(\nabla \ell) \approx \mathbf{D}_{t-1} \odot \nabla \ell$, $\mathbf{D}_{t-1} = 1/(\sqrt{\hat{v}_{t-1}} + \epsilon)$, and the alignment vector becomes

$$\widehat{b}_i^{(t)} = \left\langle \big(\mathbf{D}_{t-1} \odot \nabla \ell_{\text{val}}^{(t)}\big)/\sigma^{(t)},\ \widehat{\nabla \ell_i}^{(t)} \right\rangle. \tag{4}$$

Table 1: **Time and space complexity analysis**. Cost are computed and compared over one selection step over L adapted linear layers, with N candidates processed in α micro-batches, V validation examples, and sequence length T. D_1, D_2 denote the original layer widths and d_1, d_2 their projected dimensions, with fixed projection budget $d_1 d_2$; C_{bwd} denotes one micro-batch backward pass. Space reports score-side memory only, excluding the model's forward/backward working set.

		Time	Space
Representation	Per-example gradients	$\mathcal{O}((N+V)C_{\text{bwd}})$	$\mathcal{O}(LND_1D_2)$
	Rank-1 factors	$\mathcal{O}(\alpha C_{\text{bwd}})$	$\mathcal{O}(LND_1D_2)$
	+ projected	$+\,\mathcal{O}(LT(N+V)(D_1d_1+D_2d_2))$	$\mathcal{O}(LT(N+V)(d_1+d_2)+LNd_1d_2)$
Alignment $\mathbf{b}$	Ghost	$\mathcal{O}(LNVT^2(d_1+d_2))$	$\mathcal{O}(NVT^2)$
	Ours	$\mathcal{O}(LT(N+V)d_1d_2)$	$\mathcal{O}(LNd_1d_2)$
Gram $\mathbf{G}$	Ghost	$\mathcal{O}(LN^2T^2(d_1+d_2))$	$\mathcal{O}(N^2T^2)$
	Ours	$\mathcal{O}(Ld_1d_2(NT+N^2))$	$\mathcal{O}(LNd_1d_2+N^2)$

The two transformations enter asymmetrically on purpose. Division by $\sigma^{(t)}$ defines the metric in which alignment and redundancy are both measured, so it appears on both sides. Centering removes only what the candidate pool shares, and the validation target has no candidate mean to subtract. $\mathbf{D}_{t-1}$ describes how the optimizer will move, a property of the direction we are trying to reach and not of how two candidates overlap, so it enters $\widehat{b}$ alone; Section 4.4 shows that dropping it and matching on raw gradients throughout costs 2.47 F1 at the peak and 2.98 at the end of the budget, and inflates the across-seed deviation fivefold. Freezing the denominator avoids recomputing optimizer-dependent statistics for every candidate, and closely preserves the rankings and selected supports obtained with the full preconditioner (Spearman 0.95–0.98, support overlap 81%–91%; Appendix E.11).

Stage 1: Greedy residual filtering. Given a candidate pool $\mathcal{S}^{(t)}$ and a budget k, we initialize $\mathcal{S}' = \varnothing$ and iteratively select

$$i^* = \arg\max_{i \notin \mathcal{S}'} \left(\widehat{b}_i^{(t)} - \sum_{j \in \mathcal{S}'} \widehat{G}_{ij}^{(t)} \right), \qquad \mathcal{S}' \leftarrow \mathcal{S}' \cup \{i^*\}, \tag{5}$$

until $|\mathcal{S}'| = k$. This is OMP without the inner weight-resolve step: holding the intermediate weights at unity scores each candidate by its alignment with the remaining residual while penalizing similarity to previously selected samples. Stage 1 therefore retains residual-based diversity modeling without allowing noisy intermediate coefficient estimates to alter subsequent selection decisions. A controlled intervention isolates what the residual term buys: stripping candidate–candidate redundancy from the pool makes greedy and top-k select identical batches, while stripping gradient magnitude leaves their disagreement intact, so the gain over top-k comes from diversity rather than from implicitly discarding large-norm gradients (Appendix E.7).

Stage 2: Non-negative reweighting. Given the fixed set $\mathcal{S}'$, we solve

$$w^* = \arg\min_{w \geq 0} \; w^\top \widehat{\mathbf{G}}_{\mathcal{S}'}^{(t)} w - 2\widehat{\mathbf{b}}_{\mathcal{S}'}^{(t)\top} w + \lambda\, \bar{g}^{(t)} \|w\|_2^2, \qquad \bar{g}^{(t)} = \tfrac{1}{k} \textstyle\sum_{i \in \mathcal{S}'} \widehat{G}_{ii}^{(t)}. \tag{6}$$

Scaling the ridge by the mean Gram diagonal makes λ dimensionless, so one value transfers across steps and backbones whose gradient scales differ by orders of magnitude. Because $|\mathcal{S}'|$ equals the per-step batch size, this optimization is negligible relative to gradient computation. The non-negativity constraint follows Section 3.1, while optimizing the coefficients globally over a fixed, small set maintains high fidelity to the optimizer-aware target. The model is then updated with the composite gradient $\bar{\nabla}\ell^{(t)} = \sum_{i \in \mathcal{S}'} w_i^* \nabla \ell_i^{(t)}$.

Our ablations (Section 4.4) show that recoupling the two operations, as standard OMP does, recovers the unstable behavior of GRAD-MATCH, while replacing the residual filter with top-k admits redundant samples and degrades over the trajectory. Algorithm 2 gives the complete in-step procedure.

3.3 SCALING GRADIENT STATISTICS TO LONG-CONTEXT LLMS

Filter-then-Weight requires two gradient statistics at every selection step: the candidate–target alignment vector $\mathbf{b}$ and the candidate–candidate Gram matrix $\mathbf{G}$. The alignment vector $\mathbf{b}$ could be

computed from transient rank-1 factors without storing candidate gradients, while the Gram matrix $\mathbf{G}$ cannot: it explicitly compares candidates against one another, so a representation of the entire candidate pool must remain available throughout selection. The computational problem is therefore to make this resident representation small enough for LLM fine-tuning, while avoiding the T^2 token-pair expansion incurred by standard ghost inner products on long sequences.

Factors, projected before they are combined. For a linear layer with weight $W \in \mathbb{R}^{D_1 \times D_2}$, including LoRA's two projections, the per-example gradient is an outer product of the input activation a_i and the pre-activation gradient g_i, so inner products factorize as in the ghost dot-product (Bu et al., 2023; Wang et al., 2025b):

$$\langle \nabla \ell_i, \nabla \ell_j \rangle = (a_i^\top a_j)(g_i^\top g_j). \tag{7}$$

Forward and backward hooks capture both factors for a whole micro-batch in one backward pass, so the pool costs α passes rather than $N+V$. We then compress with Gaussian random projection (Xia et al., 2024; Wang et al., 2025b), which preserves inner products by the Johnson–Lindenstrauss lemma (Johnson et al., 1984), and apply it to the two factors separately rather than to a gradient we would first have to form. The projected dimensions are chosen under a fixed product budget $d_1 d_2$, each no smaller than the LoRA rank. This buys the residency $\mathbf{G}$ requires, since the whole pool must stay comparable at once. The second moment behind $\mathbf{D}_{t-1}$ and the statistics μ, σ are maintained in the projected space, as squaring does not commute with projection, and the pool is read in α micro-batches to cap peak memory.

Aggregating the target before touching candidates. For sequences of length T the factors become matrices $a_i \in \mathbb{R}^{T \times d_1}$ and $g_i \in \mathbb{R}^{T \times d_2}$, with $\nabla \ell_i = a_i^\top g_i$ summing the per-token outer products. Evaluated in ghost form, Eq. 7 contracts the feature axes first and leaves a $T \times T$ token-pair matrix for every example pair, so context length enters quadratically. It also fixes the pairing: both factors of $\langle a_j a_i^\top, g_j g_i^\top \rangle_F$ carry the validation index j, so the target cannot be summed out and $\mathbf{b}$ costs NV evaluations. We contract the token axis first, which frees the target to be aggregated once per step:

$$\langle \nabla \ell_i, \nabla \ell_{\text{val}} \rangle = \Big\langle \hat{a}_i^\top \hat{g}_i, \sum_{x_j \in \mathcal{S}_{\text{val}}^{(t)}} \hat{a}_j^\top \hat{g}_j \Big\rangle, \tag{8}$$

where $\hat{a}, \hat{g}$ denote projected factors. The validation sum is a single $d_1 \times d_2$ matrix per layer and is reused across the entire candidate pool. Consequently, $\mathbf{b}$ requires $N + V$ sequence contractions rather than NV pairwise token contractions, reducing the cost to $\mathcal{O}(LT(N+V)d_1 d_2)$ and avoiding any $T \times T$ intermediate.

Long-context memory scaling. As Table 1 shows, the ghost score tensors grow as NVT^2 for $\mathbf{b}$ and N^2T^2 for $\mathbf{G}$, whereas our score-side storage contains no quadratic dependence on T. At long context, ghost therefore becomes score-memory bound, while our computation remains below the model's own forward/backward working set. In practice, the limiting memory then shifts to model-side tensors such as the vocabulary-sized logits buffer rather than gradient-statistic computation. Appendix D.1 gives the full derivation and Appendix E.13 reports the measured memory frontier.

4 Experiments

We evaluate Filter-then-Weight on instruction-following benchmarks to answer four questions: **effectiveness** and **data efficiency** against existing selection methods (Section 4.2); **generalization** across backbone scale, optimizer, and target task (Section 4.3); and **mechanism**, i.e., which components drive the gains (Section 4.4).

4.1 Setup and Baselines

Data and benchmarks. The training pool is the Open-Instruct corpus (Lambert et al., 2024), aggregating FLAN v2 (Longpre et al., 2023), Chain-of-Thought (Wei et al., 2022), Dolly (Conover et al., 2023), and OASST1 (Köpf et al., 2023) (≈270k pairs), with any sample overlapping the target benchmarks removed. We evaluate on **MMLU** (Hendrycks et al., 2021) (5-shot accuracy over 57 subjects) and **TyDiQA** (Clark et al., 2020) (1-shot macro-F1 over 11 languages).

Training protocol. Backbones are Llama-3.2-1B, Llama-3.2-3B and Llama-3.1-8B (Grattafiori et al., 2024), and Qwen3-0.6B (Yang et al., 2025). All selection methods train under the same 5% data budget, realized as the checkpoint nearest 5% of the 270,679-example pool: step 1,600

Table 2: **Best-of-run performance.** For each seed we take the checkpoint with the highest target-benchmark score, applied identically to every method, and report mean$^{\pm \text{std}}$ over seeds. Appendix E.4 reads the same runs at the final checkpoint instead and reaches the same ordering. The TyDiQA block additionally covers a different optimizer (AdamW) and larger Llama-family backbones. Origin is a single evaluation of the untrained backbone rather than a mean over seeds, so it carries no deviation in any row. **Bold** marks the best result, underline the second best.

		References		*Selection only*			*Coupled*	*Ours*
Dataset	Backbone (Optimizer)	Origin	Full	TracIn	LESS	GREATS	GRAD-MATCH	Filter-then-Weight
MMLU (Acc)	Llama-3.2-1B (Adam)	28.13	29.09 $^{\pm 1.01}$	29.24 $^{\pm 0.24}$	29.74 $^{\pm 0.07}$	30.07 $^{\pm 0.37}$	30.14 $^{\pm 0.41}$	**30.93** $^{\pm \mathbf{0.03}}$
	Qwen3-0.6B (Adam)	45.86	**46.64** $^{\pm \mathbf{0.17}}$	46.45 $^{\pm 0.08}$	46.56 $^{\pm 0.28}$	46.06 $^{\pm 0.42}$	45.72 $^{\pm 0.62}$	46.63 $^{\pm 0.40}$
TyDiQA (F1)	Llama-3.2-1B (Adam)	12.69	42.47 $^{\pm 0.30}$	47.01 $^{\pm 1.43}$	47.21 $^{\pm 2.58}$	47.50 $^{\pm 0.56}$	45.35 $^{\pm 1.35}$	**48.86** $^{\pm \mathbf{1.67}}$
	Llama-3.2-1B (AdamW)	12.69	42.52 $^{\pm 0.41}$	47.14 $^{\pm 0.72}$	45.83 $^{\pm 1.03}$	46.45 $^{\pm 1.02}$	45.78 $^{\pm 1.18}$	**48.24** $^{\pm \mathbf{1.12}}$
	Llama-3.2-3B (Adam)	19.04	60.42 $^{\pm 1.64}$	61.90 $^{\pm 0.61}$	61.53 $^{\pm 0.11}$	60.47 $^{\pm 0.10}$	61.10 $^{\pm 0.73}$	**62.59** $^{\pm \mathbf{0.84}}$
	Llama-3.1-8B (Adam)	26.49	67.70 $^{\pm 0.23}$	67.45 $^{\pm 0.08}$	66.57 $^{\pm 1.74}$	67.86 $^{\pm 0.85}$	67.52 $^{\pm 1.00}$	**69.09** $^{\pm \mathbf{0.21}}$
	Qwen3-0.6B (Adam)	20.94	43.65 $^{\pm 0.61}$	45.17 $^{\pm 0.97}$	45.01 $^{\pm 0.95}$	45.25 $^{\pm 0.43}$	44.23 $^{\pm 1.33}$	**46.29** $^{\pm \mathbf{0.62}}$

at k=8 and step 2,200 at k=6, or 12.8k and 13.2k selected examples. The step count therefore varies with k while the data consumed does not. Codes for this paper are available in `https://anonymous.4open.science/r/FWDS-647E`. Each step draws k samples from a candidate pool of size $N = \alpha k$ ($\alpha = 4$), with validation gradients estimated analogously from $V = \alpha k_{\text{val}}$; due to memory we use k=8, k_{val}=4 for Llama and k=6, k_{val}=2 for Qwen. Sequences are truncated at 1,024 tokens, checkpoints are written every 200 steps, and every configuration is run with three seeds. Section 4.3 varies three axes while holding everything else fixed: backbone scale (Llama-3.2-3B/Llama-3.1-8B), optimizer (AdamW, decoupled weight decay 0.01), and target task (BBH (Suzgun et al., 2022), IFEval (Zhou et al., 2023)); for in-domain instruction following we swap in the much smaller Tulu-3 personas-IF pool (Ge et al., 2025), where a 25% budget is needed to reach a comparable number of steps. Details are in Appendix E.1.

Baselines. *Origin* (untrained backbone) and *Full* (SFT on all data) serve as references. Three selection-only baselines apply uniform weights: *TracIn* (Pruthi et al., 2020) scores samples by gradient similarity to the validation set; *LESS* (Xia et al., 2024) matches Adam-preconditioned gradient embeddings; *GREATS* (Wang et al., 2024) greedily accounts for intra-batch interactions. TracIn and LESS are adapted to the online setting by computing gradients on-the-fly without checkpoints: the checkpoint ensemble both rely on needs the training history and full-corpus access that the online setting of Section 2 does not provide, so what we compare is each method's scoring rule given the same gradients at the same step as ours. *GRAD-MATCH* (Killamsetty et al., 2021) jointly selects and weights via OMP and is the closest prior formulation to ours.

4.2 Main Results

Table 2 reports the best checkpoint per seed at a 5% data budget. Filter-then-Weight is the top selection method in all seven cells and beats full-data training in six of them. The margins run from +0.07 on Qwen3-0.6B/MMLU to +1.36 on Llama-3.2-1B/TyDiQA, against across-seed deviations that reach 1.67. Winning all seven has probability 0.008 under a sign test and Fisher's combination of the seven tests gives $p = 0.004$ (Appendix E.5). The best checkpoint is picked on the target benchmark, and we also report the final checkpoint, and the ranking in every row is unchanged (Appendix E.4). The one cell we lose is the one with nothing to select: on Qwen3-0.6B/MMLU every trained method lands between 45.7 and 46.6, ours included, and full-data training is the best of them by 0.01. That pool is already aligned with the target, so no subset of it is much better than the average one, and Appendix E.12 bounds what any selector could gain there by the spread of the candidate–target alignment scores.

Weighting helps, but only when it is kept out of the search. TracIn, LESS and GREATS each pick a batch and then train on it with equal weights. Ours solves for the weights instead, and that is where its margins come from. GRAD-MATCH also solves for weights, but re-solves them after every greedy addition. On Llama-3.2-1B/TyDiQA that costs it 3.51 F1 against ours and leaves it 1.66 below even the weakest selection-only baseline, so the coupling does not merely fail to help: it gives back the gain that reweighting was supposed to buy. Separating the two operations, as in Section 3.2, keeps the gain.

Table 3: Multi-seed ablation over all variants (Llama-3.2-1B/TyDiQA, mean $^{\pm\text{std}}$ over three seeds). Every variant is read at the same two checkpoints: the best one within the budget, and the last one.

Method	Best F1	Final F1
1. Hard Filter+Reweight	48.72 $^{\pm 1.86}$	45.08 $^{\pm 1.17}$
2. Optimizer-Aware Filter Only	47.21 $^{\pm 2.58}$	46.88 $^{\pm 0.91}$
3. Vanilla Filter Only	47.01 $^{\pm 1.43}$	47.01 $^{\pm 1.43}$
4. Vanilla Filter+Reweight	46.39 $^{\pm 6.13}$	45.69 $^{\pm 5.34}$
5. Unbounded Weights	40.89 $^{\pm 1.34}$	13.81 $^{\pm 10.74}$
6. Token-Level Score	39.05 $^{\pm 10.65}$	27.38 $^{\pm 14.47}$
Ours	**48.86** $^{\pm 1.67}$	**48.67** $^{\pm 2.68}$

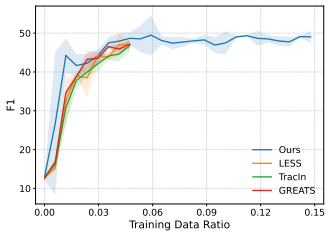


Figure 1: TyDiQA F1 against cumulative training data ratio, Llama-3.2-1B. Baselines are drawn over the first 5% of the pool; ours is continued to 15% to show the curve has flattened.

Data efficiency. Figure 1 plots TyDiQA F1 against the cumulative training data ratio. Filter-then-Weight leads at seven of the nine ratios the methods share, and the margin is widest between 1% and 2% of the pool, where it reaches 9 F1 and the choice of examples matters most; by 5% the selectors are within two points of one another. Continuing our run to 15% gains under a point over its own 5% checkpoint, which is where the budget sits.

4.3 Generalization Across Scale, Optimizer, and Target

Larger backbones. The TyDiQA block of Table 2 extends the comparison to 3B and 8B Llama backbones. Filter-then-Weight remains the strongest method at both scales. Full-data SFT becomes a much harder reference at 8B—three of the four selection baselines fall below it—while ours still clears it by 1.39 F1 on 5% of the corpus. At larger scales, several selection baselines fall below full-data training, whereas Filter-then-Weight retains its advantage, suggesting that subset-level construction remains useful as the backbone grows.

Optimizer. The preconditioner of Section 3.2 depends only on the adaptive moments, which AdamW shares with Adam, so the selection rule can transfer unchanged. Ours remains the highest F1 under both (Table 2), whereas baselines shift inconsistently.

Beyond knowledge QA. Appendix E.6 (Table 9) adds BBH (reasoning) and IFEval (instruction following) and varies the training pool; three regimes emerge. On BBH the general pool is already aligned with the target, so all selectors land within 1.2 points and ours is within 0.14 of the best. On IFEval with the general Open-Instruct pool, no method exceeds the untrained model because the pool carries no instruction-compliance signal; what differs is how much existing capability is lost—13.9 for Full, 4.0–5.0 for the selection baselines, and 2.1 for ours, since non-negative matching can drive off-target candidates to zero weight. With the in-domain Tulu-3-IF pool, ours is the strongest selector (+1.25 points over the runner-up) and reaches 98% of full-data performance on a quarter of the budget, leading outright at matched step counts.

Other paradigms. ICA (Zhang et al., 2026) scores candidates once, offline, by in-context learning, and ADAPT (Zhao et al., 2026) reweights the whole pool without selecting; neither tracks the model or optimizer as training proceeds. As shown in Appendix E.10, both fall below our method.

Cost. Selection takes 7.4–8.2% of a training step from 1B to 8B: the greedy filter and the NNLS solve are 3.5–4.0% of the step and forming the two statistics they read is the rest. Every selection method is cheaper than data-matched full SFT (0.65–0.73×), ours the most expensive of them by 0.06–0.08× of that reference. What the factorized representation buys is not a cheaper step but a longer feasible context: contracting the token axis first removes the quadratic term that a feature-first contraction carries (Appendix E.13). Cost grows linearly in the candidate-pool size while accuracy is stable across a fourfold pool increase (Appendix E.8, E.9).

4.4 Ablation Studies and Hyperparameter Analysis

We ablate Filter-then-Weight on Llama-3.2-1B/TyDiQA over three seeds (Table 3). Variants 1–4 form a factorial design over the filter (greedy vs. top-k), the presence of NNLS reweighting, and optimizer-aware preconditioning, so each choice can be read with the others held fixed; variants 5–6 remove the weight constraint and the sequence-level aggregation. Variant definitions, and the correspondence between two of them and the TracIn/LESS baselines, are in Appendix E.3.

Two constraints are necessary. *Unbounded Weights* and *Token-Level Score* collapse across all seeds, ending 34.9 and 21.3 F1 below ours at the final checkpoint with the largest across-seed variance of any variant. Without the non-negativity constraint in Eq. 2, the solver exploits gradient co-linearity by subtracting opposing vectors, pushing the optimizer away from training samples (Section 3.1); restricting similarity to identical token positions discards sequence-level structure.

Reweighting needs optimizer-aware preconditioning. On raw gradients, adding NNLS reweighting hurts (*Vanilla Filter+Reweight* $<$ *Vanilla Filter Only*, on both readings): the solver overfits to high-variance directions misaligned with the optimizer's update geometry, and its across-seed deviation rises above five points. With a preconditioned alignment vector the same weighting stage is worth 2.47 at the peak and 2.98 at the end of the budget (Ours over *Vanilla Filter+Reweight*), so the linearized target correction of Section 3.2 is what makes the weighting stage usable at all.

Neither stage carries the result alone. Holding the filter at top-k, adding NNLS (*Hard Filter+Reweight* over *Optimizer-Aware Filter Only*) raises the peak by 1.51 but *costs* 1.80 by the end of the budget: top-k admits redundant candidates, and a solver obliged to zero them out spends batch capacity doing so. Holding the weighting at NNLS, replacing top-k with the residual filter (Ours over *Hard Filter+Reweight*) is worth only 0.14 at the peak but 3.59 at the end. Preconditioning behaves the same way: 0.20 when added to top-k without reweighting (*Optimizer-Aware* over *Vanilla Filter Only*), 2.47 when added under greedy filtering with NNLS. None of the three components is sufficient on its own; the gain emerges only when greedy filtering, NNLS reweighting, and optimizer-aware preconditioning are combined, consistent with the decoupled design in Section 3.2. GRAD-MATCH provides an additional contrast: refitting the weights inside the greedy loop eliminates the gain.

Table 4: Sensitivity to the projection budget on Llama-3.2-1B/MMLU. d is the dimension of the LoRA gradient projection; the d=256 row is the configuration of Table 2.

d	Best Acc.	Final Acc.
64	30.21 $\pm$0.49	28.36 $\pm$0.57
128	30.17 $\pm$0.26	28.67 $\pm$0.39
256	**30.93** $\pm$**0.03**	**30.93** $\pm$**0.03**

Projection dimension. Table 4 sweeps the per-layer projection budget. Accuracy is flat between d=64 and d=128 and rises at d=256 on both readings, by 0.72 at the peak and 2.26 at the end, so we use 256 throughout; that row is the main-table configuration.

5 Conclusion

We have introduced *Filter-then-Weight*, a framework for online data selection and reweighting in LLM fine-tuning that reformulates the problem as subset-level gradient matching with non-negative weights. We model batch utility as a *composite* update whose alignment with the target depends jointly on which samples are selected and how they are weighted, and decouple this into a robust filtering stage and a precise weighting stage—a design choice that, our ablations show, is itself the principal driver of gains on noisy LLM gradients. To keep the framework practical for long context, we further develop a factorized rank-1 gradient representation with sequence-length-aware reordering. Across backbones from 0.6B to 8B and two optimizers, Filter-then-Weight ranks above every online selection baseline in all seven cells of the main comparison under the same data budget, and holds its ground on two further targets where the candidate pool leaves no selector any headroom; selection itself costs under 9% of a training step.

Limitations. Every configuration is run with three seeds, the usual budget at this scale, which resolves the ablation separations but not the roughly one-point margins over the strongest baselines; we therefore read the baseline comparison across cells rather than row by row (Appendix E.5). Selection cannot create signal that the candidate pool does not contain. When the pool is far from the target, as in our general-pool IFEval setting, no method—ours included—recovers the untrained model's performance; subset-level matching only limits the damage. A measurable diagnostic that would predict this regime before training is discussed in Appendix E.12, but we treat it as an interpretive proxy rather than a validated predictor. Natural extensions include reusing a bounded buffer of historical gradient trajectories across steps, which stays within the online constraint while trading memory and time for richer training dynamics, analogous to common practice in the offline setting (Xia et al., 2024; Deng et al., 2024).

AI USE STATEMENT

In this work, we used generative AI tools for the following tasks whose disclosure ICLR requires. *Implementing methods*: the selection algorithm, the gradient machinery beneath it, and the analysis scripts were written, debugged and audited with AI assistance. *Designing and giving feedback on methodology and experiments*: the choice of ablation variants, the diagnostics reported in the appendix, and which controls to run were developed in dialogue with AI tools. *Interpreting results*: the statistical tests in Appendix E.5, the reading of the ablation as a factorial design, and the reading of Figure 1 were produced with AI assistance and then checked by the authors against the underlying job outputs.

Additionally, we used generative AI tools for tasks including: creating and editing software code; preparing the released artifact; drafting and editing parts of this paper, including its structure, its readability, and the wording in which the authors' claims are stated. We have reviewed all AI-assisted work. AI-written code was read and tested by the authors, including a unit test that pins the contraction used to form the validation target, and every claim, number and citation in the paper was checked against the runs that produced it. We take responsibility for the final content of this work, including text, claims and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies data selection for supervised fine-tuning. All training pools and evaluation benchmarks are publicly released datasets, used under the licenses listed in Appendix E.2; we collected no new data and the work involves no human subjects, private data, or personally identifiable information.

The method selects which existing examples a model trains on, so it inherits whatever biases the candidate pool carries and can concentrate them: a subset chosen to match a target gradient is, by construction, less diverse than the pool it came from. Our validation targets are task-accuracy benchmarks, and nothing in the objective rewards fairness, safety, or coverage of under-represented groups. Applying it where those properties matter would require targets that encode them. The same mechanism could in principle be pointed at a harmful target, but it offers no capability that ordinary fine-tuning on a curated subset does not.

REPRODUCIBILITY STATEMENT

The selection objective and its assumptions are stated in Sections 2 and 3; complete proofs, the OMP sub-routine, and the gradient-factorization derivations are given in the appendix. Datasets, model checkpoints, data budgets, optimizer settings, and the evaluation protocol are described in Section 4, with the remaining hyperparameters listed in the appendix. The selection algorithm, the gradient machinery beneath it, the training loop that calls it, and the evaluation harness for the three reported benchmarks are released as anonymized code at `https://anonymous.4open.science/r/FWDS-647E`; the environment flags that affect numerics are documented there and default to the configuration these results were produced with.

REFERENCES

Rahaf Aljundi, Min Lin, Baptiste Goujaud, and Yoshua Bengio. Gradient based sample selection for online continual learning. *Advances in neural information processing systems*, 32, 2019.

Aleksandar Botev, Hippolyt Ritter, and David Barber. Practical gauss-newton optimisation for deep learning. In *International Conference on Machine Learning*, pp. 557–565. PMLR, 2017.

Zhiqi Bu, Yu-Xiang Wang, Sheng Zha, and George Karypis. Differentially private optimization on large model at small cost. In *International Conference on Machine Learning*, pp. 3192–3218. PMLR, 2023.

T Tony Cai and Lie Wang. Orthogonal matching pursuit for sparse signal recovery with noise. *IEEE Transactions on Information theory*, 57(7):4680–4688, 2011.

Anshuman Chhabra, Bo Li, Jian Chen, Prasant Mohapatra, and Hongfu Liu. Outlier gradient analysis: Efficiently improving deep learning model performance via hessian-free influence functions. *CoRR*, 2024.

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 ty pologically di verse languages. *Transactions of the Association for Computational Linguistics*, 8: 454–470, 2020.

Mike Conover, Matt Hayes, Ankit Mathur, Jianwei Xie, Jun Wan, Sam Shah, Ali Ghodsi, Patrick Wendell, Matei Zaharia, and Reynold Xin. Free dolly: Introducing the world's first truly open instructiontuned llm. 2023.

Zhiwei Deng, Tao Li, and Yang Li. Influential language data selection via gradient trajectory pursuit. *arXiv preprint arXiv:2410.16710*, 2024.

Jie Ding, Laming Chen, and Yuantao Gu. Perturbation analysis of orthogonal matching pursuit. *IEEE Transactions on Signal processing*, 61(2):398–410, 2012.

Logan Engstrom, Axel Feldmann, and Aleksander Madry. Dsdm: Model-aware dataset selection with datamodels. In *International Conference on Machine Learning*, pp. 12491–12526. PMLR, 2024.

Runa Eschenhagen, Alexander Immer, Richard Turner, Frank Schneider, and Philipp Hennig. Kronecker-factored approximate curvature for modern neural network architectures. *Advances in Neural Information Processing Systems*, 36:33624–33655, 2023.

Simin Fan, Matteo Pagliardini, and Martin Jaggi. Doge: domain reweighting with generalization estimation. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 12895–12915, 2024.

Tao Ge, Xin Chan, Xiaoyang Wang, Dian Yu, Haitao Mi, and Dong Yu. Scaling synthetic data creation with 1,000,000,000 personas, 2025. URL `https://arxiv.org/abs/2406.20094`.

Aaron Grattafiori et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Roger Grosse, Juhan Bae, Cem Anil, Nelson Elhage, Alex Tamkin, Amirhossein Tajdini, Benoit Steiner, Dustin Li, Esin Durmus, Ethan Perez, et al. Studying large language model generalization with influence functions. *arXiv preprint arXiv:2308.03296*, 2023.

Xiaochuang Han and Yulia Tsvetkov. Influence tuning: Demoting spurious correlations via instance attribution and instance-driven updates. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pp. 4398–4409, 2021.

Kazuaki Hanawa, Sho Yokoi, Satoshi Hara, and Kentaro Inui. Evaluation of similarity-based explanations. In *International Conference on Learning Representations*, 2020.

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

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, Yanning Chen, et al. Liger-kernel: Efficient triton kernels for llm training. In *Championing Open-source DEvelopment in ML Workshop@ ICML25*.

William B Johnson, Joram Lindenstrauss, et al. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

Krishnateja Killamsetty, Sivasubramanian Durga, Ganesh Ramakrishnan, Abir De, and Rishabh Iyer. Grad-match: Gradient matching based data subset selection for efficient deep model training. In *International Conference on Machine Learning*, pp. 5464–5474. PMLR, 2021.

Diederik Kinga, Jimmy Ba Adam, et al. A method for stochastic optimization. In *International conference on learning representations (ICLR)*, 2015.

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

Andreas Köpf, Yannic Kilcher, Dimitri Von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens, Abdullah Barhoum, Duc Nguyen, Oliver Stanley, Richárd Nagyfi, et al. Openassistant conversations-democratizing large language model alignment. *Advances in neural information processing systems*, 36:47669–47681, 2023.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hannaneh Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. 2024.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. 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. 7602–7635, 2024.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, et al. The flan collection: Designing data and methods for effective instruction tuning. In *International Conference on Machine Learning*, pp. 22631–22648. PMLR, 2023.

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

James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International conference on machine learning*, pp. 2408–2417. PMLR, 2015.

Baharan Mirzasoleiman, Jeff Bilmes, and Jure Leskovec. Coresets for data-efficient training of machine learning models. In *International Conference on Machine Learning*, pp. 6950–6960. PMLR, 2020.

Garima Pruthi, Frederick Liu, Satyen Kale, and Mukund Sundararajan. Estimating training data influence by tracing gradient descent. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 19920–19930. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/e6385d39ec9394f2f3a354d9d2b88eec-Paper.pdf.

Yulei Qin, Yuncheng Yang, Pengcheng Guo, Gang Li, Hang Shao, Yuchen Shi, Zihan Xu, Yun Gu, Ke Li, and Xing Sun. Unleashing the power of data tsunami: A comprehensive survey on data assessment and selection for instruction tuning of language models. *Transactions on Machine Learning Research*, 2024.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

Lloyd S Shapley et al. A value for n-person games. 1953.

Martin Slawski and Matthias Hein. Non-negative least squares for high-dimensional linear models: Consistency and sparse recovery without regularization. *Electronic Journal of Statistics*, 7:3004–3056, 2013.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*, 2022.

Jiachen T Wang, Prateek Mittal, Dawn Song, and Ruoxi Jia. Data shapley in one training run. In *The Thirteenth International Conference on Learning Representations*, 2025a.

Jiachen T. Wang, Dawn Song, James Zou, Prateek Mittal, and Ruoxi Jia. Capturing the temporal dependence of training data influence. In *The Thirteenth International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=uHLgDEgiS5.

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, 2024.

Shaobo Wang, Xuan Ouyang, Tianyi Xu, Yuzheng Hu, Jialin Liu, Guo Chen, Tianyu Zhang, Junhao Zheng, Kexin Yang, Xingzhang Ren, et al. Opus: Towards efficient and principled data selection in large language model pre-training in every iteration. *arXiv preprint arXiv:2602.05400*, 2026.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: selecting influential data for targeted instruction tuning. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 54104–54132, 2024.

Sang Michael Xie, Shibani Santurkar, Tengyu Ma, and Percy S Liang. Data selection for language models via importance resampling. *Advances in Neural Information Processing Systems*, 36: 34201–34227, 2023.

An Yang et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Jipeng Zhang, Yaxuan Qin, Renjie Pi, Weizhong Zhang, Rui Pan, and Tong Zhang. Tagcos: Task-agnostic gradient clustered coreset selection for instruction tuning data. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 4671–4686, 2025.

Ling Zhang, Xianliang Yang, Juwon Yu, Park Cheonyoung, Lei Song, and Jiang Bian. Holdout-loss-based data selection for llm finetuning via in-context learning. In *International Conference on Learning Representations*, volume 2026, pp. 15751–15775, 2026.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

Wanru Zhao, Yihong Chen, Yuzhi Tang, Wentao Ma, Shengchao Hu, Xu Hu, Alex Iacob, Abhinav Mehrotra, and Nic Lane. Rethinking data curation in llm training: Online reweighting offers better generalization than offline methods. In *International Conference on Learning Representations*, volume 2026, pp. 112899–112923, 2026.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models, 2023. URL https://arxiv.org/abs/2311.07911.

A Extended Related Work

Influence and gradient-based selection. Influence functions estimate how removing or upweighting a training example changes a target loss (Koh & Liang, 2017; Han & Tsvetkov, 2021), and TracIn (Pruthi et al., 2020) replaces the inverse-Hessian term with an inner product of gradients accumulated along the training trajectory. Grosse et al. (2023) scale the estimator to LLMs with Kronecker-factored curvature. LESS (Xia et al., 2024) adapts the idea to instruction tuning by storing projected LoRA gradients from a handful of warmup checkpoints and scoring candidates against an Adam-preconditioned validation gradient. All of these score examples *independently*, which is what Section 3.1 argues is inadequate once the selected examples are used together in one update. Cheaper proxies avoid gradients altogether, ranking data by representation similarity to the target (Zhang et al., 2018; Hanawa et al., 2020; Xie et al., 2023), by learned difficulty (Li et al., 2024), or by outlier structure (Chhabra et al., 2024); they are far cheaper but carry no guarantee about the update the model will actually take.

Coreset construction and gradient matching. A second line selects a *subset* whose aggregate gradient approximates a reference. CRAIG (Mirzasoleiman et al., 2020) casts this as submodular cover, and GRAD-MATCH (Killamsetty et al., 2021) solves it with Orthogonal Matching Pursuit, refitting the coefficients after each greedy addition. TAGCOS (Zhang et al., 2025) applies greedy coreset selection to clustered LLM gradients for task-agnostic coverage, and DoGE (Fan et al., 2024) matches at the domain rather than the example level. DSDM (Engstrom et al., 2024) selects pretraining data with datamodels. These methods are offline: the reference gradient and the candidate scores are computed once against a fixed model. Our formulation keeps the matching objective but makes it step-conditioned, and Section 3.2 argues—and Section 4.2 confirms—that the coupled OMP solver these methods rely on is precisely the part that fails on LLM gradients, since the greedy support-recovery conditions of Cai & Wang (2011) and Ding et al. (2012) require a separation between dictionary atoms that stochastic, projected LoRA gradients do not provide. The non-negativity constraint we impose is the standard remedy for the signed cancellation that unconstrained least squares exhibits on co-linear dictionaries (Slawski & Hein, 2013).

Online selection. GREATS (Wang et al., 2024) is the closest online method: it scores a candidate pool at the current parameters and adds a greedy intra-batch correction, but assigns unit weights to whatever it selects. Deng et al. (2024) pursue gradient trajectories across steps, and Aljundi et al. (2019) select by gradient diversity in the continual setting. The concurrent work of Wang et al. (2026) also targets optimizer-aware online selection, but in pretraining and within the individual-scoring paradigm, so it does not address the interactions among jointly selected samples that subset-level reweighting creates. ICA (Zhang et al., 2026) and ADAPT (Zhao et al., 2026) relax different assumptions than we do—offline in-context scoring and selection-free soft reweighting respectively—and we compare against both in Appendix E.10.

Fine-tuning differs from pretraining. The two settings ask different questions of a selector. Pretraining selection seeks broad coverage of a general distribution under a token budget, so diversity and deduplication dominate and per-example weights buy little. Fine-tuning has an explicit target distribution and a small validation set that represents it, which makes *matching* well posed and makes weighting worthwhile: with a few thousand steps and a few examples per step, how much each selected example contributes is a decision the selector can afford to make. This is why we reweight and why the target enters through an optimizer preconditioner rather than through a corpus-level mixture.

Per-example gradients without per-example backward passes. The rank-1 structure of a linear layer's gradient underlies the ghost dot-product used for per-example gradient norms in differentially private training (Bu et al., 2023) and for influence estimation (Wang et al., 2025a;b). Our representation shares that factorization and differs in contraction order (Section 3.3). Random projection of the factors follows the Johnson–Lindenstrauss construction (Johnson et al., 1984) as used by Xia et al. (2024). The Kronecker-factored curvature models we evaluate as an alternative target-side geometry come from Martens & Grosse (2015); Botev et al. (2017); Eschenhagen et al. (2023).

B Algorithm Details

B.1 Orthogonal Matching Pursuit (OMP)

This appendix details the standard Orthogonal Matching Pursuit (OMP) sub-routine and its relation to the simpler greedy filter used in our main Algorithm 2.

OMP and the precomputed objective. OMP is a greedy algorithm that iteratively selects samples to minimize the residual error between the current weighted combination and the target vector. To make iterative selection tractable, we expand the squared L_2 objective in Eq. 2:

$$\mathcal{L}(\mathbf{w}, \mathcal{S}) = \|\nabla\tilde{\ell}_{\text{val}}\|_2^2 - 2\sum_{i\in\mathcal{S}} w_i\, b_i + \sum_{i\in\mathcal{S}}\sum_{j\in\mathcal{S}} w_i w_j\, G_{ij} + \lambda\|\mathbf{w}\|_2^2, \tag{9}$$

where b_i and G_{ij} are the alignment-vector and Gram-matrix entries defined in Section 3.2. Both $\mathbf{b} \in \mathbb{R}^{|\mathcal{S}^{(t)}|}$ and $\mathbf{G} \in \mathbb{R}^{|\mathcal{S}^{(t)}|\times|\mathcal{S}^{(t)}|}$ are computed once per step on the projected low-rank gradients (Section 3.3); preconditioning is absorbed entirely into $\mathbf{b}$ via the target-side rescaling of Section 3.2.

The OMP sub-routine. Algorithm 1 details the selection process. At each iteration, we search for a candidate u that, when added to the current set $\mathcal{S}'$, yields the minimal objective value. For a fixed trial set $\mathcal{S}_{\text{trial}} = \mathcal{S}' \cup \{u\}$, the optimal weights have a closed-form ridge solution:

$$\mathbf{w}^*_{\text{trial}} = (\mathbf{G}_{\mathcal{S}_{\text{trial}}} + \lambda\mathbf{I})^{-1}\mathbf{b}_{\mathcal{S}_{\text{trial}}}. \tag{10}$$

Since the batch size k is small in the online setting, solving this system is negligible relative to gradient computation.

Why our main algorithm uses a simplified variant. In the main Filter-then-Weight procedure (Algorithm 2, Stage 1), we adopt a simplified version of OMP that skips the inner ridge solve when each new sample is added, retaining only the residual update $r \leftarrow r - \nabla\ell_{i^*}$. Solving for optimal weights at every greedy step, as standard OMP does, attempts to construct a linear combination that perfectly fits the noisy validation gradient—often selecting samples that cancel out random fluctuations in the validation batch rather than capturing the general descent direction. Our two-stage decoupling instead separates directional agreement (Stage 1, unit weights) from precise weight assignment (Stage 2, NNLS over the fixed set), reducing the noise propagated through greedy selection.

B.2 Pseudo-codes of Filter-then-Weight

Algorithm 2 summarizes one iteration. Stage 0 reads the pool in α micro-batches and forms the two statistics once. Stage 1 selects a support of size k greedily under unit weights, tracking the residual through the penalty $p_i = \sum_{j\in\mathcal{S}'} G_{ij}$, since $b_i - p_i = \langle\nabla\ell_i, r\rangle$. Stage 2 re-forms the statistics on that support alone, a problem a factor α^2 smaller, and solves for non-negative coefficients under the relative ridge of Eq. 6. Stage 3 hands the composite gradient to the optimizer. Section 3.2 motivates the split.

C Proofs

Role. This section collects the derivations behind Sections 2 and 3. Two of them—the reordering identity of Appendix C.5 and the isotropic connection of Appendix C.3—are used by the shipped algorithm. The rest establish that the objective is the right one to solve, or document alternatives we evaluated and did not adopt; each subsection says which.

C.1 Shapley value Using First-order Taylor Expansion

Role. under a linear $\mathcal{P}_t$, the Shapley value of a sample's first-order utility is exactly its weighted gradient inner product with the target. This is why additive scoring is the correct rule in that regime, and therefore what Section 3.1 has to leave behind. The algorithm does not compute Shapley values.

We First discuss the first-order Taylor expansion as an approximation to the utility change after selecting and reweighting the training samples, such that $U(S^{(t)}; x_j) = l(\tilde{\theta}_t(S^{(t)}), x_j) - l(\theta_{t-1}, x_j) \approx -\eta_t\nabla l_j(\tilde{\theta}_t(S^{(t)}) - \theta_{t-1})$.

Importantly, our two-stage algorithm **separates the weighting stage from the Shapley value's subset selection**. This means a sample's weight is fixed and does not change depending on which subset it appears in. Because weights are assigned only after the batch is selected, they are independent of the probability of inclusion. The weights therefore only affect the scale of the gradient update, not the sampling process itself.

Using the definition for optimizer and weighting in Section 2.2, we derive

$$U(S^{(t)}; x_j) = -\eta_t \nabla l_j \mathcal{P}_t(g(w)) = -\eta_t \nabla l_j \mathcal{P}_t(\sum_{x_i \in S^{(t)}} w_i \nabla l_i). \tag{11}$$

Recall the Shapley value definition (Shapley et al., 1953):

$$\phi_i(U^{(t)}(x_i; x_j)) = \sum_{s=1}^{|S^{(t)}|} \begin{bmatrix} |S^{(t)}| - 1 \\ s-1 \end{bmatrix}^{-1} \sum_{S \subseteq S^{(t)} \setminus x_i, |S|=s-1} [U^{(t)}(S \cup x_i; x_j) - U^{(t)}(S; x_j)]. \tag{12}$$

Substituting Equation 11 into the definition, and consider linearized $\mathcal{P}_t$, we get

$$\begin{aligned} \phi_i(U^{(t)}(x_i; x_j)) &= \nabla l_j \sum_{s=1}^{|S^{(t)}|} \begin{bmatrix} |S^{(t)}| - 1 \\ s-1 \end{bmatrix}^{-1} \sum_{S \subseteq S^{(t)} \setminus x_i, |S|=s-1} [\mathcal{P}_t(\sum_{x_i \in S \cup x_i} w_i \nabla l_i) - \mathcal{P}_t(\sum_{x_i \in S} w_i \nabla l_i)] \\ &= -\eta_t \nabla l_j \mathcal{P}_t(w_i \nabla l_i). \end{aligned} \tag{13}$$

For 0-1 data selection problem, the utility of selecting sample x_j, $\phi_i(U^{(t)}(x_i; x_j)) = \nabla l_j \mathcal{P}_t(\nabla l_i)$; for weighting problem, $\phi_i(U^{(t)}(x_i; x_j)) = -\eta_t w_i \nabla l_j \mathcal{P}_t(\nabla l_i)$.

C.2 Shapley value Using Second-order Taylor Expansion

Role. once curvature enters, the same value acquires a term coupling the sample to the rest of the batch, so batch utility is no longer a sum over its members. This is the formal statement of the second failure mode in Section 3.1. Not used at run time.

Next, we discuss the second-order Taylor expansion at θ_{t-1} as an approximation, $U(S^{(t)}; x_j) = l(\tilde{\theta}_t(S^{(t)}), x_j) - l(\theta_{t-1}, x_j) \approx \nabla l(\theta_{t-1}, x_j)(\tilde{\theta}_t(S^{(t)}) - \theta_{t-1}) + \frac{1}{2}(\tilde{\theta}_t(S^{(t)}) - \theta_{t-1})^T H_j^{(t)} (\tilde{\theta}_t(S^{(t)}) - \theta_{t-1})$, where the Hessian matrix $H_j^{(t)} := \nabla^2 l_j$. Notably, the utility of the total set is not a linear sum of all individual samples, due to the presence of second-order Hessian matrix. In the rest of this paper, we substitute $H_j^{(t)}$ for H_j for simplicity.

Similarly, updating with Equation 11, we draw connection between $\tilde{\theta}_t(S^{(t)})$ and θ_{t-1}, i.e., $\tilde{\theta}_t(S^{(t)}) - \theta_{t-1} = -\eta_t \mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i)$. Substituting into the original utility function, for training sample $x_i \in S^{(t)}$ and validation sample $x_j \in D_{val}$, we get candidate set utility

$$\begin{aligned} U(S^{(t)}; x_j) = -\eta_t \nabla l_j \mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i) + \frac{\eta_t^2}{2} \mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i) H_j \mathcal{P}_t(\sum_{z_k \in S^{(t)}} w_k \nabla l_k), \\ \text{where } H_j = \nabla^2 l_j. \end{aligned} \tag{14}$$

We take the last item as an approximation of candidate set utility, denoted as $\hat{U}^{(t)}(S^{(t)}; x_j)$. This expression can be derived from only the step size η_t and the individual gradients of current model θ_{t-1} to involved training and validation samples, i.e., ∇l_i and ∇l_j.

Recall that $S^{(t)}$ is the final training data batch selected at step t. For a sample x_i, denote $S \subseteq S^{(t)} \setminus x_i$ as any combinations without x_i. Similarly, we consider only linearized $\mathcal{P}_t$. By Equation 14, the marginal contribution of adding a new sample into the current candidate batch S can be derived as

$$
\begin{aligned}
&U^{(t)}(S \cup x_i; x_j) - U^{(t)}(S; x_j) \\
=& -\eta_t \Big(\nabla l_j \mathcal{P}_t(\sum_{z_i \in S \cup x_i} w_i \nabla l_i) - \nabla l_j \mathcal{P}_t(\sum_{z_i \in S} w_i \nabla l_i) \Big) \\
&+ \frac{\eta_t^2}{2} \Big(\mathcal{P}_t(\sum_{z_i \in S \cup x_i} w_i \nabla l_i) H_j \mathcal{P}_t(\sum_{z_k \in S \cup x_i} w_k \nabla l_k) - \mathcal{P}_t(\sum_{z_i \in S} w_i \nabla l_i) H_j \mathcal{P}_t(\sum_{z_k \in S} w_k \nabla l_k) \Big) \\
=& -\eta_t \mathcal{P}_t(w_i \nabla l_i) \nabla l_j + \frac{\eta_t^2}{2} \Big(2 \mathcal{P}_t(w_i \nabla l_i) H_j \mathcal{P}_t(\sum_{x_k \in S} w_k \nabla l_k) + \mathcal{P}_t(w_i \nabla l_i) H_j \mathcal{P}_t(w_i \nabla l_i) \Big).
\end{aligned} \tag{15}
$$

For the formula of Shapley value in Equation 12, only the second item in Equation 15 is variant cross different S. Substituting it into Equation 12, we get $\eta_t^2 \mathcal{P}_t(w_i \nabla l_i) H_j^{(t)} \mathcal{P}_t(\sum_{x_k \in S} w_k \nabla l_k)$. Based on the linearity of $\mathcal{P}_t$ and the symmetry property of Shapley values, the expectation of the sum term over all subsets S is half the sum of the remaining samples:

$$
\mathbb{E}_S \left[\sum_{x_k \in S} \nabla w_k l_k \right] = \frac{1}{2} \sum_{x_k \in S^{(t)} \setminus x_i} w_k \nabla l_k.
$$

Substituting this back into Equation 12 and combining it with the invariant terms from Equation 15:

$$
\begin{aligned}
\phi_i(U^{(t)}(x_i; x_j)) &= -\eta_t \mathcal{P}_t(w_i \nabla l_i) \nabla l_j + \frac{\eta_t^2}{2} \mathcal{P}_t(w_i \nabla l_i) H_j \mathcal{P}_t(w_i \nabla l_i) \\
&\quad + \eta_t^2 \mathcal{P}_t(w_i \nabla l_i) H_j \mathcal{P}_t \left(\frac{1}{2} \sum_{x_k \in S^{(t)} \setminus x_i} w_k \nabla l_k \right) \\
&= -\eta_t \mathcal{P}_t(w_i \nabla l_i) \nabla l_j + \frac{\eta_t^2}{2} \mathcal{P}_t(w_i \nabla l_i) H_j \mathcal{P}_t \left(w_i \nabla l_i + \sum_{x_k \in S^{(t)} \setminus x_i} w_k \nabla l_k \right).
\end{aligned} \tag{16}
$$

Since $\nabla w_i l_i + \sum_{x_k \in S^{(t)} \setminus x_i} w_k \nabla l_k$ equals the sum over the entire batch $S^{(t)}$, we obtain the final expression:

$$
\phi_i(U^{(t)}(x_i; x_j)) = -\eta_t \mathcal{P}_t(w_i \nabla l_i) \nabla l_j + \frac{\eta_t^2}{2} \mathcal{P}_t(w_i \nabla l_i) H_j \mathcal{P}_t \left(\sum_{x_k \in S^{(t)}} w_k \nabla l_k \right). \tag{17}
$$

C.3 Connection between Second-Order Contribution and Gradient Matching

Role. under an isotropic Hessian the second-order utility reduces to the gradient-matching objective of Eq. 2, up to a weight-norm penalty. This is the derivation Section 3.1 cites, and the Gram matrix the algorithm builds is its quadratic term.

Notably, even using approximation $H_j^{(t)} := \nabla^2 l_j$ as a parameter size matrix, the cost can be high for online computation. Therefore, previous work Wang et al. (2024) has simplified it further to an identity matrix I. Considering a validation batch S_{val}, utility of training sample combination $S^{(t)}$ using the second-order Taylor Expansion can be simplified as

$$
U(S^{(t)}; x_j) = -\eta_t \left(\mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i) \sum_{x_j \in S_{val}} \nabla l_j - \frac{\eta_t}{2} \mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i)^2 \right) \tag{18}
$$

Then, minimizing the total utility is equivalent to:

$$\min_w U(S^{(t)}; x_j) \equiv \min_w -\mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i) \sum_{x_j \in S_{val}} \nabla l_j + \frac{\eta_t}{2} \mathcal{P}_t(\sum_{z_i \in S^{(t)}} w_i \nabla l_i)^2 \tag{19}$$

Meanwhile, expanding the gradient matching goal in Equation 2, we are optimizing over

$$\begin{aligned} &\min_w \left\| \sum_{x_j \in S_{val}} \nabla l_j - \mathcal{P}_t(\sum_{x_i \in S^{(t)}} w_i \nabla l_i) \right\|_2^2 + \lambda \|w\|_2^2 \\ \equiv &\min_w -\mathcal{P}_t(\sum_{x_i \in S^{(t)}} w_i \nabla l_i) \sum_{x_j \in S_{val}} \nabla l_j + \frac{1}{2} \mathcal{P}_t(\sum_{x_i \in S^{(t)}} w_i \nabla l_i)^2 + \frac{1}{2} \lambda \|w\|_2^2. \end{aligned} \tag{20}$$

Comparing Equation 19 and 20, both objectives share the same first-order alignment term $-\mathcal{P}_t(\sum_i w_i \nabla l_i) \sum_{x_j \in S_{val}} \nabla l_j$, which captures the training–validation interaction. The two objectives differ in three ways:

- **Quadratic term coefficient.** The second-order Taylor expansion weights $\mathcal{P}_t(\sum_i w_i \nabla l_i)^2$ by $\eta_t/2$, while gradient matching weights it by $1/2$. Since $\eta_t \sim 10^{-4}$ in fine-tuning, gradient matching amplifies this quadratic penalty by roughly four orders of magnitude. Rather than a flaw, this amplification acts as an *implicit regularizer* on the update norm, discouraging the solver from constructing large composite gradients that may overfit a noisy validation direction.
- **Pairwise interactions encoded in the quadratic term.** The quadratic term expands as $\mathcal{P}_t(\sum_i w_i \nabla l_i)^2 = \sum_{i,j} w_i w_j \langle \mathcal{P}_t(\nabla l_i), \mathcal{P}_t(\nabla l_j) \rangle$, introducing pairwise interactions across selected samples. Highly correlated samples incur larger cross terms $w_i w_j \langle \cdot, \cdot \rangle$, naturally encoding diminishing returns and redundancy control. This is the structural property that motivates the subset-level matching formulation in Section 3.1: pairwise structure is precisely what individual scoring discards.
- **Ridge term.** The $\frac{\lambda}{2}\|w\|_2^2$ term in gradient matching has no analog in the second-order expansion. It serves a different role from the quadratic term: rather than controlling the geometry of $\mathcal{P}_t(\sum_i w_i \nabla l_i)$, it controls the mixture coefficients w themselves, preventing degenerate solutions where a few samples receive arbitrarily large weights.

Caveat: the isotropic-Hessian assumption. The connection above relies on $H_j \approx I$, a strong simplification at LLM scale where Hessian eigenspectra are highly anisotropic. A more accurate approximation such as K-FAC (Martens & Grosse, 2015) would replace $\mathcal{P}_t(\sum_i w_i \nabla l_i)^2$ with the Mahalanobis form $\mathcal{P}_t(\sum_i w_i \nabla l_i)^\top H_j \mathcal{P}_t(\sum_i w_i \nabla l_i)$, yielding pairwise terms $\langle \mathcal{P}_t(\nabla l_i), H_j \mathcal{P}_t(\nabla l_j) \rangle$ in the Gram matrix rather than the raw inner product. We derive the corresponding K-FAC-based scoring procedure and an efficient factorized computation in Appendix C.6. Measured against the shipped isotropic score, however, neither K-FAC nor a diagonal Fisher proxy changes the selected support much, and both cost more and score slightly worse downstream (Table 5), so we retain the isotropic form.

C.4 Inner-product and cosine maximization as squared matching

Role. maximizing alignment and minimizing squared distance share optimizers when the update norm is fixed or regularized. This licenses the change of objective from Eq. 1 to Eq. 2; it is a statement about the formulation, not a step the algorithm takes.

The gradient matching design is a natural extension from the optimization goal defined in Section 2.2. Formally, Theorem C.1 establishes that maximizing inner-product or cosine alignment is equivalent to minimizing a squared ℓ_2 distance up to terms that depend only on the norm of the update. In particular, when the optimizer-induced mapping $\mathcal{P}_t$ is linear (e.g., SGD) and the update norm is fixed or regularized, the two formulations admit the same set of optimal solutions. Therefore, under these conditions, distance-based matching preserves the optimal solutions of the original alignment objective.

When $\mathcal{P}_t$ is nonlinear and optimizer-dependent (e.g., Adam/AdamW), strict equivalence between the two objectives no longer holds. Nevertheless, minimizing the distance in the optimizer-induced feature space can be viewed as matching the effective update directions seen by the optimizer.

Theorem C.1 (Inner-product and cosine maximization as squared ℓ_2 matching)**.** *Let $v \in \mathbb{R}^d$ be fixed, and let $h : \mathcal{W} \to \mathbb{R}^d$ be any mapping from a feasible set $\mathcal{W}$ to $\mathbb{R}^d$.*

(i) Inner-product alignment. *For any $w \in \mathcal{W}$,*

$$\arg\max_{w\in\mathcal{W}} \langle v, h(w)\rangle \;=\; \arg\min_{w\in\mathcal{W}} \Big(\|v - h(w)\|_2^2 - \|h(w)\|_2^2\Big). \tag{21}$$

(ii) Cosine alignment. *Assume $\|v\|_2 \neq 0$ and $\|h(w)\|_2 \neq 0$ for all $w \in \mathcal{W}$. Define the cosine similarity objective $C(w) := \frac{\langle v, h(w)\rangle}{\|v\|_2 \, \|h(w)\|_2}$, and the normalized directions $\hat{v} := \frac{v}{\|v\|_2}$, $\hat{h}(w) := \frac{h(w)}{\|h(w)\|_2}$. Then*

$$\arg\max_{w\in\mathcal{W}} C(w) \;=\; \arg\min_{w\in\mathcal{W}} \|\hat{v} - \hat{h}(w)\|_2^2. \tag{22}$$

Proof. We prove the two statements separately.

(i) Inner-product case. By the polarization identity, for any $w \in \mathcal{W}$,

$$\|v - h(w)\|_2^2 = \|v\|_2^2 + \|h(w)\|_2^2 - 2\langle v, h(w)\rangle. \tag{23}$$

Rearranging (23) yields

$$\langle v, h(w)\rangle = \frac{1}{2}\|v\|_2^2 + \frac{1}{2}\big(\|h(w)\|_2^2 - \|v - h(w)\|_2^2\big).$$

Since $\|v\|_2^2$ is constant with respect to w, maximizing $\langle v, h(w)\rangle$ over $\mathcal{W}$ is equivalent (in the sense of sharing the same set of optimizers) to minimizing

$$\|v - h(w)\|_2^2 - \|h(w)\|_2^2,$$

which proves (21).

(ii) Cosine case. We do not assume $\|v\|_2 = 1$ nor $\|h(w)\|_2 = 1$. Instead, we *define* $\hat{v} := v/\|v\|_2$ and $\hat{h}(w) := h(w)/\|h(w)\|_2$, which are well-defined under the stated assumptions and satisfy $\|\hat{v}\|_2 = \|\hat{h}(w)\|_2 = 1$.

Expanding the squared distance gives

$$\begin{aligned}
\|\hat{v} - \hat{h}(w)\|_2^2 &= \|\hat{v}\|_2^2 + \|\hat{h}(w)\|_2^2 - 2\langle \hat{v}, \hat{h}(w)\rangle \\
&= 2 - 2\left\langle \frac{v}{\|v\|_2}, \frac{h(w)}{\|h(w)\|_2} \right\rangle \\
&= 2 - 2\,\frac{\langle v, h(w)\rangle}{\|v\|_2 \, \|h(w)\|_2} \\
&= 2 - 2C(w).
\end{aligned} \tag{24}$$

The mapping $x \mapsto 2 - 2x$ is strictly decreasing. Hence minimizing $\|\hat{v} - \hat{h}(w)\|_2^2$ over w is equivalent to maximizing $C(w)$ over w, which establishes (22). □

C.5 Equivalence of dot product calculation order

Role. the Kronecker identity that lets the token axis be contracted before the feature axes. This one the implementation depends on: it is what makes Eq. 8 legitimate and the validation target aggregable.

An important property of Kronecker product is that, for matrices A, B, C, D with matching dimensions,

$$< A \otimes B, C \otimes D >_F = < A, C >_F \otimes < B, D >_F \; . \tag{25}$$

For dot product (Frobenius inner product) between A and B, $< A, B >_F$, it also can be rewritten in trace of matrix:

$$< A, B >_F =< B, A >_F = tr(A^\top B) = tr(BA^\top). \tag{26}$$

The dot product between vector a and b can also be denoted through transposition: $a \cdot b = a^T b$. These properties serve as an important foundation of proofs in this paper.

Non-sequential data. We start by proving the equivalence for non-sequential data:

$$\sum_{x_j \in S_{val}^{(t)}} \nabla l_j \cdot \nabla l_i = (\sum_{x_j \in S_{val}^{(t)}} g_j^\top g_i)(a_j^\top a_i) = a_i^\top (\sum_{x_j \in S_{val}^{(t)}} a_j g_j^\top) g_i. \tag{27}$$

Here, the derivative of loss to i-th pre-activation as $g_i = \frac{\partial l}{\partial s_i} \in \mathbb{R}^{d_2 \times 1}$, and the i-th input vector as $a_i \in \mathbb{R}^{d_1 \times 1}$. The first equation is derived by:

$$\nabla l_j \cdot \nabla l_i = (g_j \otimes a_j) \cdot (g_i \otimes a_i) = (g_j \cdot g_i)(a_j \cdot a_i) = (g_j^\top g_i)(a_j^\top a_i),$$

where the first equation is by definition and the second and third one applies the property of Kronecker product in Equation 25 and dot product, respectively.

Looking into the last item, we have

$$\begin{aligned} a_i^\top (\sum_{x_j \in S_{val}^{(t)}} a_j g_j^\top) g_i &= \sum_{x_j \in S_{val}^{(t)}} a_i^\top (a_j g_j^\top) g_i \\ &= \sum_{x_j \in S_{val}^{(t)}} (a_i^\top a_j)(g_j^\top g_i) \\ &= \sum_{x_j \in S_{val}^{(t)}} (g_j^\top g_i)(a_j^\top a_i). \end{aligned}$$

The first two equations rearranges the combination of vectors. The last item is swapping scalar $(g_j^\top g_i)$ and $(a_j^\top a_i)$. Proved for non-sequential data.

Sequential data.

$$< \sum_{x_j \in S_{val}^{(t)}} \nabla l_j, \nabla l_i >_F =< \sum_{x_j \in S_{val}^{(t)}} g_j^\top g_i, a_j^\top a_i >_F =< g_i^\top, a_i^\top (\sum_{x_j \in S_{val}^{(t)}} a_j g_j^\top) >_F .$$

In computation for sequential data, the 3-dimensional tensor (B, T, d) is generally reshaped as (BT, d) for better position-agnostic parallelization. We only need to substitute $a_i \in \mathbb{R}^{d_1 \times 1}$ to $a_i \in \mathbb{R}^{d_1 \times T}$ and $g_i = \frac{\partial l}{\partial s_i} \in \mathbb{R}^{d_2 \times 1}$ to $g_i = \frac{\partial l}{\partial s_i} \in \mathbb{R}^{d_2 \times T}$ in the above proving process. Notably, the equivalences remain, as all computation is inside the dot product. By denoting the trace matrix as $tr(\cdot)$, the second equation can be converted into

$$tr((a_j^\top a_i)^\top (\sum_{x_j \in S_{val}^{(t)}} g_j^\top g_i)) = tr(a_i^\top (\sum_{x_j \in S_{val}^{(t)}} a_j g_j^\top) g_i),$$

which is essentially the same as Equation 27. Further, using the property of dot product, $< g_i^\top, a_i^\top (\sum_{x_j \in S_{val}^{(t)}} a_j g_j^\top) >_F =< a_i, (\sum_{x_j \in S_{val}^{(t)}} a_j g_j^\top) g_i >_F$. Using this equivalence, we can flexibly select the more efficient way for calculation according to dimensionality.

C.6 Equivalence of vector-Hessian-vector calculation order

Role. the same reordering for a K-FAC target-side curvature model. We implemented and measured it (Table 5); it changes the selected support little, costs more, and scores slightly worse, so the shipped algorithm uses the isotropic form. Included to document the design space.

Non-sequential data. We first prove the corrected derivation. The goal is to compute the influence of a set of candidate samples on a target sample i, mediated by the Hessian estimated from a validation set.

$$\text{Score} = \nabla l_i^\top \Big(\sum_{x_j \in S_{val}^{(t)}} H_j \Big) \Big(\sum_{x_k \in S_{cand}^{(c,t)}} \nabla l_k \Big).$$

Assume the block-diagonal property for Hessian matrix in a layer level, and the K-FAC approximation where $H_{j,\ell} \approx G_\ell \otimes A_\ell$ represents the aggregate Hessian statistics derived from the validation batch. Using the property of the Kronecker product $(A \otimes B)(C \otimes D) = (AC) \otimes (BD)$, we expand the term for each layer ℓ:

$$\begin{aligned}
&\nabla l_i^\top H_{val} \sum_{x_k \in S_{cand}^{(c,t)}} \nabla l_k \\
=&\sum_{\ell=1}^{L} (g_{i,\ell}^\top \otimes a_{i,\ell}^\top)(G_\ell \otimes A_\ell) \sum_{x_k \in S_{cand}^{(c,t)}} (g_{k,\ell} \otimes a_{k,\ell}) \\
=&\sum_{\ell=1}^{L} \sum_{x_k \in S_{cand}^{(c,t)}} \left[(g_{i,\ell}^\top \otimes a_{i,\ell}^\top)(G_\ell \otimes A_\ell)(g_{k,\ell} \otimes a_{k,\ell}) \right] \\
=&\sum_{\ell=1}^{L} \sum_{x_k \in S_{cand}^{(c,t)}} \left[(g_{i,\ell}^\top G_\ell g_{k,\ell}) \otimes (a_{i,\ell}^\top A_\ell a_{k,\ell}) \right].
\end{aligned}$$

Since $g_{i,\ell}^\top G_\ell g_{k,\ell}$ and $a_{i,\ell}^\top A_\ell a_{k,\ell}$ are both scalars, the Kronecker product becomes a simple scalar multiplication. We now expand $G_\ell = \sum_{m=1}^{|V|} g_{m,\ell} g_{m,\ell}^\top$:

$$\begin{aligned}
g_{i,\ell}^\top G_\ell g_{k,\ell} &= g_{i,\ell}^\top \Big(\sum_{m=1}^{|V|} g_{m,\ell} g_{m,\ell}^\top \Big) g_{k,\ell} \\
&= \sum_{m=1}^{|V|} (g_{i,\ell}^\top g_{m,\ell})(g_{m,\ell}^\top g_{k,\ell}) = \sum_{m=1}^{|V|} (g_{i,\ell} \cdot g_{m,\ell})(g_{m,\ell} \cdot g_{k,\ell}).
\end{aligned}$$

Similarly, for the activation term:

$$a_{i,\ell}^\top A_\ell a_{k,\ell} = \sum_{m=1}^{|V|} (a_{i,\ell} \cdot a_{m,\ell})(a_{m,\ell} \cdot a_{k,\ell}).$$

Substituting these back, we obtain the final corrected equation. Note that the summation over candidates x_k must remain outside the product terms:

$$\text{Score} = \sum_{\ell=1}^{L} \sum_{x_k \in S_{cand}^{(c,t)}} \left[\left(\sum_{m \in S_{val}^{(t)}} (g_{i,\ell} \cdot g_{m,\ell})(g_{m,\ell} \cdot g_{k,\ell}) \right) \cdot \left(\sum_{m \in S_{val}^{(t)}} (a_{i,\ell} \cdot a_{m,\ell})(a_{m,\ell} \cdot a_{k,\ell}) \right) \right].$$

Sequential data. The derivation can be naturally extended to sequential data where activations and gradients are matrices (e.g., $a \in \mathbb{R}^{d_1 \times T}$). We replace the vector dot product with the Frobenius inner product $\langle X, Y \rangle_F = \text{Tr}(X^\top Y)$. The logic of K-FAC remains consistent: it aggregates statistics across the temporal dimension, effectively treating time steps as additional batch samples. Thus, the scalar terms simply change their operator:

$$\text{Score} = \sum_{\ell=1}^{L} \sum_{x_k \in S_{cand}^{(c,t)}} \left[\left(\sum_{m \in S_{val}^{(t)}} \langle g_{i,\ell}, g_{m,\ell} \rangle_F \cdot \langle g_{m,\ell}, g_{k,\ell} \rangle_F \right) \cdot \left(\sum_{m \in S_{val}^{(t)}} \langle a_{i,\ell}, a_{m,\ell} \rangle_F \cdot \langle a_{m,\ell}, a_{k,\ell} \rangle_F \right) \right].$$

Empirical note: how much does richer curvature buy? We implemented the full K-FAC-based scoring above as a drop-in replacement for the raw Gram matrix in Section 3.2, together with a cheaper diagonal empirical-Fisher-style proxy, and measured both against the shipped isotropic score on the same Llama-3.2-1B/TyDiQA setup as Section 4.4. The relevant question is not whether the isotropic score reproduces richer curvature models exactly, but whether those models change the *selection* enough to justify their cost.

Table 5: Target-side curvature models compared against the shipped isotropic score. Spearman is computed over full candidate-score rankings, greedy overlap is the fraction of the residual-filtered support shared with the isotropic version, and cost is the additional GPU time to construct the curvature-adjusted target. Diagnostics are averaged over the same real selection steps.

Target-side curvature	Spearman	Greedy overlap	Added cost (ms/step)	Peak F1	Δ F1
Isotropic (ours)	1.000	100%	0	**48.86**	—
Diagonal empirical-Fisher proxy	0.988	91.7%	76	47.86	−1.00
K-FAC (Martens & Grosse, 2015)	0.964	79.8%	186	47.36	−1.50

Table 5 shows that both anisotropic models agree closely with the isotropic score—Spearman 0.96–0.99 and 80%–92% overlap on the final selected support—while costing 76–186 ms per step and yielding *lower* downstream F1. A possible explanation is that LoRA's low-rank structure already absorbs much of the curvature variance relevant for selection, so the extra Kronecker factors mostly add estimation noise. We do not read this as evidence that curvature is irrelevant, only that in this framework the isotropic score is an accurate-enough and much cheaper operating point. The method is also effective across markedly different candidate geometries: the 3B candidate pool has effective rank 16.53 against 8.38 for 8B, yet the TyDiQA block of Table 2 shows consistent gains at both. We include the full derivation above for completeness and to document the design space explored.

D Complexity Comparison

In sequential data, consider pre-activation derivative $g_i, g_j \in \mathbb{R}^{T \times d_2}$ and input $a_i, a_j \in \mathbb{R}^{T \times d_1}$. For simplicity of notation, we ignore all layer indicators, ℓ, when not required. Random projection is budgeted by the *product* $d_1 d_2$, held constant across layers, with each factor at least the LoRA rank; the projected implementations below are evaluated on the projected rank-1 factors, so d_1 and d_2 denote post-projection dimensions throughout.

D.1 Complexity Analysis of First-Order Matching Statistics

We analyze the cost of the two statistics Filter-then-Weight consumes: the candidate–target alignment vector $\mathbf{b}$ and the candidate–candidate Gram matrix $\mathbf{G}$. Both are gradient inner products, and this appendix derives the rows of Table 1: first how the gradients are represented at all, then, on that representation, the two ways of evaluating the inner products that differ in which axis they contract first.

Throughout, $N = |\mathcal{S}^{(t)}|$ is the candidate-pool size, $V = |\mathcal{S}^{(t)}_{\text{val}}|$ the number of validation examples aggregated into the target, $k = |\mathcal{S}'|$ the number of candidates retained by Stage 1, with $k \ll N$ and $N = \alpha k$, and L the number of adapted linear layers. For one such layer, write $a_i \in \mathbb{R}^{T \times D_1}$ for the input activations and $g_i \in \mathbb{R}^{T \times D_2}$ for the pre-activation gradients, so that the per-example parameter gradient is $\nabla \ell_i = a_i^\top g_i \in \mathbb{R}^{D_1 \times D_2}$, and d_1, d_2 for the dimensions after random projection. Costs are derived per layer and summed over L.

Two features of the LoRA setting fix these symbols and are worth stating before the derivation, because they decide which terms bind. First, the gradients in question are gradients of the adapters,

not of the backbone. LoRA injects two linear layers per adapted module, so for Llama-3.2-1B with seven target modules over sixteen blocks $L = 224$, and every one of them has $\min(D_1, D_2) = r = 8$: the adapters are (D_{in}, r) and (r, D_{out}). Summed over layers, $\sum_\ell D_1 D_2 = 5.6 \times 10^6$, the LoRA parameter count. Second, the projection is budgeted by the product. Each layer is given a target $d_1 d_2$ equal to the projection dimension, split between the two sides with each at least r, which for this model yields $(d_1, d_2) \in \{(32, 8), (8, 32)\}$ and a per-layer budget $d_1 d_2 = 256$ at $d = 256$. Summed over layers, $\sum_\ell d_1 d_2 = 5.7 \times 10^4$ and $\sum_\ell (d_1 + d_2) = 9.0 \times 10^3$. The projected gradient is therefore two orders of magnitude smaller than the adapter gradient, and $d_1 d_2$ cannot grow without the projection budget growing with it.

D.1.1 Representation

Read one example at a time, per-example gradients cost one backward pass each, $\mathcal{O}\left((N + V)C_{\text{bwd}}\right)$, and occupy $\mathcal{O}(LND_1D_2)$ once the pool is held for the Gram matrix.

The rank-1 factorization removes the backward count. Forward and backward hooks capture a_i and g_i for every example in a *single* backward pass over the concatenated micro-batch, so the pool is read in α backward passes rather than $N + V$ of them. This is the lever that matters most in wallclock, because a backward pass over the micro-batch is the single most expensive thing in a selection step, two orders of magnitude above everything below.

Projection then removes the store. It cannot be applied to the gradient, which would have to be formed first; the factors can be projected directly, $a_i P_a$ and $g_i P_g$, at $\mathcal{O}\left(LT(N + V)(D_1 d_1 + D_2 d_2)\right)$ and without ever forming a $D_1 \times D_2$ object. What is kept afterwards is $\mathcal{O}\left(LT(N + V)(d_1 + d_2)\right)$ of factors and $\mathcal{O}(LNd_1d_2)$ of projected gradients, against $\mathcal{O}(LND_1D_2)$ unprojected: 7.0 MB rather than 688 MB at the setting below. That reduction is what lets the whole pool stay resident beside the model, which $\mathbf{G}$ requires, and its own cost sits two orders of magnitude below the backward pass it accompanies.

Everything below is evaluated on this representation. The two implementations part on one choice: which axis to contract first.

D.1.2 Candidate–Target Alignment Vector

The alignment vector is $b_i = \langle \nabla \ell_i, \nabla \ell_{\text{val}} \rangle_F$ with $\nabla \ell_{\text{val}} = \sum_{x_j \in \mathcal{S}_{\text{val}}^{(t)}} \nabla \ell_j$.

Ghost: contract the feature axes first. Ghost computation rewrites the same quantity as $\langle a_j^\top g_j, a_i^\top g_i \rangle_F = \langle a_j a_i^\top, g_j g_i^\top \rangle_F$, contracting d_1 and d_2 before T and leaving two $T \times T$ matrices. Each candidate–validation pair costs $\mathcal{O}(T^2 d_1)$ and $\mathcal{O}(T^2 d_2)$ to build the two matrices and $\mathcal{O}(T^2)$ to contract them.

Crucially, the validation sum cannot be pulled out: both factors of $\langle a_j a_i^\top, g_j g_i^\top \rangle_F$ depend on j, so the product must be evaluated for every one of the NV pairs, giving $\mathcal{O}\left(LNVT^2(d_1 + d_2)\right)$ in time and $\mathcal{O}(NVT^2)$ in space for a vectorized implementation.

Ours: contract the token axis first. Contracting T first leaves a $d_1 \times d_2$ object per example and frees the target to be aggregated once, $\nabla \ell_{\text{val}} = \sum_j a_j^\top g_j$ at $\mathcal{O}(LVTd_1d_2)$ and $\mathcal{O}(Ld_1d_2)$ of storage, before any candidate is read. Each candidate then contributes $\mathcal{O}(Td_1d_2)$, for $\mathcal{O}\left(LT(N + V)d_1d_2\right)$ overall and $\mathcal{O}(LNd_1d_2)$ of space. The context length enters linearly rather than quadratically, and the pair count is $N + V$ rather than NV.

The contraction may be reassociated as $b_i = \langle a_i^\top g_i, \nabla \ell_{\text{val}} \rangle_F = \langle g_i, a_i \nabla \ell_{\text{val}} \rangle_F$, or symmetrically against a_i, which replaces the $d_1 \times d_2$ intermediate by a $T \times \min(d_1, d_2)$ one. The two orders cost the same in time, so this is purely a statement about the per-candidate working set: the reassociated form grows linearly in the projection width where the direct form grows quadratically, but carries a factor T, so the direct form is the smaller of the two exactly when $T > \max(d_1, d_2)$. That is what makes the token-first route reachable when a parameter-sized intermediate cannot be held. After projection the budget $d_1 d_2$ is fixed by construction and both are far below anything that binds, so the implementation uses the direct order. The aggregation of the validation batch is the separate and larger effect, and it is the one ghost computation cannot recover at any ordering.

D.1.3 CANDIDATE–CANDIDATE GRAM MATRIX

The Gram matrix $G_{ij} = \langle \nabla \ell_i, \nabla \ell_j \rangle_F$ is intrinsically pairwise across examples, and unlike the alignment vector it has no shared aggregate to reorder against: both operands carry both factors, so the reassociation above has no analogue here.

Ghost. Per pair, $G_{ij} = \langle a_i a_j^\top, g_i g_j^\top \rangle_F$ costs $\mathcal{O}(T^2(d_1 + d_2))$, and there are N^2 pairs, giving $\mathcal{O}\left(LN^2T^2(d_1 + d_2)\right)$ in time and $\mathcal{O}(N^2T^2)$ in space. The pool size now enters quadratically alongside the context length.

Ours. Stack the flattened projected gradients into $\Phi \in \mathbb{R}^{N \times Ld_1d_2}$ and take $\mathbf{G} = \Phi\Phi^\top$: $\mathcal{O}\left(Ld_1d_2[NT + N^2]\right)$ in time and $\mathcal{O}(LNd_1d_2 + N^2)$ in space. Stage 2 re-forms the Gram on the selected support alone at $\mathcal{O}\left(Lk^2d_1d_2\right)$, a factor α^2 below Stage 1, so decoupling confines the precise part of the problem to a batch-sized subset.

D.1.4 WHERE THE MEMORY GOES

The asymptotics above are score-side only, and read alone they invite the wrong conclusion, because a selection step does not run on an empty device. It runs inside a forward and backward pass whose own working set sets the scale everything else is measured against. Table 6 gives that ledger for the configuration of our experiments: Llama-3.2-1B with L=224 adapted linear layers, T=1024, k=8, α=4 so N=32, four validation examples per micro-batch so V=16 aggregated over the step, and fp32 score-side intermediates.

Table 6: Memory ledger for one selection step, Llama-3.2-1B at T=1024, N=32, V=16, micro-batch B=12, fp32 score-side. Model-side terms are what the backward pass already holds. Both implementations read the same projected factors, so that store is charged to neither; the rows below it are what each adds on top. The ghost coefficient 3 is the calibration of Appendix E.13.

	Term	**Size**
Model-side	Frozen backbone weights, bf16	2.3 GB
	Logits buffer and its fp32 cross-entropy, $BT\lvert\mathcal{V}\rvert$	$\sim$15 GB
Shared	Projected rank-1 factors, $LT(N{+}V)(d_1{+}d_2)$	1.64 GB
	(unprojected gradients, LND_1D_2, for reference)	688 MB
Ours adds	Projected gradients feeding $\mathbf{b}$ and $\mathbf{G}$, LNd_1d_2	7.0 MB
	Aggregated target $\nabla\ell_{\text{val}}$, Ld_1d_2	224 KB
Ghost adds	Token-pair tensor at T=1024, $3NVT^2$	6.0 GB
	Token-pair tensor at T=4096	96 GB
Per-layer temp.	Direct order, d_1d_2	1.0 KB
	Reassociated order, $T\min(d_1, d_2)$	32 KB

Both implementations are given the same representation. Ghost computation composes with the projection exactly as ours does, so its factors are d_1 and d_2 throughout, and the 1.64 GB factor store is common to both rather than a cost we charge it. What the projection does not shrink is the token-pair tensor: $a_j a_i^\top$ is indexed by pairs of token positions, not by features, so it is $T \times T$ for any d_1. Projection lowers the cost of *building* it, from T^2D_1 to T^2d_1, and leaves its size untouched. That is a property of the contraction order, not a handicap we impose.

Reading the ledger by implementation, ours adds 7.2 MB to the shared store and ghost adds 6.0 GB, for 1.65 GB against 7.6 GB at T=1024. The shared term grows linearly in T and the token-pair term quadratically, so at T=4096 the same accounting gives 6.6 GB against 103 GB. Against a model-side floor of roughly 17 GB, that is the difference between a step that fits and one that does not, and it is why the empirical frontier in Appendix E.13 separates so sharply. The per-layer temporaries at the bottom, the quantity the reassociation of Eq. 8 trades between, differ by 32$\times$ in absolute terms seven orders of magnitude below the floor, which is why the choice of order is settled by convenience rather than by cost.

D.1.5 SUMMARY

Three levers separate the implementation we ship from the alternatives, and they act on different resources.

The rank-1 factorization acts on time. It reads the pool in α backward passes rather than $N+V$, and a backward pass over the micro-batch dominates the step: at the setting above the model's forward and backward is $\sim 2.4 \times 10^{14}$ FLOPs against 1.1×10^{12} for the entire selection computation, projection included. Nothing else in this appendix is visible in wallclock beside it.

Projection acts on residency. Every $D_1 D_2$ becomes $d_1 d_2$ and the candidate store falls from 688 MB to 7.0 MB, a factor of 98. This is what allows the pool to be held in full on a device that is already carrying the model, which the Gram matrix requires, and the projection's own cost is one of the terms two orders of magnitude below the backward pass.

Contracting the token axis first acts on context length, and this is where we part from ghost computation. Two consequences compound: the $T \times T$ token-pair matrices never exist, worth $T(d_1{+}d_2)/(d_1 d_2) = 160\times$ per layer at our setting, and the validation batch can be aggregated before any candidate is touched, worth a further $NV/(N{+}V) \approx 11\times$, for 1.7×10^3 on the alignment vector's leading terms; counting projection on both sides the whole selection computation is $14\times$ cheaper. The memory gap is the sharper one and it grows with T, which Appendix E.13 measures directly. The aggregation is the part ghost cannot recover by reordering: both factors of $\langle a_j a_i^\top, g_j g_i^\top \rangle_F$ carry the validation index.

E ADDITIONAL EXPERIMENT DETAILS

E.1 SETTING

All experiments were implemented using the PyTorch framework and the Hugging Face Transformers and PEFT libraries, running on NVIDIA A100 (40 GB), H100 (80 GB), H200 (141 GB), and Tesla V100 (32 GB)GPUs. We use Llama-3.2-1B (Grattafiori et al., 2024) as our primary backbone, fine-tuned via Low-Rank Adaptation (LoRA). Unless otherwise stated, the LoRA rank is $r = 8$, LoRA scaling factor 32, and dropout rate 0, applied to all linear layers (query, key, value, output, gate, up, and down projections). We optimize with Adam (β_1=0.9, β_2=0.999, ϵ=10^{-8}); the AdamW comparison in Section 4.3 uses the same moments with decoupled weight decay 0.01. The learning rate warms up linearly over the first 600 steps to 1×10^{-4} and then decays over 3,000 steps to a floor of 1×10^{-5}; runs are capped at 5,000 steps. Sequences are truncated at 1,024 tokens and training uses BF16 mixed precision via Hugging Face Accelerate.

Selection uses a projection dimension of d=256, a candidate pool of N=αk (N=32 for every Llama backbone, from 1B to 8B, and 24 for Qwen3-0.6B), greedy residual filtering, and an NNLS solve with relative ridge 10^{-4} scaled by the mean Gram diagonal. Checkpoints are written every 200 steps; a lightweight validation-loss probe over 1,000 held-out samples runs every 50 steps for monitoring only and does not feed selection. Every configuration is run with three seeds (10/20/30), and all reported means and standard deviations are taken over those seeds.

The per-layer projection dimensions are derived from the requested budget d by $d_1 d_2 \approx d$ with $d_1/d_2 \approx \sqrt{D_2/D_1}$ and both sides floored at the LoRA rank r. For the adapters of Llama-3.2-1B this yields $(d_1, d_2) \in \{(32, 8), (8, 32)\}$ at d=256 and $\{(16, 8), (8, 16)\}$ at d=128 and $\{(8, 8)\}$ at d=64, which is the smallest budget the rank floor admits. The d=256 row of the projection sweep in Table 4 is the main-table configuration; the pool-size study of Appendix E.9 is a separate set of runs and its absolute values should not be read against the main tables.

Numerical details of the standardization. Standard deviations are floored before division, at 10^{-2} times the median standard deviation over the coordinates that have one, rather than by a fixed additive constant. Coordinate scales here span several orders of magnitude, so an absolute floor lets a coordinate with near-zero variance be divided up by an arbitrary factor and dominate the metric. Exactly constant coordinates are harmless either way, since the centering already sends them to zero; the ones that matter are small but nonzero. For the first two steps of a run roughly half the coordinates are exactly constant, because LoRA initializes B to zero and the gradient with respect to A vanishes until it moves, which is why the reference scale is taken over the live coordinates only.

One switch in the released code is off for every run reported here. The contraction that forms the projected gradients can be batched across candidates, which is roughly an order of magnitude faster on that term but agrees with the per-candidate path only to float32 rounding (Appendix E.8); the reported runs use the per-candidate path.

Budget accounting. The training pool holds $270{,}679$ examples and checkpoints are written every 200 steps, so one checkpoint is $200k$ selected examples. The budget is a fraction of the pool rather than a step count: every configuration stops at the checkpoint nearest 5%, which is step $1{,}600$ at $k{=}8$ ($12{,}800$ examples, 4.73%) and step $2{,}200$ at $k{=}6$ ($13{,}200$, 4.88%). The step count therefore varies with k while the data consumed does not, which is what makes the comparison across backbones budget-matched. Because the learning-rate schedule is indexed by optimization step and is independent of the data budget, truncating a run at step T yields exactly the run one would obtain by setting the budget to $T \cdot k$ training samples. Every budget-matched comparison in this paper is therefore a comparison at a common amount of data, and within a single backbone at a common step count; the data-ratio curve in Figure 1 is a family of fixed-budget results read off one run per method.

E.2 Datasets and Models

We use the following publicly released datasets and pre-trained models, all consistent with their licenses for non-commercial research purposes:

Datasets.

- **Open-Instruct** (Lambert et al., 2024) — ODC-BY 1.0.
- **FLAN v2** (Longpre et al., 2023) — Apache 2.0.
- **Chain-of-Thought** (Wei et al., 2022) — Apache 2.0.
- **Dolly** (Conover et al., 2023) — CC-BY-SA 3.0.
- **OASST1** (Köpf et al., 2023) — Apache 2.0.
- **MMLU** (Hendrycks et al., 2021) — MIT License.
- **TyDiQA** (Clark et al., 2020) — Apache 2.0.
- **BBH** (Suzgun et al., 2022) — MIT License.
- **IFEval** (Zhou et al., 2023) — Apache 2.0.
- **Tulu-3 personas-IF** (Ge et al., 2025) — ODC-BY 1.0.

Models.

- **Llama-3.2-1B** and **Llama-3.2-3B** (Grattafiori et al., 2024) — Meta Llama 3.2 Community License Agreement.
- **Llama-3.1-8B** (Grattafiori et al., 2024) — Meta Llama 3.1 Community License Agreement.
- **Qwen3-0.6B** (Yang et al., 2025) — Apache 2.0.

E.3 Ablation Variant Definitions

We list the full set of variants studied in Section 4.4. Variants 1–4 together with Ours are the cells of a factorial design over the filter, the weighting stage, and the preconditioning of the alignment vector:

Variant	Filter	Weighting	Preconditioned $\mathbf{b}$
3. Vanilla Filter Only	top-k	unit	no
2. Optimizer-Aware Filter Only	top-k	unit	yes
4. Vanilla Filter+Reweight	greedy	NNLS	no
1. Hard Filter+Reweight	top-k	NNLS	yes
Ours	greedy	NNLS	yes

Adjacent pairs therefore isolate one factor each, which is how Section 4.4 reads them. Variants 2 and 3 coincide by construction with the LESS and TracIn baselines of Table 2—top-k selection on a preconditioned and on a raw alignment score respectively—and we report those same runs

in both tables rather than repeating them, which is why the entries agree to the printed precision. Figure 2 shows the preliminary single-seed trajectories originally used to prune the design space; the multi-seed results that supersede it, including the two divergent variants, are reported in Table 3.

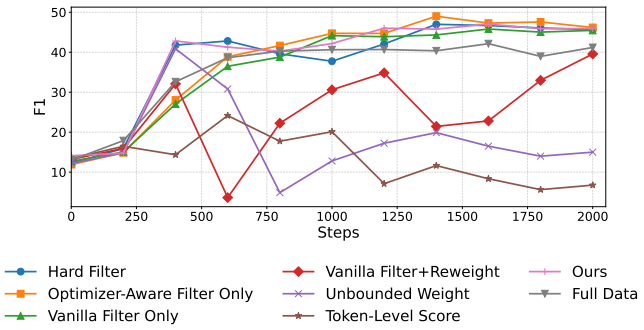


Figure 2: Preliminary single-seed ablation used to prune the design space. *Unbounded Weights* and *Token-Level Score* diverge; Table 3 confirms both collapses across three seeds.

1. **Hard Filter + Reweight.** Replaces the greedy filtering (Stage 1) with a top-k selection based on optimizer-aware scores, while retaining the NNLS weighting stage (Stage 2).
2. **Optimizer-Aware Filter Only.** Selects top candidates using optimizer-aware scores without the subsequent weighting stage. This is the LESS baseline of Table 2.
3. **Vanilla Filter Only.** Selects candidates based on raw gradient dot products, without optimizer-aware preconditioning or the weighting stage. This is the TracIn baseline of Table 2.
4. **Vanilla Filter + Reweight.** Implements the two-stage Filter-then-Weight framework using raw gradients for both filtering and weighting, removing all optimizer-aware preconditioning.
5. **Unbounded Weights.** Relaxes the non-negative constraint in Eq. 2, allowing weights to take negative values.
6. **Token-Level Score.** Restricts gradient similarity computation to identical token positions, ignoring sequence-level context.
7. **Ours.** The full Filter-then-Weight algorithm proposed in this paper.

E.4 Fixed-Budget Performance

Table 2 reads each run at its best checkpoint. Reading instead at the final checkpoint of the budget fixes the resources every method consumes and removes any dependence on checkpoint selection. Table 7 reports that view. The strongest method in each row is unchanged; the runner-up changes on Llama-3.2-1B/MMLU and Qwen3-0.6B/TyDiQA, where LESS overtakes GRAD-MATCH and GREATS respectively. Ours ranks above every selection baseline in all four rows, at $p = 0.063$ on the sign test used in Appendix E.5; per row the Welch p-values are 0.12, 0.37, 0.22 and 0.08, so this read is weaker per cell than the best-of-run one but points the same way with no checkpoint chosen.

Table 7: **Fixed-budget performance.** All methods evaluated at the final checkpoint of the 5% data budget; formatting as in Table 2.

		References		*Selection only*			*Coupled*	*Ours*
Dataset	Backbone	Origin	Full	TracIn	LESS	GREATS	GRAD-MATCH	Filter-then-Weight
MMLU (Acc)	Llama-3.2-1B	28.13	28.67 $\pm$0.63	29.04 $\pm$0.04	29.67 $\pm$1.31	29.59 $\pm$0.40	28.77 $\pm$0.56	**30.93** $\pm$**0.03**
	Qwen3-0.6B	45.86	**46.59** $\pm$**0.10**	46.16 $\pm$0.07	45.98 $\pm$0.37	46.06 $\pm$0.42	44.98 $\pm$0.72	46.23 $\pm$0.67
TyDiQA (F1)	Llama-3.2-1B	12.69	38.68 $\pm$1.36	47.01 $\pm$1.43	46.88 $\pm$0.91	47.16 $\pm$1.08	44.15 $\pm$0.85	**48.67** $\pm$**2.68**
	Qwen3-0.6B	20.94	41.14 $\pm$0.64	43.07 $\pm$0.97	43.57 $\pm$0.28	42.70 $\pm$0.92	43.14 $\pm$1.13	**44.82** $\pm$**1.00**

E.5 Significance of the Main Comparison

Every cell of Table 2 is three seeds, which is enough to resolve a difference of roughly one across-seed deviation and not enough to resolve anything smaller. We therefore give both readings: the per-cell test, which is weak, and the across-cell test, which is not.

Table 8 compares ours against the strongest *selection* baseline in each cell with a one-sided Welch t-test on the three seeds—Welch rather than Student, since the deviations differ by up to a factor of four between methods in the same cell. Three of the seven cells reach $p < 0.07$ and none reaches $p < 0.01$; Qwen3-0.6B/MMLU and Llama-3.2-3B/TyDiQA do not separate on their own at all. Every quantity below is computed from the means and deviations printed in Table 2, so the test is reproducible from the table without access to the runs.

Table 8: Ours against the strongest selection baseline in each cell of Table 2. One-sided Welch t-test, three seeds per arm. The full-data reference is excluded, since it is not a selection method; it outranks ours only in the Qwen3-0.6B/MMLU cell.

Cell	Strongest baseline	Δ	t	p
MMLU / Llama-3.2-1B (Adam)	GRAD-MATCH	$+0.79$	3.33	0.039
MMLU / Qwen3-0.6B (Adam)	LESS	$+0.07$	0.25	0.409
TyDiQA / Llama-3.2-1B (Adam)	GREATS	$+1.36$	1.34	0.146
TyDiQA / Llama-3.2-1B (AdamW)	TracIn	$+1.10$	1.43	0.119
TyDiQA / Llama-3.2-3B (Adam)	TracIn	$+0.69$	1.15	0.160
TyDiQA / Llama-3.1-8B (Adam)	GREATS	$+1.23$	2.43	0.061
TyDiQA / Qwen3-0.6B (Adam)	GREATS	$+1.04$	2.39	0.042
Sign test over the seven cells, $7/7$				0.008
Fisher combination, $\chi^2_{14} = 32.0$				0.004

The seven cells are not independent replicates of one hypothesis, but they are different backbones, targets and optimizers, and ours ranks first in all of them. Under the null that each cell is decided by a coin flip, a sign test gives $p = 2^{-7} = 0.008$, and Fisher's combination of the seven one-sided p-values gives $\chi^2_{14} = 32.0$, $p = 0.004$. We read the main comparison at this level rather than at the level of individual cells, and the ablation of Section 4.4—where the two collapse modes are separated from ours by 20–35 F1—carries correspondingly more weight per cell than the baseline comparison does.

E.6 Additional Targets and Training Pools

Table 9 extends the comparison to BBH (Suzgun et al., 2022) and IFEval (Zhou et al., 2023), and varies the training pool between the general Open-Instruct corpus and the in-domain Tulu-3 personas-IF pool (Ge et al., 2025). Because the latter is far smaller than Open-Instruct, we use a 25% budget there so that step counts remain comparable. At matched budget on Tulu-3-IF, ours is ahead of Full outright (31.42 vs. 30.50 at step 400; 34.49 vs. 32.83 at step 800), so the remaining gap in Table 9 is a budget effect rather than a ceiling.

E.7 What Makes Greedy Differ from Top-k

Greedy residual selection could help merely by discarding large-norm, high-variance gradients. Table 10 separates the two causes by intervening on the same pool at every step: unit-normalizing candidates (removing magnitude, keeping redundancy) leaves the greedy/top-k divergence largely intact, whereas a norm- and score-preserving decorrelation (removing only redundancy) eliminates it entirely. On the steps where the rules disagree, greedy picks the less redundant subset (mean pairwise cosine 0.696 vs. 0.730, lower on 17 of 22 steps). The gain over top-k is a signature of diversity modeling, not of a variance clip.

E.8 In-iteration Running Time Analysis

We report measured wall-clock cost on a single A100-80 GB GPU. All numbers use the GPU-side greedy filter and the optimized NNLS solver; a variant of our method that scores candidates with the

Table 9: **Additional targets and training pools. Bold** marks the best *trained* method, underline the second best; Origin is the untrained Llama 3.2-1B backbone, listed for reference only. Cells without a standard deviation are single-seed.

		References		*Selection only*			*Coupled*	*Ours*
Training pool	Target	Origin	Full	TracIn	LESS	GREATS	GRAD-MATCH	Filter-then-Weight
Open-Instruct	BBH	32.78	33.43 $\pm$0.21	34.17 $\pm$0.09	**34.58** $\pm$**0.57**	34.17 $\pm$0.46	33.39 $\pm$0.44	<u>34.44</u> $\pm$0.28
Open-Instruct	IFEval	31.82	17.90	26.79	27.38	27.57	<u>27.81</u>	**29.73**
Tulu-3-IF	IFEval	31.82	**37.53** $\pm$**0.57**	34.86 $\pm$1.47	35.56 $\pm$0.82	34.50 $\pm$0.76	33.84 $\pm$1.39	<u>36.81</u> $\pm$0.54

Table 10: **What makes greedy differ from top-k.** Two interventions applied to the same candidate pool at every selection step, in isolation and together. The last column counts steps at which the two rules select different subsets.

Condition	Candidate norm	Redundancy	Greedy $\neq$ top-k
A Original	original	original	22 / 58
B Unit-normalized	all $= 1$ (variance removed)	preserved	15 / 58
C Decorrelated	preserved	removed	0 / 58
D Both	all $= 1$	removed	3 / 58

ghost dot-product (Wang et al., 2025a) instead of the factorized representation of Section 3.3 runs out of memory at projection dimension 256 and is therefore omitted.

Table 11 compares per-step cost against data-matched full training, i.e. standard SFT consuming the same number of samples over four steps. Every selection method, including ours, is cheaper than this reference. The saving does not come from touching less data—all methods perform roughly four forward/backward passes over the same samples—but from three sources: fewer optimizer updates (one per four candidate batches); a lighter candidate backward pass that reads gradients and activations through hooks without retaining the computational graph (Section 3.3); and a selection step whose cost is small relative to the step. Ours is nonetheless the most expensive of the selection methods, $0.06\times$ of the reference above the cheapest at 3B and $0.08\times$ at 8B, and it is worth saying exactly where that goes.

It is not the Gram matrix. $\mathbf{G}$ is a single $N \times N$ product on already-formed vectors and takes 0.1 ms. The cost is the contraction that forms those vectors: one small matmul per candidate per adapted layer, NL of them per step, 14,336 at 8B. Each is 524 K FLOPs and takes $69\,\mu$s, the same $69\,\mu$s whether it is a candidate or the target, which is the signature of launch overhead rather than arithmetic; the measured throughput is 0.04% of the device peak. Summed, this is the *Statistics* column of Table 12, 0.50 s at 1B and 1.00 s at 8B, growing linearly in L and accounting for roughly a third of our gap to the cheapest baseline alongside the filter and the solve.

Because the term is launch overhead, it is an artifact of how the loop is written rather than a cost of the method. Contracting all candidates in one einsum per layer issues L launches instead of NL and cuts it by about $11\times$: $500 \rightarrow 48$ ms at 1B, $895 \rightarrow 84$ ms at 3B, $995 \rightarrow 95$ ms at 8B. End to end on Llama-3.2-1B/TyDiQA the median step falls from 8.62 s to 8.22 s, the 0.40 s the component measurement predicts. We nonetheless report the per-candidate figures throughout, and every result in this paper was produced with it: the two paths agree only to float32 rounding, and over 25 steps that was enough to change one of the eight selected candidates on two of them. Adopting the faster path is a decision to re-measure, not a free speed-up, and we leave it as an option in the released code.

A phase-level breakdown confirms that selection is not the bottleneck. Table 12 profiles the greedy filter, the NNLS solve, and the construction of the statistics they read, against candidate-gradient computation and the optimizer update. The filter and the solve are 3.5%–4.0% of a training step and forming $\mathbf{b}$ and $\mathbf{G}$ a further 3.9%–4.3%, so selection is 8.2% of a step at 1B and 7.4% at 8B. The share *decreases* with model size, since gradient computation and the optimizer update grow faster than the selection problem, whose dimension is set by the pool size and the projection dimension rather than by the backbone.

Table 11: **Running time comparison** on a single A100-80 GB GPU (seconds per step). *full*-$\times 4$ denotes standard SFT consuming the same number of samples over four steps; relative columns are normalized to it.

		8B		3B	
Method	Pool	s / step	rel.	s / step	rel.
full-$\times 4$ (SFT)	α=1	34.8	1.00×	29.2	1.00×
TracIn (Pruthi et al., 2020)	α=4	22.6	0.65×	19.4	0.66×
LESS (Xia et al., 2024)	α=4	23.4	0.67×	20.1	0.69×
GRAD-MATCH (Killamsetty et al., 2021)	α=4	23.6	0.68×	20.3	0.69×
GREATS (Wang et al., 2024)	α=4	24.1	0.69×	19.8	0.68×
Ours	α=4	25.4	0.73×	20.9	0.72×

Table 12: **Where a selection step spends its time** (ms, single A100-80 GB, N=32, T=1024). *Statistics* is the cost of building $\mathbf{b}$ and $\mathbf{G}$, measured separately at the same shapes; the second figure contracts all candidates in one einsum per layer rather than one per candidate per layer. Every result in this paper uses the first, so it is the one the remaining columns and the total are consistent with. The columns do not sum to the total, which also carries the forward pass, data loading and the periodic loss probe.

			Statistics					
Backbone	L	Grad. compute	per cand.	batched	Filter	NNLS	Optimizer	Total
1B	224	7,476	500	48	346	126	1,081	11,799
3B	392	14,160	895	84	590	220	2,054	20,900
8B	448	16,518	995	95	634	250	3,298	25,395

E.9 Candidate Pool Size

The experiments in this subsection vary the oversampling factor α while holding the selected batch size fixed at the small-backbone setting of Section 4, k=6 and k_{val}=2, so the data budget is unchanged and only the number of candidates the selector inspects per step grows. The target is TyDiQA. These are a separate set of runs from Table 2 and their absolute values are not comparable with it; what the tables below report is the trend across α within this set.

Cost grows approximately linearly in the pool size (Table 13): quadrupling the candidate count raises per-step time by 3.65×, below the 4× that a naive accounting would predict, because the selection stage itself remains a small fraction of the step. Accuracy is stable across the range (Table 14): our method leads GREATS at every pool size, and neither method improves monotonically with more candidates, indicating that the benefit of a larger pool saturates once it contains enough complementary directions to fill the batch.

Table 13: Cost scales near-linearly with candidate-pool size.

Pool	N	s / step	Norm.
α=4	24	11.8	1.00×
α=8	48	22.4	1.90×
α=16	96	43.1	3.65×

Table 14: TyDiQA best-of-run F1 (3 seeds).

Pool	GREATS	Ours
α=4 (N=24)	43.16 $^{\pm 1.61}$	**48.94** $^{\pm 1.83}$
α=8 (N=48)	46.85 $^{\pm 0.95}$	**47.63** $^{\pm 1.20}$
α=16 (N=96)	44.58 $^{\pm 0.50}$	**48.90** $^{\pm 0.65}$

E.10 Comparison with Offline Scoring and Full-Pool Reweighting

Two recent methods share our goal of improving fine-tuning data efficiency but rest on assumptions that make a head-to-head comparison only partially meaningful. ICA (Zhang et al., 2026) scores candidates offline against a fixed holdout set via in-context learning; the scores are computed once and do not track the model or optimizer state as training proceeds. ADAPT (Zhao et al., 2026)

performs no selection at all, instead soft-reweighting the entire candidate pool, so it cannot exploit redundancy when the pool contains near-duplicate gradients. Neither targets the step-wise, optimizer-conditioned problem studied here. We implement both following their original descriptions and evaluate them under our protocol in Table 15; we report them here rather than in the main comparison precisely because they are not drop-in baselines for this setting. Under our protocol both trail every selection-only baseline, which is the behavior their assumptions predict: offline scores go stale as the model moves, and reweighting without selection spends the batch on redundant directions.

Table 15: TyDiQA best-of-run F1 (3 seeds) including two recent methods evaluated under our online protocol.

	References		*Selection only*			*Coupled*	*Other paradigms*		*Ours*
Backbone	Origin	Full	TracIn	LESS	GREATS	GRAD-MATCH	ADAPT	ICA	FW
Llama-3.2-1B	12.69	42.47 $^{\pm 0.30}$	47.01 $^{\pm 1.43}$	47.21 $^{\pm 2.58}$	<u>47.50</u> $^{\pm 0.56}$	45.35 $^{\pm 1.35}$	42.98 $^{\pm 1.01}$	46.22 $^{\pm 1.99}$	**48.86** $^{\pm 1.67}$
Qwen3-0.6B	20.94	43.65 $^{\pm 0.61}$	45.17 $^{\pm 0.97}$	45.01 $^{\pm 0.95}$	<u>45.25</u> $^{\pm 0.43}$	44.23 $^{\pm 1.33}$	42.86 $^{\pm 1.14}$	44.90 $^{\pm 0.67}$	**46.29** $^{\pm 0.62}$

E.11 Quality of the Frozen-Preconditioner Approximation

Section 3.2 freezes the Adam second moment at the previous-step state. We test this approximation directly on real checkpoints by recomputing candidate scores with a fresh, candidate-specific second moment and comparing against the shipped frozen version.

Table 16: Frozen versus candidate-specific second moment, each row a checkpoint taken from a real run. Overlap is the fraction of the selected support shared by the two variants.

Setting	Spearman	Selected-sample overlap	Relative ℓ_2 error
1B / TyDiQA, mid-run	0.951	81%	0.84
3B / TyDiQA, late-run	0.978	91%	0.71

Table 16 reports the comparison. The raw score magnitudes differ substantially, yet the candidate ordering and the resulting support are largely preserved: Spearman correlation is 0.95–0.98 and selection overlap 81%–91%. The approximation is therefore *selection-stable* rather than pointwise accurate, which is what Eq. 2 actually requires, and is consistent with the slowly varying preconditioner argument in Section 3.2.

E.12 When Does Selection Have Headroom?

Section 4.2 observes that selection helps substantially on some backbone/target pairs and barely at all on others (Qwen3-0.6B/MMLU), and Section 4.3 finds the same flat regime on BBH. We give here a simple step-conditioned quantity that makes the distinction precise, and state plainly that we use it as an interpretive lens rather than as a validated predictor.

Under the notation of Section 3.1, let $\nabla \ell_{\text{val}}$ be the target gradient and $\nabla \ell_i$ that of candidate i. Following the first-order approximation behind Eq. 2, write the individual alignment score as $s_i = \langle \nabla \ell_{\text{val}}, \nabla \ell_i \rangle$. For a pool of size N, the equal-weight pool update has first-order utility $\bar{s} = \frac{1}{N} \sum_{i=1}^{N} s_i$, while the best equal-weight subset of size B attains $\bar{s}_{\text{top-}B} = \frac{1}{B} \sum_{i \in \text{Top}B(s)} s_i$. Define the one-step *selection headroom* as $H_B = \bar{s}_{\text{top-}B} - \bar{s}$.

Proposition E.1. *Let $\sigma_s^2 = \frac{1}{N} \sum_{i=1}^{N} (s_i - \bar{s})^2$ be the dispersion of the alignment scores. Then $H_B \le \sigma_s \sqrt{(N-B)/B}$.*

Proof. Let $d_i = s_i - \bar{s}$, so $\sum_i d_i = 0$ and $\sum_i d_i^2 = N\sigma_s^2$. Write $S = \sum_{i \in \text{Top}B} d_i$, hence $\sum_{i \notin \text{Top}B} d_i = -S$. Cauchy–Schwarz on each group gives $S^2 \le B \sum_{i \in \text{Top}B} d_i^2$ and $S^2 \le (N-B) \sum_{i \notin \text{Top}B} d_i^2$. Dividing and adding, $S^2 \left(\frac{1}{B} + \frac{1}{N-B} \right) \le \sum_i d_i^2 = N\sigma_s^2$, so $S^2 \le \sigma_s^2 B(N-B)$ and $H_B = S/B \le \sigma_s \sqrt{(N-B)/B}$. □

The bound says that headroom is controlled by score dispersion: no selector can beat the equal-weight pool update by much when the candidates are equally aligned with the target, however sophisticated its subset construction. A complementary quantity is the pool alignment $A_{\text{pool}} = \cos\left(\nabla\ell_{\text{val}}, \frac{1}{N}\sum_i \nabla\ell_i\right)$, which measures whether the average candidate update already points at the target.

High A_{pool} together with low H_B describes a regime in which the average candidate is already useful and target-aligned signal is spread across the pool. Our results are consistent with this reading: Qwen3-0.6B/MMLU and the BBH column of Table 9 are settings where all methods, including full-data training, land within roughly one point of each other, whereas the settings with the largest spread between methods are also the ones where our method gains most. MMLU's coverage of 57 subjects is compatible with such diffuse support, though breadth alone is not the explanation, since Llama-3.2-1B/MMLU does benefit from selection; the Qwen backbone may simply already encode much of what MMLU tests.

We deliberately stop short of a stronger claim. Both quantities are one-step, first-order proxies that depend on the current parameters and the sampled validation minibatch; they ignore reweighting, the optimizer state, and error accumulation across steps, and we have not verified that they predict final downstream improvement. Turning H_B and A_{pool} into a calibrated, adaptive selection policy—for instance, falling back to uniform sampling when measured headroom is negligible—is an open direction.

E.13 Empirical Memory Frontier at Long Context

This appendix expands the complexity discussion in Section 3.3. While Table 1 establishes our $\mathcal{O}(T(N{+}V)d_1d_2)$ score-tensor cost against Ghost's $\mathcal{O}(NVT^2(d_1{+}d_2))$ asymptotically, whether this gap binds in practice depends on what else lives in GPU memory during a training step. The long-context feasibility frontier is shaped by two orthogonal memory axes—a *model-side floor* (weights, activations, and notably the cross-entropy logits buffer of size $\mathcal{O}(BTV)$, where $V \approx 128$K for Llama-3) and an *influence-side ceiling* (the score tensor of Eq. 8). Each axis must be addressed separately: an asymptotic advantage on one becomes empirically observable only after the other has been brought below it.

E.13.1 Influence-Side Microbenchmark

We isolate influence-side scaling with a pure-tensor microbenchmark on a single 80 GB H100. Synthetic gradient buffers of the exact pipeline shapes (activations $N \cdot T \times d_1$ and backprops $N \cdot T \times d_2$, analogously for validation tokens) are fed directly to the score-tensor kernel of Eq. 8 in both Ghost and our forms; the model forward and cross-entropy are bypassed entirely. The per-layer setting is slightly wider than the deployed one, which splits its budget as $(d_1, d_2) \in \{(32, 8), (8, 32)\}$; the wider choice costs both methods more arithmetic and leaves Ghost's $T \times T$ tensor unchanged, so it does not favor us. Setting: $V{=}4$, $d_1{=}d_2{=}32$, fp32 intermediates, 1 warmup + 3 timed iterations. Tables 17 and 18 report peak memory and wallclock.

Table 17: Empirical peak GPU memory (MB) of the score tensor under the Llama-3.2-1B LoRA setting. **Bold** = OOM on H100 80 GB.

N	Method	$T{=}512$	$T{=}1024$	$T{=}2048$	$T{=}4096$	$T{=}8192$	$T{=}16384$	$T{=}65536$
8	Ghost	136	439	1,651	6,489	25,829	**OOM**	**OOM**
	Ours	36	39	44	55	76	118	371
32	Ghost	441	1,654	6,495	25,842	**OOM**	**OOM**	**OOM**
	Ours	43	52	69	105	177	321	1,183
64	Ghost	848	3,273	12,955	51,646	**OOM**	**OOM**	**OOM**
	Ours	51	68	103	173	312	591	2,265
128	Ghost	1,662	6,511	25,874	**OOM**	**OOM**	**OOM**	**OOM**
	Ours	68	102	171	308	583	1,132	4,429

Table 18: Empirical wallclock (ms, mean over 3 calls) of one score-tensor invocation. Same setting as Table 17.

N	Method	T=512	T=1,024	T=2,048	T=4,096	T=8,192	T=16,384	T=65,536
8	Ghost	0.18	0.46	1.55	5.59	22.30	**OOM**	**OOM**
	Ours	0.20	0.13	0.12	0.15	0.08	0.12	0.35
64	Ghost	0.75	2.87	11.29	48.73	**OOM**	**OOM**	**OOM**
	Ours	0.08	0.09	0.12	0.20	0.32	0.60	2.05
128	Ghost	1.48	5.68	22.46	**OOM**	**OOM**	**OOM**	**OOM**
	Ours	0.08	0.11	0.17	0.30	0.57	1.03	3.97

The measurements validate the asymptotic prediction. Ghost's peak memory approaches $4\times$ per T-doubling ($3.2\times$ at the smallest sizes, $3.98\times$ at the largest), matching the T^2 term, while ours stays at or below $2\times$, matching the T term. Ghost's peak is $3NVT^2$ fp32 entries plus a fixed 35–105 MB of allocator overhead, which reproduces every measured cell to within 0.5%; the $3NVT^2$ term alone is within 0.5% above 6 GB and up to 35% low at the smallest cell, where the overhead dominates. The fit puts the 80 GB boundary at $N \cdot T^2 \approx 1.7 \times 10^9$ tokens^2: the largest feasible cell is N=64, T=4096 at 1.1×10^9, and every cell at 2.1×10^9 or above fails. Ours has no observed OOM at any cell tested. Wallclock follows the same scaling: at equal feasibility (N=8, T=8192), our score-step is $278\times$ faster than ghost's (0.08 ms against 22.3 ms).

E.13.2 End-to-End Frontier with Model-Side Mitigation

The microbenchmark above isolates the influence-side advantage, but end-to-end training is also bounded by the model-side floor. Under stock PyTorch, training in our reference setting (Llama-3.2-1B LoRA, N=8, V=4) OOMs at T=4096, with the binding allocation being the 15.6 GB cross-entropy logits buffer rather than the score tensor. We therefore activate Liger Kernel (Hsu et al.), which streams the lm_head matmul through softmax and cross-entropy without materializing the $(B \cdot T, V)$ logits tensor, reducing the loss-step memory from $\mathcal{O}(BTV)$ to $\mathcal{O}(BT \cdot \text{hidden})$ (an $\approx 60\times$ saving for Llama-3). Table 19 shows that Liger extends the end-to-end frontier from T=4096 to T=16384.

Table 19: End-to-end OOM frontier with and without Liger Kernel (Llama-3.2-1B LoRA, N=8, V=4, single H100 80 GB).

T	Stock PyTorch	+ Liger
2,048	✓	✓
4,096	**OOM** (15.6 GB CE)	✓
8,192	—	✓
16,384	—	✓
32,768	—	**OOM**

At this post-Liger frontier, the asymptotic gap from Table 1 becomes the binding constraint: at T=16384, N=8, V=4, the Ghost score tensor would require $\approx$ 103 GB ($3NVT^2$ fp32 entries under the calibration above), exceeding GPU capacity, while ours uses 118 MB. The two axes thus compose multiplicatively: model-side mitigation shrinks the method-agnostic floor (cross-entropy), and our influence-side decomposition shrinks the method-specific ceiling (score tensor). Notably, with better optimization, our method can scale up to even longer context.

Algorithm 1 `OMP-Select Sub-routine`

Input: Candidate set $\mathcal{S}^{(t)}$; Gram matrix $\mathbf{G}$; alignment vector $\mathbf{b}$; gradients $\{\nabla \ell_i\}$; preconditioned target $\nabla \tilde{\ell}_{\text{val}}$; budget k; ridge λ.
Output: Selected set $\mathcal{S}'$, weights $\mathbf{w}$.

Initialize selected set $\mathcal{S}' \leftarrow \emptyset$, weights $\mathbf{w} \leftarrow \emptyset$.
for $j = 1$ to k **do**
  Initialize best candidate u^*, min error $\varepsilon^* \leftarrow \infty$.
  for each candidate $u \in \mathcal{S}^{(t)} \setminus \mathcal{S}'$ **do**
    Form trial set $\mathcal{S}_{\text{trial}} \leftarrow \mathcal{S}' \cup \{u\}$.
    Solve weights: $\mathbf{w}^*_{\text{trial}} \leftarrow (\mathbf{G}_{\mathcal{S}_{\text{trial}}} + \lambda \mathbf{I})^{-1} \mathbf{b}_{\mathcal{S}_{\text{trial}}}$.
    Compute objective error: $\mathcal{L} \leftarrow \|\nabla \tilde{\ell}_{\text{val}} - \sum_{m \in \mathcal{S}_{\text{trial}}} w^*_m \nabla \ell_m\|^2 + \lambda \|\mathbf{w}^*_{\text{trial}}\|^2$.
    if $\mathcal{L} < \varepsilon^*$ **then**
      $u^* \leftarrow u$, $\varepsilon^* \leftarrow \mathcal{L}$, $\mathbf{w} \leftarrow \mathbf{w}^*_{\text{trial}}$.
    end if
  end for
  $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{u^*\}$.
end for
return $\mathcal{S}', \mathbf{w}$

Algorithm 2 `Filter-then-Weight at Step` t

Require: Model θ_{t-1}, Adam second moment $\hat{v}_{t-1}$; candidate pool $\mathcal{S}^{(t)}$ of size $N{=}\alpha k$, validation pool $\mathcal{S}^{(t)}_{\text{val}}$; batch k, ridge λ, learning rate η_t.
Ensure: Updated model θ_t.

Stage 0. One backward per micro-batch; hooks give the projected factors $\hat{a}, \hat{g}$ of every example.
$\phi_i \leftarrow \hat{a}_i^\top \hat{g}_i$ for $i \in \mathcal{S}^{(t)}$; $\;m \leftarrow \mathbf{D}_{t-1} \odot \sum_{j \in \mathcal{S}^{(t)}_{\text{val}}} \hat{a}_j^\top \hat{g}_j$
Standardize by the pool statistics μ, σ: $\phi_i \leftarrow (\phi_i - \mu)/\sigma$, $\;m \leftarrow m/\sigma$
$b_i \leftarrow \langle \phi_i, m \rangle$, $\;G_{ij} \leftarrow \langle \phi_i, \phi_j \rangle$
Stage 1. $\mathcal{S}' \leftarrow \emptyset$, $p \leftarrow 0$
while $|\mathcal{S}'| < k$ **do**
  $i^* \leftarrow \arg\max_{i \notin \mathcal{S}'} (b_i - p_i) \quad \{b_i - p_i = \langle \nabla \ell_i, r \rangle\}$
  $\mathcal{S}' \leftarrow \mathcal{S}' \cup \{i^*\}$; $\;p \leftarrow p + G_{\cdot\, i^*}$
end while
Stage 2. Re-form $\mathbf{G}_{\mathcal{S}'}, \mathbf{b}_{\mathcal{S}'}$ on $\mathcal{S}'$ under the same μ, σ
$\mathbf{w}^* \leftarrow \arg\min_{\mathbf{w} \geq 0} \mathbf{w}^\top \mathbf{G}_{\mathcal{S}'} \mathbf{w} - 2\mathbf{w}^\top \mathbf{b}_{\mathcal{S}'} + \lambda \bar{g} \|\mathbf{w}\|_2^2$, $\;\bar{g} = \frac{1}{k} \sum_{i \in \mathcal{S}'} G_{ii}$
Stage 3. $\theta_t \leftarrow \theta_{t-1} - \eta_t \, \mathcal{P}_t\big(\sum_{i \in \mathcal{S}'} w^*_i \nabla \ell_i\big)$