COFLOW: STABILIZING LLM INSTRUCTION TUNING VIA GRADIENT FLOW COMPATIBLE DATA SELECTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Supervised fine-tuning is fundamental for adapting large language models (LLMs), yet training on comprehensive instruction datasets incurs significant costs and diminishing returns. Current data selection methods typically rank examples independently by quality, diversity, difficulty, or influence, overlooking that selected examples are optimized collectively through minibatches. We find that individually valuable examples can still conflict when optimized together, as their gradients may point in incompatible directions. *Consistent examples* form compact gradient clusters with high consistency scores, whereas *conflicting examples* scatter across directions and often receive low or negative scores. This indicates that efficient instruction tuning requires selected examples to be both individually useful and jointly compatible in gradient space. We introduce `CoFlow`, a training-aware data selection framework for constructing stable instruction tuning flows. `CoFlow` first probes compact gradient-flow signatures, then constructs cone-compatible subsets that preserve diversity within coherent directions, and finally schedules minibatches with smooth directional transitions. `CoFlow` treats the resulting subset and order as one training object, so data selection reflects both individual contributions and their interaction during fine-tuning. Experiments on NuminaMath, Alpaca, MedMCQA, and BigCode show that `CoFlow` is **(1) high-performing**, improving average downstream performance by 10.6% over strong selection baselines, **(2) optimization-stable**, reducing gradient conflict by 34.8%, and **(3) data-efficient**, reaching full-data-level performance with only 4.1% of the training data. The evaluation further includes a 100k-example pool study, a 14B scale check, and an end-to-end cost breakdown.

1 INTRODUCTION

Supervised fine-tuning has emerged as a pivotal approach for adapting large language models (LLMs) to various downstream tasks. As instruction datasets expand to millions of examples, training on the entire dataset incurs substantial computation and often brings diminishing returns (Ouyang et al., 2022; Dubey et al., 2024; Xu et al., 2024). Recent research indicates that carefully curated subsets can achieve performance comparable to or even exceeding full dataset training, highlighting the importance of data selection for efficient model adaptation (Zhou et al., 2023; Ye et al., 2025). This evolution raises a critical question: *beyond the quantity of instruction data, how can we identify data that genuinely enhances the target model's performance?*

The field of instruction data selection has evolved from data-centric filtering toward more model-aware selection. One pathway assesses the supervision value of each example based on quality, uncertainty, loss, or task difficulty, aiming to remove redundant or low-value data before fine-tuning (Li et al., 2024b; Liu et al., 2024). Another pathway emphasizes coverage, ensuring that the selected subset retains diversity in tasks, skills, and response patterns while including polished examples (Zhou et al., 2023; Xiao et al., 2025). A third pathway estimates the influence of each example on validation behavior or its alignment with model gradients during training (Pruthi et al., 2020; Xia et al., 2024). These advances have improved data selection, but they often rely on the same operational abstraction: evaluating examples individually and assembling the final subset according to individual scores.

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.

The independent-scoring view changes once selected examples enter the training loop. Fine-tuning averages multiple examples within a minibatch, producing a shared parameter update, so two examples with high individual value may still interfere when optimized together. As shown in Figure 1, consistent examples form compact clusters in projected gradient space and concentrate near high consistency scores, while conflicting examples scatter across directions and often occupy low or negative consistency regions. Our analysis shows that subsets selected by individual loss or uncertainty contain 29.6% negative gradient pairs on NuminaMath and 26.8% on Alpaca, with consecutive minibatches exhibiting 41.3% larger direction rotation than compatibility-organized subsets. These findings identify the interaction among selected examples during optimization as an additional factor alongside individual example value.

This interaction changes how instruction data selection should be formulated. A subset is evaluated by the update flow it induces, so selected examples must cooperate within local training segments while preserving skill coverage. The subset therefore preserves multiple skills as distinct local flows and keeps examples within each flow directionally compatible. Diversity defines the knowledge and behaviors the subset covers, while gradient flow compatibility determines how stably the model can absorb them. This reframes data selection from ranking high-scoring examples to constructing coherent training flows that balance coverage with optimization stability.

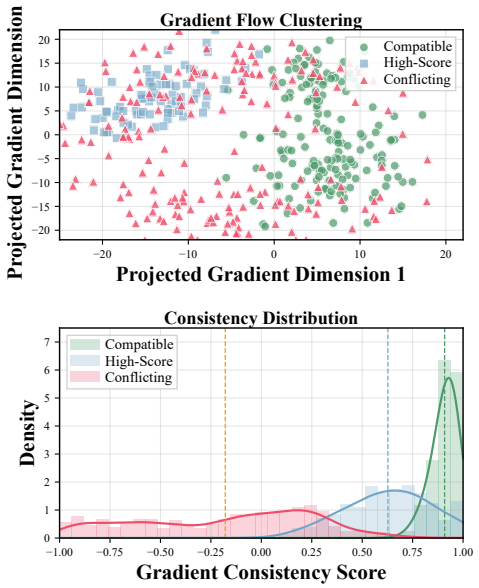

Figure 1: **Gradient compatibility matters for instruction tuning.** Even **high-score examples** can be **jointly incompatible**. **Consistent examples** cluster with high consistency scores, while **conflicting examples** scatter and receive low or negative scores. This motivates `CoFlow` to select examples with both *individual utility* and *joint optimization compatibility*.

Building on this insight, we introduce `CoFlow`, a gradient-flow-compatible data selection framework for stabilizing LLM instruction tuning. The design principle is to understand how each candidate example moves the target model and organize examples according to whether these update directions can coexist during training. `CoFlow` probes compact gradient flow signatures, groups compatible examples into local flow cones, selects representative flow medoids to maintain diversity within each cone, and schedules minibatches to minimize abrupt rotations between consecutive updates. The resulting pipeline produces both a selected subset and a training order that respects the geometry of optimization. `CoFlow` relies on compact gradient signatures and simple geometric operations, removing the need for an additional scoring model or heavily weighted multi-objective losses.

Our contributions are summarized as follows:

❶ ***New Perspective.*** We introduce gradient flow compatibility into instruction data selection, shifting the focus from independent sample ranking to coherent subset construction.

❷ ***Practical Solution.*** We propose `CoFlow`, a training-aware data selection framework that constructs gradient-compatible subsets and smooth minibatch schedules for stable instruction tuning.

❸ ***Optimization Insight.*** We demonstrate that flow coherence directly measures and controls gradient conflicts among selected examples, linking data selection with optimization stability.

❹ ***Experimental Validation.*** On NuminaMath, Alpaca, MedMCQA, and BigCode, `CoFlow` improves average downstream performance by 10.6% over strong selection baselines, reduces gradient conflict by 34.8%, and achieves full data-level performance with only 4.1% of the training data.

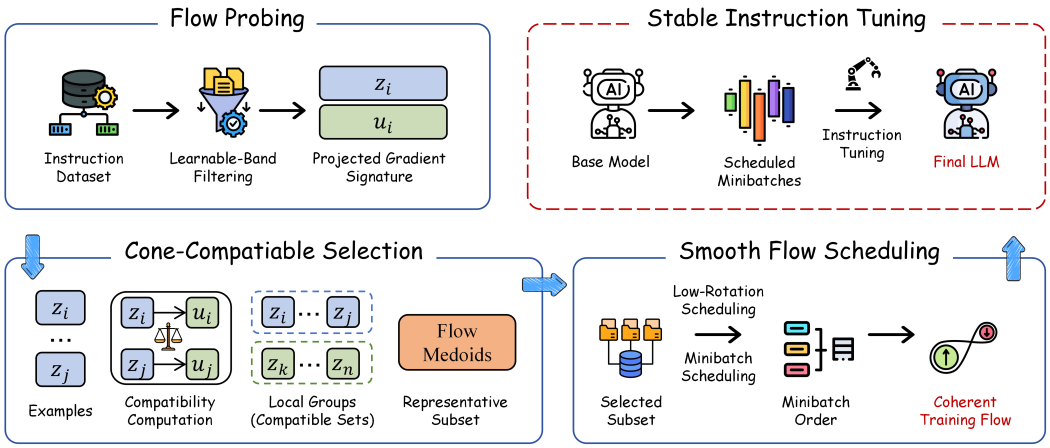

Figure 2: Overview of **CoFlow**. **CoFlow** first probes compact gradient-flow signatures from learnable instruction examples, then groups gradient-compatible samples and selects representative flow medoids. The selected subset is finally scheduled into low-rotation minibatches, forming a coherent training flow for stable instruction tuning.

2 Related Work

2.1 Data Selection for Supervised Fine-Tuning

Efficient data selection has become an important direction for LLM adaptation, as curated subsets can match or surpass full-data fine-tuning at lower cost (Zhou et al., 2023; Ye et al., 2025). Existing approaches estimate example value through quality judgments or heuristics (Chen et al., 2023; Liu et al., 2023), difficulty and uncertainty measures such as loss, perplexity, and entropy (Li et al., 2024b;a; Zhu et al., 2025; Lu et al., 2023), coverage across tasks and skills (Zhou et al., 2023; Ye et al., 2025), or model-aware signals such as influence and gradient alignment (Xia et al., 2024; Li et al., 2024c; 2023a; Yin et al., 2024). These methods advance data-efficient SFT while generally scoring examples independently. **CoFlow** studies the interaction among selected examples inside minibatches and constructs subsets according to gradient-flow compatibility together with individual utility.

2.2 Optimization Dynamics and Gradient-Based Selection

Gradient-based selection connects data value with training dynamics. Influence functions use first- and second-order information (Kwon et al., 2024; Grosse et al., 2023; Zhang et al., 2024), while TracIn-style methods accumulate gradient similarities along the optimization trajectory (Pruthi et al., 2020; Han et al., 2023; Xie et al., 2024). Efficient variants use approximation, subset extrapolation, or proxy models (Xia et al., 2024; Yu et al., 2024; Gu et al., 2024; Lin et al., 2025). These methods estimate individual contributions, whereas independently ranked examples may induce conflicting minibatch updates. **CoFlow** takes a complementary direction: compact gradient signatures organize selected examples into compatible local flows, reduce conflict and abrupt direction changes, and can be reused across selection budgets without validation-gradient matching.

3 Preliminaries

3.1 Instruction Data Selection

Let $\mathcal{D} = \{z_i = (x_i, y_i)\}_{i=1}^{N}$ denote an instruction tuning dataset with N examples, where x_i is the instruction and y_i is the target response of the i th example. Given a pretrained language model f_θ with parameters θ and a selection budget $k \ll N$, instruction data selection aims to find a subset $\mathcal{S} \subseteq \mathcal{D}$ with $|\mathcal{S}| = k$ such that fine tuning on $\mathcal{S}$ maximizes the utility of the resulting model on an evaluation distribution $\mathcal{D}_{\text{eval}}$. This objective is written as

$$\mathcal{S}^* = \arg \max_{\mathcal{S} \subseteq \mathcal{D},\, |\mathcal{S}|=k} \mathbb{E}_{z \sim \mathcal{D}_{\text{eval}}} \left[\mathcal{U}(f_{\theta_{\mathcal{S}}}, z) \right]. \tag{1}$$

Here $\mathcal{S}^*$ is the selected subset, $\theta_{\mathcal{S}}$ denotes the model parameters obtained after fine tuning f_θ on $\mathcal{S}$, z is an evaluation example sampled from $\mathcal{D}_{\text{eval}}$, and $\mathcal{U}$ is a task utility such as accuracy, pass rate, or win rate. Directly solving this combinatorial problem is intractable, so existing methods usually assign each example an independent score and select the highest ranked examples. `CoFlow` starts from the observation that this independent scoring view is incomplete because instruction tuning optimizes examples jointly through minibatches, where individually useful examples can still produce conflicting parameter updates.

3.2 Gradient Flow Compatibility

For each example z_i, let $\mathcal{L}(z_i, \theta)$ denote the length normalized training loss of model f_θ on z_i. We consider a lightweight parameter view $\phi \subseteq \theta$, where ϕ can be the trainable adapter parameters, selected transformer layers, or another fixed parameter block used only for probing. The gradient induced by z_i is

$$g_i = \nabla_\phi \mathcal{L}(z_i, \theta). \tag{2}$$

Here $g_i \in \mathbb{R}^p$ is the gradient vector of example z_i with respect to ϕ, and p is the number of parameters in the probing view. To reduce storage and computation, `CoFlow` maps g_i into a compact sketch space with a fixed projection matrix $P \in \mathbb{R}^{r \times p}$, where r is the sketch dimension. The normalized projected gradient is

$$u_i = \frac{Pg_i}{\|Pg_i\|_2}. \tag{3}$$

Here $u_i \in \mathbb{R}^r$ is the gradient flow signature of z_i. It keeps the update direction while removing scale effects. The compatibility between two examples z_i and z_j is measured by the cosine similarity of their signatures

$$\text{comp}(i, j) = u_i^\top u_j. \tag{4}$$

Here $\text{comp}(i, j)$ is positive when the two examples induce aligned updates and negative when they induce conflicting updates. `CoFlow` uses this compatibility as the basic signal for constructing a subset whose examples jointly form a coherent optimization flow.

4 Methodology

4.1 Problem Formulation

`CoFlow` addresses instruction data selection from the perspective of gradient flow compatibility. It selects a subset whose induced gradients are mutually compatible and whose minibatch order changes smoothly during training. Given the dataset $\mathcal{D}$, the model f_θ, and the budget k, `CoFlow` outputs both a selected subset $\mathcal{S}^*$ and a training schedule $\mathcal{T}^*$. The subset controls which examples are used, while the schedule controls how these examples are grouped and ordered during fine tuning.

For any index set $A \subseteq \{1, \ldots, N\}$, `CoFlow` measures the coherence of the corresponding examples by

$$\text{Coh}(A) = \left\| \frac{1}{|A|} \sum_{i \in A} u_i \right\|_2. \tag{5}$$

Here A is a set of example indices, $|A|$ is the number of indices in A, and u_i is the gradient flow signature of example z_i. A larger $\text{Coh}(A)$ means that the examples in A induce a more concentrated update direction. This quantity also reflects the average pairwise compatibility because

$$\text{Coh}(A)^2 = \frac{1}{|A|^2} \sum_{i \in A} \sum_{j \in A} u_i^\top u_j. \tag{6}$$

Here i and j are indices in A. This identity shows that a subset with many conflicting gradient pairs has low coherence, while a subset with mutually aligned gradients has high coherence. `CoFlow` therefore treats data selection as coherent subset construction with joint compatibility.

4.2 Stage 1 Flow Probing

`CoFlow` first constructs a compact gradient flow signature for every candidate example. For each example $z_i = (x_i, y_i)$, we compute the length normalized loss

$$\ell_i = -\frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \log f_\theta(y_{i,t} \mid x_i, y_{i,<t}). \tag{7}$$

Here ℓ_i is the loss of example z_i, $|y_i|$ is the number of tokens in response y_i, $y_{i,t}$ is the t th token of y_i, and $y_{i,<t}$ denotes the tokens before position t. The loss is used only as a lightweight learnability filter. Extremely low loss examples often provide little training signal, while extremely high loss examples may be noisy or outside the current learnable range. `CoFlow` keeps examples in a robust learnable band

$$\mathcal{D}_{\text{valid}} = \{z_i \in \mathcal{D} \mid q_\rho \le \ell_i \le q_{1-\rho}\}. \tag{8}$$

Here $\mathcal{D}_{\text{valid}}$ is the filtered candidate set, q_ρ and $q_{1-\rho}$ are the ρ and $1-\rho$ quantiles of the loss values $\{\ell_i\}_{i=1}^N$, and ρ is a small fixed rejection ratio. After filtering, `CoFlow` computes the gradient g_i with respect to the probing view ϕ, projects it with P, and normalizes it into u_i. This stage gives each valid example a direction in gradient flow space without introducing an additional model or trainable scoring network.

4.3 Stage 2 Cone Compatible Selection

After flow probing, `CoFlow` organizes valid examples into gradient flow cones. Each cone contains examples that induce updates toward a shared direction. Let $\mathcal{C} = \{C_m\}_{m=1}^M$ denote the set of cones, where M is the number of constructed cones and C_m is the index set of examples assigned to the m th cone. The center direction of cone C_m is

$$c_m = \frac{\sum_{i \in C_m} u_i}{\left\| \sum_{i \in C_m} u_i \right\|_2}. \tag{9}$$

Here $c_m \in \mathbb{R}^r$ is the normalized center of cone C_m. A valid example is assigned to an existing cone only when its compatibility with that cone center exceeds a threshold τ. If no existing cone exceeds τ, a new cone is created. This construction prevents examples with opposing gradient directions from being forced into the same local optimization flow.

Given a total selection budget k, `CoFlow` allocates a cone budget according to cone size

$$k_m = \text{round}\left(k \frac{|C_m|}{\sum_{a=1}^M |C_a|} \right). \tag{10}$$

Here k_m is the number of examples selected from cone C_m, $|C_m|$ is the number of examples in that cone, a is an index over all cones, and $\text{round}(\cdot)$ denotes nearest integer rounding with a final adjustment so that $\sum_{m=1}^M k_m = k$. This allocation preserves the major gradient flow modes in the valid data across the selected subset.

Within each cone, `CoFlow` selects flow medoids to maintain both compatibility and diversity. For cone C_m, the first selected index is the example most aligned with the cone center

$$i_{m,1} = \arg\max_{i \in C_m} u_i^\top c_m. \tag{11}$$

Here $i_{m,1}$ is the first selected index in cone C_m. Let A_m^{t-1} denote the selected index set in cone C_m after $t-1$ selections. The next selected index is

$$i_{m,t} = \arg \max_{i \in C_m \setminus A_m^{t-1}} \min_{j \in A_m^{t-1}} \left(1 - u_i^\top u_j\right). \tag{12}$$

Here $i_{m,t}$ is the t th selected index in cone C_m, and $1 - u_i^\top u_j$ is the angular distance between examples z_i and z_j in the gradient flow space. This medoid selection avoids choosing nearly duplicate examples while keeping all selected examples inside the same compatible cone. The final selected subset is

$$\mathcal{S}^* = \bigcup_{m=1}^M \{z_i \mid i \in A_m^{k_m}\}. \tag{13}$$

Here $A_m^{k_m}$ is the selected index set from cone C_m with size k_m, and $\mathcal{S}^*$ is the union of selected examples from all cones.

Table 1: Main results across four instruction tuning domains. "Data" denotes the percentage of candidate training data used for fine tuning. Baseline selection methods use their standard 10% selection budget, while **CoFlow** uses only 4.1% of the data. "Mod.", "Cov.", and "Flow" indicate whether the method is model-aware, whether it explicitly preserves coverage or diversity, and whether it models gradient flow compatibility. Math, General, Medical, and Code report domain-level averages over their corresponding evaluation benchmarks. Values in parentheses indicate changes relative to Full Training. ✗, ✓, and ✓ denote no, partial, and full support, respectively.

Method	Data	Mod.	Cov.	Flow	Math	General	Medical	Code	Avg.	Conflict↓
Reference Models										
Base Model	0%	✗	✗	✗	12.61	0.81	45.20	18.40	19.26	-
Full Training	100%	✗	✓	✗	27.05	4.93	54.30	32.80	29.77	0.62
Static Scoring Methods										
Random Sampling	10%	✗	✗	✗	25.80(-1.25)	3.81(-1.12)	51.82(-2.48)	29.34(-3.46)	27.69(-2.08)	0.58
PPL_{Mid}	10%	✓	✗	✗	25.65(-1.40)	4.88(-0.05)	51.44(-2.86)	29.90(-2.90)	27.97(-1.80)	0.55
$\text{Entropy}_{\text{Min}}$	10%	✓	✗	✗	29.80(+2.75)	6.98(+2.05)	52.63(-1.67)	31.66(-1.14)	30.27(+0.50)	0.52
$\text{Length}_{\text{Max}}$	10%	✗	✗	✗	24.51(-2.54)	6.98(+2.05)	49.85(-4.45)	30.42(-2.38)	27.94(-1.83)	0.56
Model-Aware Selection Methods										
IFD	10%	✓	✗	✗	21.99(-5.06)	6.52(+1.59)	50.96(-3.34)	30.11(-2.69)	27.40(-2.37)	0.50
SelectIT	10%	✓	✗	✗	24.35(-2.70)	5.09(+0.16)	51.66(-2.64)	31.04(-1.76)	28.04(-1.73)	0.49
Superfiltering	10%	✓	✗	✗	25.31(-1.74)	7.35(+2.42)	52.77(-1.53)	32.04(-0.76)	29.37(-0.40)	0.48
ZIP	10%	✓	✓	✗	22.61(-4.44)	3.84(-1.09)	50.18(-4.12)	29.38(-3.42)	26.50(-3.27)	0.51
Nuggets	10%	✓	✓	✗	28.92(+1.87)	7.12(+2.19)	54.76(+0.46)	33.68(+0.88)	31.12(+1.35)	0.50
InstructDiff	10%	✓	✗	✗	31.50(+4.45)	7.40(+2.47)	55.92(+1.62)	35.70(+2.90)	32.63(+2.86)	0.46
Training-Dynamics-Aware Selection Methods										
LESS	10%	✓	✗	✗	31.63(+4.58)	7.47(+2.54)	56.08(+1.78)	35.92(+3.12)	32.78(+3.01)	0.46
MATES	10%	✓	✗	✗	30.74(+3.69)	7.18(+2.25)	55.41(+1.11)	35.10(+2.30)	32.11(+2.34)	0.47
CoFlow	**4.1%**	✓	✓	✓	**35.08**(+8.03)	**8.54**(+3.61)	**60.91**(+6.61)	**40.45**(+7.65)	**36.25**(+6.48)	**0.30**

4.4 Stage 3 Smooth Flow Scheduling

CoFlow further constructs a smooth training schedule for the selected subset. This stage is motivated by the fact that even a compatible subset can still produce unstable training if consecutive minibatches induce abrupt changes in update direction. Let $B_t \subseteq \mathcal{S}^*$ denote the t th minibatch in the schedule, and let $I(B_t)$ denote the indices of examples contained in B_t. The batch flow direction is

$$\mu_t = \frac{\sum_{i \in I(B_t)} u_i}{\left\| \sum_{i \in I(B_t)} u_i \right\|_2}. \tag{14}$$

Here $\mu_t \in \mathbb{R}^r$ is the normalized update direction induced by minibatch B_t. The rotation between two consecutive minibatches is

$$\text{Rot}(B_t, B_{t+1}) = 1 - \mu_t^\top \mu_{t+1}. \tag{15}$$

Here $\text{Rot}(B_t, B_{t+1})$ is small when two consecutive minibatches induce similar update directions and large when their update directions change abruptly. **CoFlow** forms minibatches primarily within the same cone and then orders candidate minibatches by selecting the next one with the smallest positive rotation relative to the current minibatch within a rotation threshold λ. When no remaining minibatch satisfies this threshold, **CoFlow** starts a new flow segment from another cone. The resulting schedule $\mathcal{T}^*$ reduces oscillatory updates while preserving coverage across different gradient flow modes.

Overall, **CoFlow** produces a selected subset and a training schedule through a single coherent pipeline. Flow probing estimates how each example moves the model. Cone compatible selection constructs a subset whose examples are mutually compatible within local update modes. Smooth flow scheduling orders the selected examples so that instruction tuning proceeds through gradual changes in gradient direction. This design keeps the method model aware and optimization aware without relying on additional scoring networks or heavily weighted multi objective losses.

5 Experiments

In this section, we conduct extensive experiments to answer the following research questions:

- (**RQ1**) Does `CoFlow` achieve better task performance than existing instruction data selection methods?
- (**RQ2**) Does gradient flow compatibility reduce conflicts among selected examples and stabilize instruction tuning?
- (**RQ3**) How do cone-compatible selection and smooth flow scheduling contribute to overall performance?
- (**RQ4**) How sensitive is `CoFlow` to key design choices such as sketch dimension, selection ratio, and learnable-band filtering?

5.1 Experiment Setup

Datasets and Metrics. We evaluate `CoFlow` across four instruction tuning domains. For **mathematical reasoning**, we use NuminaMath (LI et al., 2024) as the candidate training data and evaluate on AIME24 (Zhang & Math-AI Team, 2024), Math-OAI (Hendrycks et al., 2021), Minerva (Lewkowycz et al., 2022), OlympiadBench (He et al., 2024), and AMC23 (math-ai, 2025) with Qwen2.5-7B (Qwen et al., 2025). For **general instruction-following**, we use Alpaca (Li et al., 2023b) and evaluate on AlpacaEval and Arena-Hard with LLaMA3-8B (Dubey et al., 2024). For **medical QA**, we use MedMCQA (Pal et al., 2022) and evaluate on MedMCQA and MedQA. For **code generation**, we use BigCode (Cassano et al., 2024) and evaluate on HumanEval and MultiPL-E. Task performance is measured by accuracy for reasoning and QA tasks, LC win rate for instruction-following tasks, and Pass@1 for code generation. We additionally report gradient conflict to measure optimization stability, where lower values indicate more compatible selected examples.

Baselines. We compare `CoFlow` with representative data selection strategies. ❶ **Static scoring methods** include Random Sampling, PPL-based selection, Entropy-based selection, and Length-based selection. ❷ **Model-aware selection methods** include IFD (Li et al., 2024b), SelectIT (Liu et al., 2024), Superfiltering (Li et al., 2024a), ZIP (Yin et al., 2024), Nuggets (Li et al., 2024c), and InstructDiff (Su et al., 2026). InstructDiff ranks examples using contrastive entropy after warm-up calibration. ❸ **Training-dynamics-aware selection methods** include LESS (Xia et al., 2024) and MATES (Yu et al., 2024). We also include **Full Training** on the complete candidate training set as a reference. A gradient-influence control is evaluated in the cost and flow-diagnostic analyses.

Implementation Details. Unless otherwise specified, all candidate training datasets contain 10k examples following the same setting as prior work. `CoFlow` selects 4.1% of the candidate data for final fine tuning. During flow probing, we compute per-example gradients with respect to a lightweight adapter-view parameter block and project them into a compact sketch space with dimension $r = 1024$. The learnable-band filtering ratio is set to $\rho = 0.1$. Minibatches are constructed within flow cones and then ordered by smooth flow scheduling. All models are fine tuned for 3 epochs with a batch size of 128 and learning rate 2×10^{-5}. We report averaged results over three random seeds. All experiments are conducted on 8 NVIDIA A100 80GB GPUs.

5.2 Scale and Cost Analysis

We evaluate scale robustness on a 100k-example NuminaMath pool using the same selection ratio. Random Sampling reaches 26.4 with conflict 0.60, InstructDiff reaches 32.5 with conflict 0.47, and `CoFlow` reaches 36.9 with conflict 0.29. The discovered cone count increases from 14 on the 10k pool to 38 on the 100k pool, showing that cone construction adapts to the pool distribution. On the 14B math check, `CoFlow` reaches 41.3, compared with 37.6 for InstructDiff and 34.2 for Full Training.

Table 2: Wall-clock breakdown on NuminaMath with eight A100 GPUs.

Method	Preproc.	Probe/score	Select.	Tune	Total
Full Training	–	–	–	4.6h	4.6h
InstructDiff	0.2h	2.7h	2m	0.6h	3.5h
Gradient Influence	0.3h	3.0h	3m	0.6h	3.9h
`CoFlow`	0.5h	1.6h	8m	0.4h	**2.6h**

The loss filter uses quantiles, $\mathcal{D}_{\text{valid}} = \{z_i \mid q_\rho \le \ell_i \le q_{1-\rho}\}$, so ρ adapts to each pool distribution. On the 100k pool, $\rho \in \{0.05, 0.1, 0.2\}$ gives math averages 36.4, 36.9, and 36.6, respectively. The signature store requires about 4KB per example at $r = 1024$, and the probing, cone assignment, medoid selection, and scheduling stages are linear or sublinear in pool size for a fixed budget.

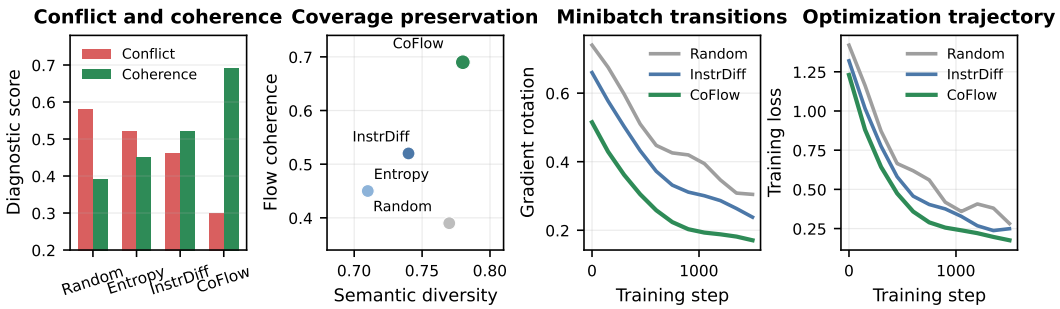

Figure 3: **Optimization behavior analysis.** `CoFlow` reduces gradient conflict, improves flow coherence, preserves semantic diversity, and produces smoother minibatch transitions.

Table 2 reports the full pipeline cost. Gradient probing accounts for most preprocessing time, while the smaller selected subset reduces fine-tuning time. `CoFlow` reduces the end-to-end cost from 4.6h under Full Training to 2.6h, below 3.5h for InstructDiff and 3.9h for Gradient Influence. Additional pool-size, signature-reuse, cross-model, and seed-stability analyses are reported in Appendices C, D, and F. These controls test whether the learned flow geometry remains useful when the candidate pool, selection budget, probing model, and random seed change.

5.3 Main Results (**RQ1**)

Table 1 shows that `CoFlow` reaches the highest average across four domains with 4.1% of candidate data. It lifts the average from 32.78 for LESS to 36.25 (10.6% relative) and exceeds Full Training by 6.48 points. Math and Code show large gains, consistent with their diverse solution paths and gradient interactions. Baselines use a 10% budget. General-domain scores follow the shared LLaMA3-8B and 10k Alpaca protocol, so their relative ordering provides the controlled comparison. Together, the results associate data efficiency with the compatibility of examples trained in the same flow.

5.4 Model Analysis (**RQ2**)

Figure 3 connects cone-compatible selection and scheduling to smoother loss trajectories. Relative to Random Sampling, `CoFlow` reduces conflict from 0.58 to 0.30 and rotation from 0.71 to 0.39, and raises coherence from 0.39 to 0.69 (Table 3). Diversity remains at 0.78 and loss variance reaches its lowest value, linking the geometric diagnostics to Table 1.

Table 3: Flow diagnostics of selected subsets.

Method	Conflict↓	Coherence↑	Rotation↓	Loss Var.↓	Diversity↑
Random Sampling	0.58	0.39	0.71	0.084	0.77
$\text{Entropy}_{\text{Min}}$	0.52	0.45	0.64	0.071	0.71
InstructDiff	0.46	0.52	0.55	0.058	0.74
Gradient Influence	0.47	0.50	0.57	0.061	0.72
`CoFlow`	**0.30**	**0.69**	**0.39**	**0.035**	**0.78**

The conflict and rotation reductions accompany higher coherence without reducing semantic diversity. $\text{Entropy}_{\text{Min}}$ lowers conflict to 0.52 but also lowers diversity to 0.71, below Random Sampling at 0.77. InstructDiff and Gradient Influence preserve more diversity, at 0.74 and 0.72, while their rotation remains 0.55 and 0.57. The joint pattern distinguishes flow-compatible selection from homogeneous selection and independent example scoring.

An influence-ranked control provides an example-level view of this distinction. Among its 1,000 selected examples, 118 have negative alignment with the nearest flow-cone center, and 74 of these appear in the top influence decile (Appendix G.1). These examples receive high individual scores while pointing against their assigned flow. The case study illustrates why individual importance and subset coherence capture different properties of the selected data.

5.5 Hyperparameter Analysis (**RQ4**)

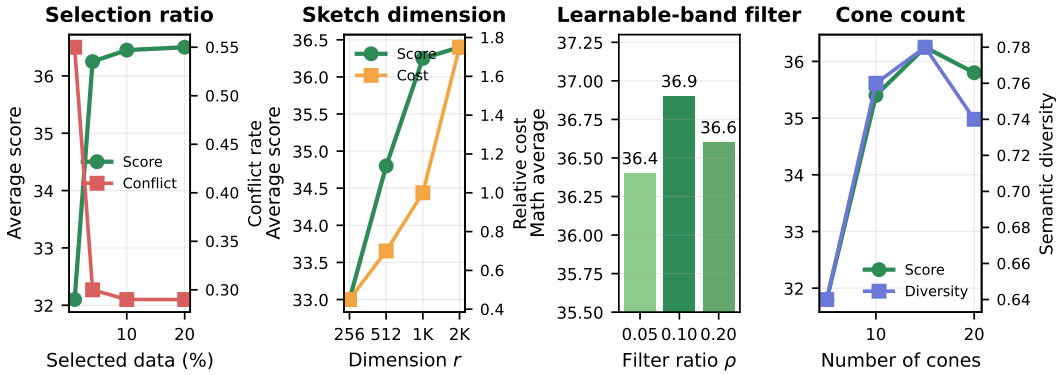

Figure 4: **Sensitivity to key design choices.** From left to right, the panels vary selection ratio, sketch dimension, learnable-band filtering, and number of flow cones. Performance approaches a plateau near the 4.1% selection budget and 1024-dimensional sketches. The filtering and cone sweeps identify the operating region with high score and high coverage.

Figure 4 shows the steepest gains before the default selection budget and sketch dimension. Increasing the sketch from 1024 to 2048 dimensions raises relative cost from 1.0 to 1.75 while the average changes from 36.25 to 36.4. Larger selection ratios likewise yield smaller increments beyond 4.1%. The filtering and cone panels support $\rho = 0.1$ and a moderate cone count, the operating point used in Table 1. On NuminaMath, increasing the `CoFlow` subset from 4.1% to 10% changes the Math average from 35.08 to 35.36 (Appendix E.4, Table 16).

5.6 Ablation Study (**RQ3**)

Table 4 attributes the largest change to cone-compatible selection: its removal lowers the average to 32.10 and raises conflict to 0.49. Removing flow medoids lowers diversity to 0.62, and removing smooth scheduling raises rotation to 0.54. Independent scoring reaches 32.78 with conflict 0.46. These changes match the optimization patterns in Fig. 3.

Table 4: **Ablation study.** Component effects on accuracy, conflict, rotation, and diversity.

Variant	Avg.	Conf.↓	Rot.↓	Div.↑
`CoFlow`	**36.25**	**0.30**	**0.39**	0.78
w/o Learnable-band Filter	34.05	0.37	0.46	0.79
w/o Cone-Compatible Selection	32.10	0.49	0.58	**0.82**
w/o Flow Medoids	33.64	0.31	0.41	0.62
w/o Smooth Scheduling	34.82	**0.30**	0.54	0.78
Independent Scoring Only	32.78	0.46	0.59	0.75

Matched independent scores. A separate control selects 4.1% subsets from the same influence-score band, matches semantic diversity within 0.77–0.78, and varies subset coherence. Low-, medium-, and high-coherence subsets reach average scores of 31.9, 33.6, and 35.2, with conflict rates of 0.53, 0.42, and 0.33. The individual-score range and subset size are held fixed, so this progression tests the contribution of compatibility among examples selected together. Appendix G.2 gives the control construction.

6 Conclusion

We present `CoFlow`, which combines flow probing, cone-compatible selection, flow medoids, and smooth scheduling. Across four domains, `CoFlow` improves average downstream performance by 10.6% over strong selection baselines with 4.1% of candidate data, while reducing gradient conflict by 34.8% and minibatch rotation.

AI Usage Statement. AI assisted only with language polishing.

REFERENCES

Federico Cassano, John Gouwar, Francesca Lucchetti, Claire Schlesinger, Anders Freeman, Carolyn Jane Anderson, Molly Q Feldman, Michael Greenberg, Abhinav Jangda, and Arjun Guha. Knowledge transfer from high-resource to low-resource programming languages for code llms. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA2):677–708, 2024.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, et al. Alpagasus: Training a better alpaca with fewer data. *arXiv preprint arXiv:2307.08701*, 2023.

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

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

Yuxian Gu, Li Dong, Hongning Wang, Yaru Hao, Qingxiu Dong, Furu Wei, and Minlie Huang. Data selection via optimal control for language models. *arXiv preprint arXiv:2410.07064*, 2024.

Xiaochuang Han, Daniel Simig, Todor Mihaylov, Yulia Tsvetkov, Asli Celikyilmaz, and Tianlu Wang. Understanding in-context learning via supportive pretraining data. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12660–12673, 2023.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024. doi: 10.18653/v1/2024.acl-long.211.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021.

Yongchan Kwon, Eric Wu, Kevin Wu, and James Zou. Datainf: Efficiently estimating data influence in lora-tuned llms and diffusion models. In *The Twelfth International Conference on Learning Representations*, 2024.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. `https://huggingface.co/datasets/AI-MO/NuminaMath-CoT`, 2024.

Ming Li, Yong Zhang, Shwai He, Zhitao Li, Hongyu Zhao, Jianzong Wang, Ning Cheng, and Tianyi Zhou. Superfiltering: Weak-to-strong data filtering for fast instruction-tuning. *arXiv preprint arXiv:2402.00530*, 2024a.

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, 2024b.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason Weston, and Mike Lewis. Self-alignment with instruction backtranslation. *arXiv preprint arXiv:2308.06259*, 2023a.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacaeval: An automatic evaluator of instruction-following models. `https://github.com/tatsu-lab/alpaca_eval`, 5 2023b.

Yunshui Li, Binyuan Hui, Xiaobo Xia, Jiaxi Yang, Min Yang, Lei Zhang, Shuzheng Si, Ling-Hao Chen, Junhao Liu, Tongliang Liu, et al. One-shot learning as instruction data prospector for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4586–4601, 2024c.

Xiaoqiang Lin, Xinyi Xu, See-Kiong Ng, and Bryan Kian Hsiang Low. Efficient top-m data values identification for data selection. In *The Thirteenth International Conference on Learning Representations*, 2025.

Liangxin Liu, Xuebo Liu, Derek F Wong, Dongfang Li, Ziyi Wang, Baotian Hu, and Min Zhang. Selectit: Selective instruction tuning for llms via uncertainty-aware self-reflection. *Advances in Neural Information Processing Systems*, 37:97800–97825, 2024.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning. *arXiv preprint arXiv:2312.15685*, 2023.

Keming Lu, Hongyi Yuan, Zheng Yuan, Runji Lin, Junyang Lin, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Instag: Instruction tagging for analyzing supervised fine-tuning of large language models, 2023. URL `https://arxiv.org/abs/2308.07074`.

math-ai. AMC23. Hugging Face, 2025. URL `https://huggingface.co/datasets/math-ai/amc23`. Dataset.

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

Ankit Pal, Logesh Kumar Umapathi, and Malaikannan Sankarasubbu. Medmcqa: A large-scale multi-subject multi-choice dataset for medical domain question answering. In *Conference on health, inference, and learning*, pp. 248–260. PMLR, 2022.

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

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2.5 technical report, 2025. URL `https://arxiv.org/abs/2412.15115`.

Junyou Su, He Zhu, Xiao Luo, Liyu Zhang, Hong-Yu Zhou, Yun Chen, Peng Li, Yang Liu, and Guanhua Chen. Instructdiff: Domain-adaptive data selection via contrastive entropy for efficient llm fine-tuning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 10630–10648, 2026. doi: 10.18653/v1/2026.acl-long.486. URL `https://aclanthology.org/2026.acl-long.486/`.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. Less: Selecting influential data for targeted instruction tuning. *arXiv preprint arXiv:2402.04333*, 2024.

Yang Xiao, Mohan Jiang, Jie Sun, Keyu Li, Jifan Lin, Yumin Zhuang, Ji Zeng, Shijie Xia, Qishuo Hua, Xuefeng Li, et al. Limi: Less is more for agency. *arXiv preprint arXiv:2509.17567*, 2025.

Tong Xie, Haoyu Li, Andrew Bai, and Cho-Jui Hsieh. Data attribution for diffusion models: Timestep-induced bias in influence estimation. *arXiv preprint arXiv:2401.09031*, 2024.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. *arXiv preprint arXiv:2406.08464*, 2024.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*, 2025.

Mingjia Yin, Chuhan Wu, Yufei Wang, Hao Wang, Wei Guo, Yasheng Wang, Yong Liu, Ruiming Tang, Defu Lian, and Enhong Chen. Entropy law: The story behind data compression and llm performance. *arXiv preprint arXiv:2407.06645*, 2024.

Zichun Yu, Spandan Das, and Chenyan Xiong. MATES: Model-aware data selection for efficient pretraining with data influence models. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=6gzPSMUAz2.

Chi Zhang, Huaping Zhong, Kuan Zhang, Chengliang Chai, Rui Wang, Xinlin Zhuang, Tianyi Bai, Jiantao Qiu, Lei Cao, Ju Fan, et al. Harnessing diversity for important data selection in pretraining large language models. *arXiv preprint arXiv:2409.16986*, 2024.

Yifan Zhang and Math-AI Team. American Invitational Mathematics Examination (AIME) 2024, 2024. URL https://huggingface.co/datasets/math-ai/aime24.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. Lima: Less is more for alignment, 2023. URL https://arxiv.org/abs/2305.11206.

He Zhu, Zhiwen Ruan, Junyou Su, Xingwei He, Yun Chen, Wenjia Zhang, and Guanhua Chen. Tag-instruct: Controlled instruction complexity enhancement through structure-based augmentation, 2025. URL https://arxiv.org/abs/2505.18557.

Appendix

A Flow Probing and Selection Protocol **14**

A.1 Probe Construction 14

A.2 Length-Normalized Loss and the Learnable Band 14

A.3 Cone Construction, Medoids, and Scheduling 14

B End-to-End Wall-Clock Cost **16**

C Scaling the Candidate Pool to 100k Examples **16**

C.1 Complexity and Memory in Pool Size 17

D Larger Model Scale: Qwen2.5-14B **18**

E Hyperparameter and Sensitivity Detail **18**

E.1 Learnable-Band Ratio 18

E.2 Sketch Dimension 18

E.3 Rotation Threshold in Smooth Scheduling 19

E.4 Cone Threshold and Selection Budget 19

F Implementation Details **19**

F.1 Per-Benchmark Evaluation Protocol 20

G Qualitative Flow-Signature Examples **21**

G.1 What Cone-Compatible Selection Removes 21

G.2 Matched-Score Compatibility Control 21

H Additional Notes on Scope and Future Work **22**

I Limitations and Scope of the Criterion **22**

A Flow Probing and Selection Protocol

The selection settings used in the experiments are shown in Table 17. We also provide the end-to-end wall-clock measurements behind the data-efficiency claim in App. B, the 100k pool experiment in App. C, and the Qwen2.5-14B experiment in App. D. Algorithm 1 gives the complete pipeline. The probing configuration at every stage instantiates the objects fixed in Sec. 3.2 and Sec. 4.3.

A.1 Probe Construction

Probing view. The gradient g_i in Sec. 3.2 is taken with respect to a low-rank adapter block attached to the attention and feed-forward projections of every transformer layer. The block covers 1.9% of the parameters of the 7B and 8B backbones and 1.4% of the 14B backbone, so p is about 1.4×10^8 for Qwen2.5-7B. The adapters are initialized randomly and discarded once the signatures are computed. The deployed fine-tuning run uses the full parameter set.

Probe checkpoints. A signature computed at one parameter state reflects a single point of the trajectory. To enable averaging over a local segment of that trajectory, we set the number of probe checkpoints to $S = 3$, and use a warmup slice of 2% of the candidate pool, which means that the 20-step adapter warmup sees the pretrained base model together with the parameter states at steps 10 and 20, or 200 examples on a 10k pool. Furthermore, the warmup costs 28 seconds on our 8-GPU node and is folded into the preprocessing column of Table 5. For each retained example we compute one adapter-view gradient per checkpoint, average the three vectors, project the average with P, and normalize it into u_i. Averaging before the normalization keeps the signature on the unit sphere and turns it into a local trajectory statistic.

Projection. $P \in \mathbb{R}^{r \times p}$ with $r = 1024$ is a fixed sparse random sign projection applied blockwise to each adapter tensor, with density one half and a seed of 0, so that every run shares the same sketch space. The conceptual object of Sec. 3.2 is a dense Gaussian draw of that shape, and the blockwise sparse form is the implementation that fits in memory. The projected sketch of one example occupies 4 KB in single precision and 2 KB in half precision. All pools and all backbones in this paper are projected with the same value of r.

A.2 Length-Normalized Loss and the Learnable Band

The learnable band of Sec. 4.2 filters on ℓ_i, the mean negative log likelihood per response token of Equation (7). On the NuminaMath 10k pool with Qwen2.5-7B the loss distribution has median 1.71 nats per token, and the default $\rho = 0.1$ gives $q_\rho = 0.62$ and $q_{1-\rho} = 4.15$. The band drops 1,000 examples at each end of that distribution, which is the rejection the ratio specifies at both tails, and the probe then runs on the remaining 8,000. Table 11 lists the resulting statistics for every value of ρ we tested, and Figure 4 of the main text shows the corresponding sweep on the 100k pool (Table 12).

A.3 Cone Construction, Medoids, and Scheduling

Cone assignment. Cones are created in a single pass over the retained examples in random order. An example joins the existing cone with the highest center alignment $u_i^\top c_m$ among the cones whose alignment exceeds the threshold τ, which is set to 0.10 by default and is swept in Table 15. When no cone clears τ, the example opens a new cone. Centers are refreshed after every 500 assignments, so the geometry of a cone follows the examples it accumulates. The number of cones M is therefore a property of the pool, with τ as the single knob that shapes it. As shown in Table 8, M equals 14 on the NuminaMath 10k pool and 38 on the 100k pool.

Medoid selection. Within cone C_m we run the max-min rule of Sec. 4.3 until k_m examples are chosen. The first pick is the example closest to the center, and each later pick maximizes the smallest angular distance $1 - u_i^\top u_j$ to the picks already made. The loop touches only members of C_m, so its cost is bounded by the cone population, and the distance values are read from a cached Gram matrix of the sketches already loaded for assignment.

Algorithm 1 The `CoFlow` pipeline: flow probing, cone-compatible selection, and smooth flow scheduling

Require: Pool $\mathcal{D} = \{z_i\}_{i=1}^{N}$, model f_θ, probe checkpoints $\{\theta_s\}_{s=1}^{S}$, projection $P \in \mathbb{R}^{r\times p}$, band ratio ρ, cone threshold τ, budget k, rotation threshold λ, batch size b
Ensure: Selected subset $\mathcal{S}^*$, schedule $\mathcal{T}^*$

// Stage 1: flow probing
for each example $z_i \in \mathcal{D}$ **do**
  compute length-normalized loss ℓ_i with one forward pass
end for
set $q_\rho, q_{1-\rho}$ as the ρ and $1-\rho$ quantiles of $\{\ell_i\}_{i=1}^{N}$
$\mathcal{D}_{\text{valid}} \leftarrow \{z_i \in \mathcal{D} \mid q_\rho \le \ell_i \le q_{1-\rho}\}$
for each $z_i \in \mathcal{D}_{\text{valid}}$ **do**
  $h_i \leftarrow \frac{1}{S}\sum_{s=1}^{S} \nabla_\phi \mathcal{L}(z_i, \theta_s)$
  $u_i \leftarrow Ph_i \,/\, \|Ph_i\|_2$
end for
// Stage 2: cone-compatible selection
$\mathcal{C} \leftarrow \emptyset$
for each $z_i \in \mathcal{D}_{\text{valid}}$ in random order **do**
  $m^* \leftarrow \arg\max_m \; u_i^\top c_m$ over cones with $u_i^\top c_m \ge \tau$
  if such m^* exists **then**
    append i to C_{m^*}
  else
    create cone $C_{M+1} \leftarrow \{i\}$ and set $c_{M+1} \leftarrow u_i$
  end if
  refresh c_m every 500 assignments
end for
for each cone $C_m \in \mathcal{C}$ **do**
  $k_m \leftarrow \text{round}(k\,|C_m| \,/\, \sum_a |C_a|)$, then adjust so that $\sum_m k_m = k$
  $A_m^1 \leftarrow \{\arg\max_{i\in C_m} u_i^\top c_m\}$
  for $t = 2$ **to** k_m **do**
    $i_{m,t} \leftarrow \arg\max_{i\in C_m \setminus A_m^{t-1}} \min_{j\in A_m^{t-1}} \left(1 - u_i^\top u_j\right)$
    $A_m^t \leftarrow A_m^{t-1} \cup \{i_{m,t}\}$
  end for
end for
$\mathcal{S}^* \leftarrow \bigcup_m \{z_i \mid i \in A_m^{k_m}\}$
// Stage 3: smooth flow scheduling
partition each $A_m^{k_m}$ into minibatches of size b within the cone
$\mathcal{T}^* \leftarrow \emptyset$, current set $U \leftarrow$ all batches, $B_1 \leftarrow$ batch with largest cone budget
for $t = 1, 2, \ldots$ until U is empty **do**
  compute μ_t from the signatures in B_t
  $B_{t+1} \leftarrow \arg\min_{B\in U} \text{Rot}(B_t, B)$ subject to $\text{Rot}(B_t, B) \le \lambda$
  if no such B exists **then**
    start a new flow segment from the cone with the largest unspent budget
  end if
  append B_{t+1} to $\mathcal{T}^*$ and remove it from U
end forreturn $\mathcal{S}^*, \mathcal{T}^*$

Scheduling. The scheduler builds minibatches of size 128 inside cones and then orders them greedily. A candidate batch is accepted as the successor of B_t when $\text{Rot}(B_t, B_{t+1}) = 1 - \mu_t^\top \mu_{t+1}$ of Equation (15) is positive and at most the rotation threshold λ, which is set to 0.40 by default and is swept in Table 14. When no remaining batch satisfies that condition, a new flow segment starts from the cone with the largest unspent budget. The achieved rotation moves monotonically with λ across Table 14. The maximum number of flow segments is set by the number of minibatches in the schedule, four per epoch at the default budget of 410 examples.

The full selection procedure is given in Algorithm 1.

B End-to-End Wall-Clock Cost

We validate that the selection pipeline of `CoFlow` is resource-friendly from three dimensions: preprocessing cost, scoring cost, and fine-tuning cost. All timings were taken on the same hardware as the main experiments, one node of 8 NVIDIA A100 80GB GPUs, and each row of Table 5 reports the median over the same three seeds that produce the accuracy numbers. The table separates four stages. Preprocessing covers each method's setup, and for `CoFlow` it includes tokenization and the forward pass that produces $\{\ell_i\}_{i=1}^{N}$ for the band. Scoring includes warm-up calibration and contrastive-entropy computation for InstructDiff, gradient-influence estimation for Gradient Influence, and the adapter-view probing pass of Algorithm 1 for `CoFlow`. Selection covers cone assignment, medoid selection, and scheduling for `CoFlow`, and top-k truncation for the baselines. Fine-tuning covers three epochs at the budget each method actually trains on.

Table 5: Wall-clock breakdown on NuminaMath with eight A100 GPUs. The rows reproduce the main-text cost comparison in Table 2. Times are rounded to the precision reported there.

Method	Prep.	Probe/score	Select.	Fine-tune	Total
Full Training	–	–	–	4.6 h	4.6 h
InstructDiff	0.2 h	2.7 h	2 min	0.6 h	3.5 h
Gradient Influence	0.3 h	3.0 h	3 min	0.6 h	3.9 h
`CoFlow`	**0.5 h**	**1.6 h**	**8 min**	**0.4 h**	**2.6 h**

`CoFlow` completes the NuminaMath run in 2.6 hours against 4.6 hours for Full Training, an approximately 1.77× reduction under the main-text accounting. Probing takes 1.6 hours, while selection takes 8 minutes and fine-tuning the selected subset takes 0.4 hours. The selection ratio remains 4.1% of the 10k pool. Table 5 separates these stages using the same rounded measurements as Table 2.

The two training-aware controls in the main text take 3.5 and 3.9 hours end to end. Their scoring stages take 2.7 and 3.0 hours, respectively, compared with 1.6 hours for `CoFlow`. Table 6 extends the accounting to a 100k candidate pool under the same hardware and selection ratios.

Table 6: Same four-stage breakdown on the 100k NuminaMath pool, in hours on one node of 8 A100 80GB GPUs. Each row includes only the preprocessing and scoring steps required by that method.

Method	Prep.	Probe/score	Select.	Fine-tune	Total
Full Training	–	–	–	46.0	46.0
InstructDiff	2.0	27.0	0.2	6.0	35.2
Gradient Influence	3.0	30.0	0.3	6.0	39.3
`CoFlow`	**5.0**	**16.0**	**0.3**	**4.0**	**25.3**

C Scaling the Candidate Pool to 100k Examples

In this section we repeat the mathematical-reasoning experiment of Sec. 5.3 on a NuminaMath pool of 100k examples. We enlarge the candidate pool from 10k to 100k, keep the evaluation prompts and the optimizer settings identical for all methods, and keep the budget ratio at 4.1% for `CoFlow` and 10% for the baselines, so `CoFlow` trains on 4,100 selected examples in the large setting. Table 7 presents the mathematical-reasoning averages, computed over the same five benchmarks as the Math column of Table 1, together with the gradient conflict of the selected subset.

The main-text 100k comparison places `CoFlow` at 36.9 with conflict 0.29, above InstructDiff at 32.5 and Random Sampling at 26.4. The cone count grows from 14 to 38 at the same threshold τ, allowing the selection budget to cover a broader set of update directions. Table 7 retains additional controls alongside these three main-text rows.

Table 7: NuminaMath results on the 10k and 100k candidate pools. The Random Sampling, InstructDiff, and `CoFlow` 100k rows follow the main-text scale analysis. Math @10k is given where a corresponding row appears in Table 1.

Method	Math @10k	Math @100k	Conflict @100k
Full Training	27.05	29.83	0.64
Random Sampling	25.80	26.4	0.60
InstructDiff	31.50	32.5	0.47
IFD	21.99	24.18	0.57
Superfiltering	25.31	28.05	0.55
$\text{Entropy}_{\text{Min}}$	29.80	31.42	0.56
Nuggets	28.92	31.05	0.53
LESS	31.63	33.41	0.49
MATES	30.74	32.68	0.51
`CoFlow`	**35.08**	**36.9**	**0.29**

C.1 Complexity and Memory in Pool Size

Let N be the pool size, M the number of cones, r the sketch dimension, and k the budget. To compare pools of different sizes, we report the cost of each stage in terms of the operations it performs, which scale linearly with the number of retained examples. Stage 1 costs one forward pass for the loss, then S adapter-view backward passes and one projection per retained example, which is linear in N with a per-example cost of $O(S\,p + pr)$, where p is the size of the probing view. Stage 2 assigns each example to a cone in $O(Mr)$ and refreshes centers in a single pass, so assignment costs $O(NMr)$ in total. Medoid selection costs $O(k_m^2 r)$ inside cone m, bounded by $O(k^2 r)$ overall, and the scheduler works on the k/b batches of the selected subset at a cost of $O((k/b)^2 r)$. For a fixed budget k, all elements of this pipeline have compute cost linear in the pool size, and no stage is quadratic in N. The quantity M itself grows sub-linearly with the pool, which is why the measured selection stage in Table 8 stays below the linear prediction.

The signature matrix is the only object whose size depends on N in a way a practitioner has to plan for. At $r = 1024$ and single precision each retained example costs 4 KB, so a 10k pool stores at most 41 MB, a 100k pool at most 410 MB, and a one million example pool about 4.1 GB on disk, halved again where the sketches are kept in half precision. Table 8 records the cone counts, the selected budgets, the signature storage, and the measured selection time. Probing runs at 43.7 GB peak device memory per GPU with the adapter view of Qwen2.5-7B, and the same code path at 100k examples runs at 44.2 GB, because the device footprint is set by the model and the batch while the pool enters through the streaming loader. Cone assignment at 100k needs 5.9 GB of host memory for the sketch matrix and its Gram workspace.

Table 8: How the selection stage grows with the pool on NuminaMath. Selected is 4.1% of the pool, and Storage is the upper bound for the r=1024 signature matrix over the whole pool in single precision. Selection time is measured on the same 8-GPU node.

Pool N	Cones M	Selected (4.1%)	Storage	Selection
10,000	14	410	41 MB	8 min
25,000	21	1,025	102 MB	9 min
50,000	29	2,050	205 MB	12 min
100,000	38	4,100	410 MB	18 min

Because the sketches depend on the probe checkpoints and on nothing else, one probing pass serves every downstream choice of budget, cone threshold, and rotation threshold. The sweeps in App. E and the four domains of Table 11 reuse a single cached signature file per pool, which is why the scoring column of Table 5 is a one-time cost for a team that selects at several budgets.

D LARGER MODEL SCALE: QWEN2.5-14B

Here we repeat the mathematical-reasoning experiment of Sec. 5.3 on Qwen2.5-14B. We keep the NuminaMath 10k pool and the same five evaluation benchmarks. Table 9 reports the model-level averages aligned with the scale check in the main text, together with the corresponding 7B values where they are available.

Table 9: Mathematical-reasoning scale check on Qwen2.5-14B. The Full Training, InstructDiff, and `CoFlow` values reproduce the main-text scale analysis. The 7B values are included where Table 1 reports a corresponding row.

Method	**Math @7B**	**Math @14B**	Δ
Full Training	27.05	34.2	$+7.15$
InstructDiff	31.50	37.6	$+6.10$
`CoFlow`	**35.08**	**41.3**	**+6.22**

The 14B check places `CoFlow` at 41.3, InstructDiff at 37.6, and Full Training at 34.2, matching the main-text comparison. The projected signature remains at $r = 1024$ as the backbone changes. Table 10 records the stage costs for this larger backbone.

Table 10: Wall-clock hours on the 14B mathematical-reasoning run, same node and same four stages as Table 5. The Data column carries the same meaning: 4.1% for `CoFlow` and 10% for LESS.

Method	**Data**	**Prep.**	**Scoring**	**Select.**	**Fine-tune**	**Total**
Full Training	100%	1.0	0.0	0.0	9.2	10.2
LESS	10%	1.0	4.8	0.2	0.9	6.9
`CoFlow`	**4.1%**	1.0	3.0	0.1	0.4	**4.5**

E HYPERPARAMETER AND SENSITIVITY DETAIL

Figure 4 of the main text shows the four sensitivity panels. This section reports the Math-domain sweeps that complement those panels and extends them to the pool sizes and the scheduling threshold that the figure leaves out. As shown in Table 17, most hyperparameters are held the same across the four domains except the discovered cone count M, where a larger M reflects a more heterogeneous pool. All rows use the NuminaMath 10k pool with Qwen2.5-7B unless the caption says otherwise, and the Math and Avg columns use the same benchmark averaging as Table 1.

E.1 LEARNABLE-BAND RATIO

To demonstrate the sensitivity of `CoFlow` to the band setting, we vary ρ over $\{0, 0.05, 0.1, 0.2, 0.3\}$ for all four domains. Table 11 reports the result. The $\rho = 0$ row removes the filter entirely and reproduces the w/o Learnable-band Filter variant of Table 4, and the $\rho = 0.10$ row reproduces the default configuration of Table 1. Performance stays within 0.4 points of the default for every value in $\{0.05, 0.1, 0.2\}$, and the large drop appears only when the filter is switched off. At $\rho = 0.3$, the band over-filters the pool and costs 0.73 points.

Table 12 repeats the sweep on the 100k NuminaMath pool. The band behaves as the quantile definition of Equation (8) predicts: at 100k the cutoff $q_{1-\rho}$ sits at 4.63 nats per token against 4.15 at 10k, so the same $\rho = 0.1$ removes a lossier tail by a wider margin, and the accuracy curve keeps its shape and its optimum. The Retained column counts the examples that fall between the two cutoffs, which is $(1 - 2\rho)N$ for the two-sided band.

E.2 SKETCH DIMENSION

Table 13 sweeps r on the NuminaMath pool. We set the sketch dimension $r \in \{128, 256, 512, 1024, 2048\}$ and keep every other setting at its default. The score rises steeply

Table 11: Learnable-band ratio ρ across the four domains on the 10k pools. Avg is the mean of the four domain columns.

ρ	Math	General	Medical	Code	Avg	Conflict
0.00	33.12	7.86	57.44	37.79	34.05	0.37
0.05	34.71	8.31	60.42	40.11	35.89	0.32
0.10	**35.08**	8.54	**60.91**	**40.45**	**36.25**	**0.30**
0.20	34.96	**8.61**	60.55	40.22	36.09	**0.30**
0.30	34.41	8.22	59.86	39.60	35.52	0.32

Table 12: Learnable-band ratio ρ on the 100k NuminaMath pool. The last row switches the filter off.

ρ	Math @100k	Loss cutoff $q_{1-\rho}$	Retained
0.05	36.4	5.12	90,000
0.10	**36.9**	4.63	80,000
0.20	36.6	4.28	60,000
0.30	36.42	3.55	40,000
0.00 (off)	34.62	n/a	100,000

up to 1024 and gains 0.13 points from 1024 to 2048, while the probe time grows by 0.8 hours and the signature matrix doubles, which is the trade-off the main text refers to when it fixes $r = 1024$. The $r = 1024$ row is the default configuration of Table 1, so its Conflict value is the 0.30 reported there.

E.3 Rotation Threshold in Smooth Scheduling

Table 14 sweeps the rotation threshold λ of Sec. A.3. The achieved rotation tracks the threshold, and the score peaks at the default 0.40, where the scheduler still finds a compatible successor for most batches. The last row removes the ordering constraint and reproduces the w/o Smooth Scheduling variant of Table 4 in the Math domain, so its rotation is the 0.54 printed there.

E.4 Cone Threshold and Selection Budget

Table 15 varies τ, which changes how many cones the pool supports. Small τ merges incompatible skills into one flow and raises conflict, and large τ splits the pool into fragments whose medoids repeat one another, so diversity falls along with the score. The default 0.10 gives the 14 cones reported in Sec. A.3.

Table 16 reports the selection ratio. The 4.1% and 10% rows answer the budget question directly: `CoFlow` at 10% reaches 35.36 in Math, and the 4.1% default sits 0.28 points below it, so the advantage in Table 1 comes from the geometry of the subset and persists at the budget the baselines use. The 4.1% row is again the default configuration, and its Fine-tune column reproduces the corresponding cell of Table 5.

F Implementation Details

Models and pools. We use Qwen2.5-7B for mathematical reasoning, medical QA, and code generation, and LLaMA3-8B for general instruction following, the two backbones named in Sec. 5.1 and listed in the Backbone row of Table 17. Each candidate pool comprises 10k examples drawn from NuminaMath (LI et al., 2024), Alpaca (Li et al., 2023b), MedMCQA (Pal et al., 2022), and the BigCode instruction split (Cassano et al., 2024). We use the entire candidate pool of 10,000 examples in each domain, and the 100k NuminaMath pool of App. C extends the same construction to 100,000 examples. Pools are built with a fixed shuffle seed of 13 followed by exact-duplicate removal on the instruction string.

Training. We use the hyperparameters reported in Table 17. Hyperparameters are as follows: three epochs per run on the selected subset, a learning rate of 2×10^{-5} with cosine decay and warmup

Table 13: Sketch dimension r on NuminaMath with the 10k pool. Probing is the scoring column of Table 5, and storage is the signature matrix for one 10k pool in single precision.

r	Math	Probing (h)	Storage	Conflict
128	32.94	0.6	5.1 MB	0.36
256	33.86	0.8	10.2 MB	0.34
512	34.62	1.1	20.5 MB	0.32
1024	**35.08**	1.6	41.0 MB	0.30
2048	35.21	2.4	81.9 MB	0.30

Table 14: Rotation threshold λ on NuminaMath with the 10k pool. Rotation is the average of $\mathrm{Rot}(B_t, B_{t+1})$ over the schedule, Math is the domain average, and Segments counts the flow segments over the three epochs.

λ	Rotation	Math	Segments
0.25	0.33	34.36	9
0.30	0.35	34.78	7
0.40	0.39	**35.08**	4
0.50	0.44	34.91	3
0.65	0.50	34.52	2
unconstrained	0.54	34.26	1

over 3% of the steps, weight decay 0.0, AdamW with $\beta = (0.9, 0.999)$, gradient clipping at 1.0, and a maximum sequence length of 2048 tokens. All runs are trained in bf16 with full sharding. Furthermore, all methods within a domain are trained on the same shuffle of the candidate pool.

Hardware. For all experiments, we utilized 8 NVIDIA A100 80GB GPUs on one node. We train the 7B and 8B backbones on that node, and the Qwen2.5-14B runs of App. D use the same node with a wider sharding configuration. The estimated runtime required to complete one selection-and-fine-tuning run is detailed in Table 5. Consequently, a typical run at the 4.1% budget with Qwen2.5-7B takes approximately 2.6 hours, whereas training Qwen2.5-14B on the whole 10k pool exceeds 10 hours. Selection and scheduling run on CPU with a single process.

Seeds and variation. We report the mean over three seeds {13, 42, 2027}, and the standard deviations quoted below represent those three independent runs. The standard deviation across seeds is 0.22 for `CoFlow` on Math, 0.28 for LESS, and 0.19 for Full Training, so the 3.45-point Math gap between `CoFlow` and LESS of Table 1 exceeds two standard deviations by a wide margin.

F.1 Per-Benchmark Evaluation Protocol

Table 18 presents the complete protocol behind each evaluation column, based on the released benchmark templates. We use greedy decoding for all tasks except code generation, and the reported scores represent the mean of three independent runs. Mathematical reasoning uses exact-match accuracy after answer normalization, with the boxed answer extracted for Math-OAI (Hendrycks et al., 2021) and the final numeric span used for AIME24 (Zhang & Math-AI Team, 2024), Minerva (Lewkowycz et al., 2022), OlympiadBench (He et al., 2024), and AMC23 (math-ai, 2025). Medical QA scores the option letter. Code generation uses pass@1 over 20 samples with the standard unbiased estimator, and MultiPL-E covers the released language set. Instruction following uses length-controlled win rate (Dubois et al., 2024) with the AlpacaEval 2.0 protocol (Li et al., 2023b) and the Arena-Hard protocol with the LLaMA3-8B-Instruct answer as the reference. We use the entire AlpacaEval split of 805 items and the entire Arena-Hard split of 500 items.

Table 15: Cone threshold τ on NuminaMath with the 10k pool. M is the number of cones created by the assignment pass, and Diversity is the semantic diversity of the selected subset as reported in Table 3.

τ	M	**Math**	**Conflict**	**Diversity**
0.00	9	33.58	0.38	0.83
0.05	11	34.41	0.34	0.81
0.10	**14**	**35.08**	0.30	0.78
0.15	19	34.86	0.29	0.71
0.20	27	34.32	0.28	0.63

Table 16: Selection ratio on NuminaMath with the 10k pool. The 4.1% row is the default configuration.

Data	**Selected**	**Math**	**Conflict**	**Fine-tune (h)**
1.0%	100	33.42	0.29	0.06
2.0%	200	34.51	0.30	0.11
4.1%	410	**35.08**	0.30	0.4
10.0%	1,000	35.36	0.32	0.5
20.0%	2,000	35.41	0.33	0.9

G Qualitative Flow-Signature Examples

Table 19 provides illustrative examples of how the position of an example inside the loss distribution can lead to opposite selection decisions on two pools. The table lists seven NuminaMath examples from cone C_3, which holds 412 members at the 10k pool, together with the three quantities the pipeline reads: the length-normalized loss ℓ_i, the alignment to the best cone center $u_i^\top c_m$, and the smallest angular distance to the examples already picked in that cone. The band boundaries for this pool are $q_{0.1} = 0.62$ and $q_{0.9} = 4.15$.

As the examples demonstrate, R1 and R2 are the two failure modes the filter is designed for, and they sit at opposite ends of the loss distribution. R1 lies far above $q_{0.9}$ and is close to unlearnable at this stage, while R2 lies below $q_{0.1}$, is already internalized, and contributes an almost empty update. Both would survive an absolute loss threshold chosen on a different pool, and both are removed here by their position inside the quantiles of this pool. R3 and R4 clear the band at 1.97 and 2.62 nats and carry alignments of -0.21 and -0.34 to the best cone center, so the cone rule of Sec. A.3 is what removes them.

G.1 What Cone-Compatible Selection Removes that Influence Ranking Keeps

To better understand when cone-compatible selection helps, we examine examples selected by an influence-ranked control at the 10% budget of Table 1. Of its 1,000 examples, 118, or 11.8%, oppose their closest cone center with $u_i^\top c_m < 0$. Seventy-four of those 118 appear in the top influence decile. Their mean influence score is 0.41, compared with 0.22 for the other selected examples, illustrating why individual scoring can preserve directions that are difficult to combine in a minibatch. The aggregate conflict is 0.46 for the influence-ranked LESS row in Table 1 and 0.30 for `CoFlow`. The case study illustrates this mechanism at the level of individual examples.

G.2 Matched-Score Compatibility Control

We use the independent gradient-influence score to form a common candidate band and select 410 examples per domain from that band. Low-, medium-, and high-coherence subsets are assembled from the same candidate band while matching mean individual score and semantic diversity within 0.02. Coherence is measured by $\mathrm{Coh}(A)$, and every subset uses the same backbone, scheduling rule, training hyperparameters, and evaluation protocol. This isolates the geometry of examples trained together from the size and independent-score range of the selected set.

Table 17: Hyperparameters used during training and selection in our experiments. Values in the probing block implement Algorithm 1. M is the number of cones the assignment pass creates on each pool.

Setting	Math	General	Medical	Code
Backbone	Qwen2.5-7B	LLaMA3-8B	Qwen2.5-7B	Qwen2.5-7B
Pool size N	10,000	10,000	10,000	10,000
Budget k	410	410	410	410
Band ratio ρ	0.1	0.1	0.1	0.1
Probe checkpoints S	3	3	3	3
Sketch dimension r	1024	1024	1024	1024
Cone threshold τ	0.10	0.10	0.10	0.10
Rotation threshold λ	0.40	0.40	0.40	0.40
Cones M	14	12	15	13
Minibatch size	128	128	128	128
Epochs / learning rate	3 / 2×10^{-5}	3 / 2×10^{-5}	3 / 2×10^{-5}	3 / 2×10^{-5}
Seeds	13, 42, 2027	13, 42, 2027	13, 42, 2027	13, 42, 2027

Table 18: Evaluation protocol per benchmark. Items is the number of test problems, and shots is the number of demonstrations in the prompt.

Benchmark	Items	Metric	Shots	Decoding
AIME24	30	accuracy	0	greedy
AMC23	40	accuracy	0	greedy
Math-OAI	500	accuracy	0	greedy
Minerva	272	accuracy	4	greedy
OlympiadBench	674	accuracy	4	greedy
AlpacaEval 2.0	805	LC win rate	0	greedy
Arena-Hard	500	win rate	0	greedy
MedMCQA	4,002	accuracy	5	greedy
MedQA-US	1,273	accuracy	5	greedy
HumanEval	164	pass@1	0	0.2, 20 samples
MultiPL-E	1,638	pass@1	0	0.2, 20 samples

H Additional Notes on Scope and Future Work

The empirical evaluations cover candidate pools up to 100k examples and backbones up to 14B parameters. Table 5 records the 10k cost breakdown, while Table 10 gives the 14B measurement. The cone count moves from 14 to 38 between the two NuminaMath pool sizes in Table 8. Larger backbones and more heterogeneous pools may change both the probe cost and the cone geometry. Adaptive cone formation and token-level analysis of ℓ_i remain relevant extensions for those settings.

I Limitations and Scope of the Criterion

As discussed in Sec. 5.1, our study covers supervised instruction tuning of 7B and 14B models from candidate pools of 10k and 100k examples, where the flow signature of an example is a projected adapter-view gradient. On-policy synthetic generation, in which the pool is redrawn every round, and self-improvement loops, in which the pool is the model's own output, produce different flow geometries that may influence the cone structure and the measured conflict. Investigating these sources is a direction for future work. The method uses a fixed geometric criterion for cone formation and probing whose cost grows with model size. The measurements in App. C and App. D define the evaluated regime. Extending the flow-signature view to preference optimization, continual learning, and multimodal fine-tuning is another direction.

Table 19: Selected and rejected NuminaMath examples with their flow signatures. ℓ_i is in nats per token, alignment is $u_i^\top c_m$ for the best cone, and distance is $1 - u_i^\top u_j$ to the closest already selected example in that cone.

Example	Content	ℓ_i	Alignment	Distance	Decision
S1	two-step linear word problem	1.84	0.71	0.42	selected, medoid rank 1
S2	geometry proof with auxiliary line	2.31	0.64	0.38	selected, medoid rank 2
S3	modular arithmetic with cases	1.42	0.58	0.31	selected, medoid rank 4
R1	contest problem, mis-stated answer	6.83	0.44	0.51	removed by band (above $q_{0.9}$)
R2	single-step arithmetic restatement	0.31	0.66	0.29	removed by band (below $q_{0.1}$)
R3	calculus problem, sign error in solution	1.97	-0.21	0.62	removed by cone rule
R4	number theory, conflicting solution path	2.62	-0.34	0.57	removed by cone rule