

# RAISE: Reinforced Adaptive Instruction Selection For Large Language Models

Anonymous ACL submission

## Abstract

In the instruction fine-tuning of large language models (LLMs), it is widely recognized that a few high-quality instructions are superior to a large number of low-quality instructions. At present, many instruction selection methods have been proposed, but most of these methods select instruction based on heuristic quality metrics, and only consider data selection before training. These designs lead to insufficient optimization of instruction fine-tuning, and fixed heuristic indicators are often difficult to optimize for specific tasks. Therefore, we design a dynamic, task-objective-driven instruction selection framework RAISE(Reinforced Adaptive Instruction SElection), which incorporates the entire instruction fine-tuning process into optimization, selecting instructions at each step based on the expected impact of each instruction on model performance improvement. Our approach is well interpretable and has strong task-specific optimization capabilities. By modeling dynamic instruction selection as a sequential decision-making process, we use RL to train our selection strategy. Extensive experiments and result analysis prove the superiority of our method compared with other instruction selection methods. Notably, RAISE achieves superior performance by updating only 1% of the training steps compared to full-data training, demonstrating its efficiency and effectiveness.

## 1 Introduction

Large Language Models (LLMs) have achieved remarkable progress in recent years, demonstrating exceptional capabilities in general language understanding (Liu et al., 2023; Chen et al., 2024b) and generation (OpenAI, 2023; Achiam et al., 2023; Liu et al., 2024b; Sun et al., 2024). A critical factor enabling these advancements is instruction fine-tuning (Wei et al., 2021; Chung et al., 2024; Longpre et al., 2023), a process that aligns pretrained models with human intentions by training them on

task-specific instructions. While existing efforts predominantly focus on scaling instruction datasets (Khashabi et al., 2020; Ye et al., 2021; Wang et al., 2022) to improve model performance, recent studies highlight that data quality often outweighs sheer quantity (Zhou et al., 2024). This underscores the need for principled methods to identify instruction subsets that maximally enhance model capabilities.

Current instruction selection approaches typically rely on heuristic quality metrics (eg. grammatical correctness, clarity, lexical diversity, etc.) to filter low-quality instructions before training (Cao et al., 2023; Li et al., 2023; Chen et al., 2023; Xia et al., 2024; Pan et al., 2024). These methods face three main issues: (i) They use a one-time static selection before training, which does not adapt to a model’s evolving data needs during training; (ii) Their heuristic metrics are prone to cognitive bias and oversimplify the continuous nature of data quality; (iii) They are task-agnostic, failing to align instruction selection with specific task objectives.

Considering a dynamic, task-aware approach to instruction selection, we introduce the concept of an instruction’s **dynamic value**—its impact on the final model performance when used for gradient updates at time step  $t$  (we put detailed descriptions of dynamic value and example in Appendix C). This dynamic value, which depends on both the training step and the task objective, serves as a quality measure that replaces fixed heuristic metrics and provides strong interpretability. Dynamic instruction selection can be modeled as a sequential decision-making process aiming to maximize the model’s performance after  $T$  steps. Obviously, the optimal selection strategy is to select those instructions that have the most dynamic value at each step.

Based on this idea, we propose **RAISE** (Reinforced Adaptive Instruction SElection), a dynamic, non-heuristic, task-driven instruction selection framework. At its core is an **acquisition func-**

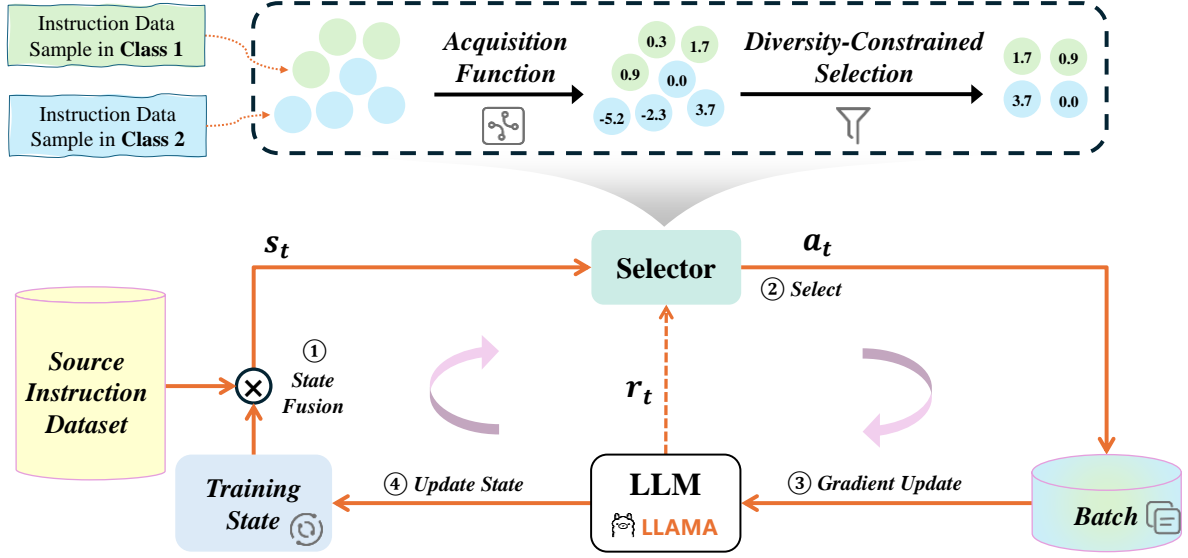


Figure 1: Overview of the RAISE framework, illustrating the training process of LLM at step  $t$ : (1) The **source instruction dataset** and **current training state** are fused to construct  $s_t$ , which encodes both data features and training progress. (2) The **Selector**, guided by the **acquisition function** (a trainable scorer), takes  $s_t$  as input and selects a batch of instruction data. (3) This selected batch is used to update LLM, resulting in performance improvement  $r_t$ . (4) Finally, the updated training state forms  $s_{t+1}$ , serving as input for the next step.

**tion**—a trainable MLP (sample-wise scorer) that estimates the **dynamic value** of each instruction and is optimized to maximize the final model performance. By leveraging a fully trained acquisition function to guide instruction selection, RAISE consistently produces high-performing models. Moreover, its task-driven design allows the acquisition function to be flexibly adapted to various tasks through adjustments in the validation set and performance metrics. Due to the sequential decision-making nature of RAISE, we employ reinforcement learning (RL) (Bellman, 1966; Mnih et al., 2015) to optimize the acquisition function, treating each training process of LLM as an episode in the RL setting.

To promote diversity, RAISE groups instructions into multiple classes via K-means (MacQueen et al., 1967) and ensures balanced sampling from each group in every training batch. These classes constitute diversity constraint. Figure 1 shows the framework for RAISE, which considers both the score of the acquisition function and the diversity constraint when selecting instructions.

In summary, our contributions are as follows:

- We design a **task-objective-driven acquisition function** to estimate the dynamic value of each instruction based on its expected impact on the final model performance, eliminating the need for heuristic quality metrics.

- We propose **RAISE**, a dynamic instruction selection framework that adaptively selects instructions during training based on their dynamic value, enabling the model to meet changing data requirements during training.
- Through extensive experiments, we demonstrate the effectiveness of our approach and provide analysis highlighting the potential of dynamic instruction selection for future advancements in instruction fine-tuning.

## 2 Related Work

### 2.1 Instruction Selection

Instruction selection focuses on identifying a subset of instructions from a fine-tuning dataset that maximize model performance, rather than training on the entire dataset. Recent studies emphasize that carefully selected subsets can often outperform full-dataset training (Cao et al., 2023; Li et al., 2023; Xia et al., 2024), underscoring the importance of effective selection strategies. The shift from dataset scaling to quality-focused selection highlights the need for principled methods to prioritize high-utility instructions.

Many methods have been proposed for instruction selection. **IFD** (Li et al., 2024) introduces an Instruction Following Difficulty metric to assess instruction complexity and select appropriate sam-

ples. **AlpaGasus** (Chen et al., 2024a) uses GPT-4 (Achiam et al., 2023) to score instruction-response pairs, filtering low-quality samples and improving training efficiency. **DEITA** (Liu et al., 2024a) combines complexity and quality scores to optimize instruction selection, balancing diversity and data quality.

However, these approaches share a common limitation: **static selection**. Once the subset is chosen, it remains fixed throughout training, failing to adapt to the model’s evolving data preferences. In contrast, our proposed RAISE framework dynamically selects instructions at each training step based on their dynamic value, enabling adaptive learning that aligns with the model’s changing needs.

## 2.2 Self-Paced Learning

Self-Paced Learning (SPL) (Kumar et al., 2010) represents a prominent curriculum learning paradigm (Bengio et al., 2009) that dynamically selects training samples based on their difficulty levels. Unlike static curriculum approaches, SPL employs an adaptive weighting mechanism where easier samples are prioritized in early training stages while harder ones are progressively incorporated. This dynamic selection is governed by a self-paced regularizer that balances sample inclusion with a pacing parameter  $\lambda$  controlling curriculum progression.

The core mechanism involves jointly optimizing model parameters and sample selection through a bi-level objective: while the model learns to minimize task loss, the sample selector determines optimal inclusion thresholds based on current loss values. This loss-driven thresholding strategy has proven effective in improving convergence robustness across various domains (Wang et al., 2021).

However, this loss-driven approach introduces critical limitations for instruction tuning: (i) Loss values often poorly reflect task-specific metrics (e.g., accuracy, BLEU); (ii) The rigid easy-to-hard progression may discard valuable hard samples; (iii) Its single optimization objective cannot adapt to diverse task requirements. RAISE addresses these issues by replacing loss with task-aware dynamic value estimation and introducing diversity constraint through clustered sampling, enabling both task-aware selection and adaptive learning.

## 3 Method

In this section, we describe our method for dynamic instruction selection. A learnable acquisition function is trained to estimate the dynamic value of each instruction, ensuring adaptive and diversity-aware selection throughout the training process.

We formally define the problem of dynamic instruction selection (§ 3.1), introduce the training framework of our selection policy (§ 3.2), and describe the state fusion mechanism that combines training state and data features (§ 3.3). We then present the instruction selection algorithm (§ 3.4) and the policy optimization algorithm for improving the selection policy (§ 3.5).

### 3.1 Problem Statements

Given an instruction dataset  $\mathcal{D} = \{d_i\}_{i=1}^N$ , our goal is to dynamically select a subset  $\mathcal{D}_t$  at each training step  $t$  to maximize the performance  $\mathcal{P}$  of the model updated at final step  $T$ .  $\mathcal{P}$  is defined as a performance metric related to downstream tasks, which includes a validation set  $\mathcal{D}_{\text{val}}$  visible during the training process and an evaluation metric (e.g., -loss/accuracy/rouge on  $\mathcal{D}_{\text{val}}$ ). The optimal selection policy  $\pi^*$  can be formulated as:

$$\pi^* = \arg \max_{\pi} \mathcal{P}(\mathcal{M}_T[\mathcal{D}, \pi], \mathcal{D}_{\text{val}}), \quad (1)$$

where,  $\mathcal{M}_T[\mathcal{D}, \pi]$  represents the model updated at step  $T$ . For simplicity, in the following content, we denote  $\mathcal{P}(\cdot, \mathcal{D}_{\text{val}})$  as  $\mathcal{P}(\cdot)$  and  $\mathcal{M}_t[\mathcal{D}, \pi]$  as  $\mathcal{M}_t$ .

### 3.2 Training Framework of Selection Policy

Dynamic instruction selection can be formulated as a sequential decision-making process. Specifically, at each training step  $t$ , the selection policy  $\pi$  determines a subset  $\mathcal{D}_t$  from the dataset  $\mathcal{D}$  to update the model  $\mathcal{M}_{t-1}$ . This process can be modeled as a Markov Decision Process (MDP) (Bellman, 1966; Puterman, 2014) consisting of:

- **State** ( $S_t$ ): The state at step  $t$ , represents all available information, building from the current training state and  $\mathcal{D}$  by **State Fusion**.
- **Action** ( $A_t$ ): The action is the selected batch data  $\mathcal{D}_t$  from  $\mathcal{D}$  according to the policy  $\pi$ , i.e.,  $A_t = \mathcal{D}_t = \pi(S_t)$ .
- **Reward** ( $R_t$ ): The reward is based on the performance improvement after using  $\mathcal{D}_t$  to update the model, i.e.,  $R_t = \mathcal{P}(\mathcal{M}_t) - \mathcal{P}(\mathcal{M}_{t-1})$ .

Once the subset  $\mathcal{D}_t$  is selected, it is used to update the model  $\mathcal{M}_{t-1}$ , resulting in the new model  $\mathcal{M}_t$  and an updated state  $S_{t+1}$ . The goal of training is to maximize the cumulative reward, which reflects the final model performance  $\mathcal{P}(\mathcal{M}_T)$ .

In this framework, the selection policy  $\pi$  consists of a learnable acquisition function  $\mathcal{F}$  and a diversity constraint  $\mathcal{C}$ . Only  $\mathcal{F}$  is trainable, so optimizing  $\pi$  is equivalent to optimizing  $\mathcal{F}$ .

### 3.3 State Fusion

In dynamic instruction selection, **State Fusion** combines the current training state with original instruction features to form a comprehensive representation for the acquisition function. Specifically, we denote the fused state as  $d' = \mathcal{H}(d, \mathcal{M}_{t-1}, t)$ , where  $d$  is the instruction sample and  $\mathcal{H}$  is the fusion function. The fusion of state involves 4 components:

- **Stage State** ( $\mathcal{H}_{\text{stage}}$ ): This component captures the model's current training progress, including  $\mathcal{M}_{t-1}$  and  $t$ . Formally:

$$\mathcal{H}_{\text{stage}}(\mathcal{M}_{t-1}, t) = \left[ \mathcal{P}_{t-1}, \frac{t}{T} \right] \quad (2)$$

- **Instruction-Difficulty State** ( $\mathcal{H}_{\text{diff}}$ ): To represent the complexity of each instruction, we collect  $\log P(y|x)$ ,  $\log P(y)$ , and the lengths of the prompt and its response. To ensure efficiency, they are precomputed using the auxiliary model<sup>1</sup>. Formally:

$$\mathcal{H}_{\text{diff}}(d) = [\text{len}(x), \text{len}(y), \log P(y|x), \log P(y)] \quad (3)$$

- **Instruction-Semantic State** ( $\mathcal{H}_{\text{sem}}$ ): This component encodes the semantic information of the instruction. We compute the embedding vector  $E(d)$  with the auxiliary model, followed by a pooling layer:

$$\mathcal{H}_{\text{sem}}(d) = [\text{Pool}(E(d))] \quad (4)$$

- **Instruction-Availability State** ( $\mathcal{H}_{\text{avail}}$ ): We record the number of times  $\nu(d)$  an instruction has already been selected during training, helping the acquisition function avoid excessive repetition of the same instruction:

$$\mathcal{H}_{\text{avail}}(d) = [\nu(d)] \quad (5)$$

<sup>1</sup>We use Llama-3.1-8B-Instruct as the auxiliary model to preprocess instruction-difficulty state and instruction embeddings.

By concatenating these 4 components, we obtain the fused state:

$$\mathcal{H}(d, \mathcal{M}_{t-1}, t) = [\mathcal{H}_{\text{stage}}(\mathcal{M}_{t-1}, t), \mathcal{H}_{\text{diff}}(d), \mathcal{H}_{\text{sem}}(d), \mathcal{H}_{\text{avail}}(d)] \quad (6)$$

### 3.4 Instruction Selection Algorithm

#### Algorithm 1 Dynamic Instruction Selection with Diversity Constraint

---

```

1: Input: Training dataset  $\mathcal{D}$ , LLM  $\mathcal{M}_{t-1}$ , Batch size  $B$ ,
   Acquisition function  $\mathcal{F}$ , Diversity constraint (classes)  $\mathcal{C}$ 
   and Fusion function  $\mathcal{H}$ 
2: Output: Selected subset of  $B$  samples
3:  $C \leftarrow |\mathcal{C}|$ ,  $b \leftarrow \frac{B}{C}$ 
4: Initialize  $S_t \leftarrow \emptyset$ ,  $s \leftarrow \emptyset$ 
5: for  $d_j \in \mathcal{D}$  do
6:    $d'_j \leftarrow \mathcal{H}(d_j, \mathcal{M}_{t-1}, t)$ 
7:    $S_t \leftarrow S_t \cup \{d'_j\}$ 
8:    $s_j \leftarrow \mathcal{F}(d'_j)$  ▷ Dynamic value of  $d_j$ 
9: end for
10: for  $\mathcal{C}_i \in \mathcal{C}$  do ▷ Divide  $S_t$  into  $C$  classes
11:    $S_{t,i} \leftarrow \emptyset$ 
12:   for  $d_j \in \mathcal{C}_i$  do
13:      $S_{t,i} \leftarrow S_{t,i} \cup d'_j$ 
14:   end for
15: end for
16: for  $i = 1, 2, \dots, C$  do
17:    $\pi(S_{t,i}) \leftarrow \arg \text{top}_b \{s_j \mid d'_j \in S_{t,i}\}$ 
18: end for
19:  $\pi(S_t) \leftarrow \bigcup_{i=1}^C \pi(S_{t,i})$ 
20: return  $\pi(S_t)$ 

```

---

Algorithm 1 presents the instruction selection process with diversity constraint at training step  $t$ . We first apply the fusion function  $\mathcal{H}$  to incorporate training state into each instruction  $d_j$ . The acquisition function  $\mathcal{F}$  then scores the fused instructions, and a diversity constraint  $\mathcal{C} = \{\mathcal{C}_1, \dots, \mathcal{C}_C\}$  (each  $\mathcal{C}_i$  represents a class) ensures balanced coverage of heterogeneous instruction types. Specifically, we select the top- $b$  instructions (based on  $\mathcal{F}$ ) from each class, and their union forms the final training subset  $\mathcal{D}_t$ . This selected batch is then used to update LLM, and the process repeats at the next training step.

### 3.5 Policy Optimization Algorithm

To train the selection policy  $\pi$ , we adopt PPO (Schulman et al., 2017), where the acquisition function  $\mathcal{F}_\theta$  acts as **Actor**, and  $V_\phi$  serves as **Critic**.

**Advantage Estimation.** To stabilize training and improve generalization, we employ Generalized Advantage Estimator (GAE) (Schulman et al., 2015) for advantage computation:

$$\delta_t = R_t + \gamma V_\phi(S_{t+1}) - V_\phi(S_t), \quad (7)$$



$$\text{Adv}_t = \sum_{l=0}^{T-t-1} (\gamma \lambda)^l \delta_{t+l}, \quad (8)$$

$$G_t = V_\phi(S_t) + \text{Adv}_t, \quad (9)$$

where  $\gamma$  is the discount factor and  $\lambda$  is the GAE parameter,  $\text{Adv}_t$  and  $G_t$  is advantage and return respectively.

**Importance Sampling with Diversity Constraint.** Under the diversity-constrained selection, the importance sampling ratio is computed on a per-class basis. Let  $\{\mathcal{C}_1, \dots, \mathcal{C}_C\}$  be the class used in instruction selection, and define:

$$\begin{aligned} p_{\text{new},i}(d'_j) &= \frac{\exp(\mathcal{F}_\theta(d'_j))}{\sum_{d'_k \in \mathcal{C}_i} \exp(\mathcal{F}_\theta(d'_k))}, \\ p_{\text{old},i}(d'_j) &= \frac{\exp(\mathcal{F}_{\theta_{\text{old}}}(d'_j))}{\sum_{d'_k \in \mathcal{C}_i} \exp(\mathcal{F}_{\theta_{\text{old}}}(d'_k))}, \end{aligned} \quad (10)$$

Then, the overall ratio for a selected batch is:

$$\hat{r}_t = \prod_{i=1}^C \prod_{d'_j \in \pi(S_{t,i})} \frac{p_{\text{new},i}(d'_j)}{p_{\text{old},i}(d'_j)}, \quad (11)$$

where  $\pi(S_{t,i})$  denotes the top- $b$  samples chosen from the  $i$ -th class at step  $t$ .

**Loss Functions.** Following PPO, we optimize both the actor and critic losses. The actor loss is given by:

$$\mathcal{L}^{\text{actor}} = -\mathbb{E}_t \left[ \min \left( \hat{r}_t \text{Adv}_t, \text{clip}(\hat{r}_t, 1 - \epsilon, 1 + \epsilon) \text{Adv}_t \right) \right], \quad (12)$$

where  $\epsilon$  is the clipping parameter ( $\epsilon = 0.2$ ). The critic loss is simply a mean-squared error:

$$\mathcal{L}^{\text{critic}} = \mathbb{E}_t \left[ (V_\phi(S_t) - G_t)^2 \right]. \quad (13)$$

**Training Framework.** We run  $K$  rounds of PPO training. In each round, the LLM is trained for  $T$  steps following the current policy  $\pi$ , with data (i.e., states, actions, rewards) being collected. We then use these collected samples to update the actor  $\mathcal{F}_\theta$  and the critic  $V_\phi$  via the aforementioned PPO objective. Iterating this process over  $K$  rounds gradually refines the acquisition function  $\mathcal{F}_\theta$ , ultimately yielding a strong policy for dynamic instruction selection. Detailed training process is in Appendix D.

## 4 Experiments

### 4.1 Experimental Setup

**Training Dataset.** We use Alpaca-52K (Taori et al., 2023) as our instruction fine-tuning dataset, which contains 52,000 multi-domain instruction-response pairs spanning tasks such as question answering, text generation, translation and so on<sup>2</sup>.

**Evaluation Datasets.** We evaluate on four benchmarks: MMLU, ARC (Challenge) (ARC-C), CommonsenseQA (ComQA), and GSM8K. See Appendix A.1 for detailed settings of them.

### 4.2 Baselines

We employ multiple baselines to compare with RAISE. The simplest one is **RAND**, which randomly samples a subset of instructions from the full training set. We report the average performance over 5 independent random samplings. We also compare against other established methods, such as **IFD**, **DEITA** and **AlpaGasus**. In addition, we design a dynamic selection variant for SPL, termed **SSPL** (see Appendix A.3).

To ensure a fair comparison between static and dynamic selection methods, we match the total number of update steps across all approaches. Concretely, for static methods, we first pick 1% of the full training set as a fixed subset and then train the model for 3 epochs. For dynamic methods, we set max\_steps to match the total number of update steps in the static setting, thereby enforcing an equivalent amount of training.

### 4.3 Main Results

We present the results of RAISE versus various baselines using different models in Tables 1, and we showcase RAISE’s capability for task-specific optimization in Figure 2. Our key findings are as follows:

**Only 1% of gradient-update steps suffices to surpass full-data training.** In Table 1, RAISE requires only 1% of the total number of update steps, yet outperforms models trained on the entire dataset on all models. Notably, RAISE achieves a significantly better result than this full-data baseline. We conjecture that only a small fraction of data truly benefits the task objective, while most of the dataset provides minimal gains. By explicitly optimizing toward the task objective, RAISE

<sup>2</sup>We use the Alpaca version from <https://huggingface.co/datasets/yahma/alpaca-cleaned>.

effectively captures these valuable data. In Qwen’s experimental results, using the entire dataset actually yielded worse results than using a subset of the data. To some extent, this further supports this point. We conducted a more detailed analysis of this phenomenon of "less is more" in Appendix G.

**RAISE consistently outperforms baselines on different models.** Tables 1 shows that RAISE achieves superior performance across all tested models. Although RAISE remains robust for both small and large model scales, its advantage over baselines is especially pronounced on stronger Llama-3.2-3B compared to smaller Llama-3.2-1B.

**RAISE exhibits strong capability of task-specific optimization.** In Figure 2, all baselines perform poorly in GSM8K due to their reliance on heuris-

tic and general “quality” metrics, which predominantly capture instruction difficulty rather than the actual task objective. Since only a small fraction of Alpaca’s instructions involve the target reasoning tasks, these baselines are largely ineffective. In contrast, RAISE explicitly identifies and prioritizes instructions that align with the final objective, as evidenced by its emphasis on computational and reasoning-focused prompts relevant to GSM8K (see Appendix I). Meanwhile, we conducted out-of-distribution evaluations on MathQA and MMLU.Math (MMLU.college\_mathematics), and RAISE still outperformed the baseline.

## 5 Analysis

In this section, we further investigate how RAISE selects instructions by examining two core mod-

Model	DATA	Avg.Q	Avg.	MMLU	ARC-C	ComQA
Llama-3.2-3B	0%	-100%	52.67	51.66	42.15	64.21
	100%	0%	54.32	52.76	43.77	<b>66.42</b>
	RAND	-7.33%	54.20	52.86	42.32	65.11
	IFD	+25.06%	54.73	52.66	46.42	65.11
	DEITA	-16.44%	54.05	51.90	44.88	65.36
	AlpaGasus	-72.14%	53.13	52.30	44.11	62.98
	SSPL	-196.75%	51.08	50.11	41.64	61.51
	<b>RAISE(Ours)</b>	<b>+70.35%</b>	<b>55.47</b>	<b>54.64</b>	<b>46.59</b>	65.19
Llama-3.2-1B	0%	-100%	38.33	35.53	33.76	45.71
	100%	0%	39.36	35.94	<b>36.86</b>	45.29
	RAND	+7.01%	39.44	35.91	34.81	<b>47.58</b>
	IFD	-15.77%	39.20	37.35	34.47	45.78
	DEITA	-65.79%	38.69	36.58	33.45	46.03
	AlpaGasus	-47.39%	38.88	36.89	33.87	45.86
	SSPL	+69.55%	40.08	37.20	36.60	46.44
	<b>RAISE(Ours)</b>	<b>+85.14%</b>	<b>40.24</b>	<b>38.14</b>	35.58	47.01
Qwen-2.5-3B	0%	-100%	62.16	63.19	47.13	76.15
	100%	0%	63.70	65.22	49.12	76.75
	RAND	+15.53%	63.94	65.30	50.18	76.33
	IFD	+34.04%	64.22	65.25	50.43	76.99
	DEITA	+28.37%	64.13	<b>65.43</b>	49.66	77.31
	AlpaGasus	+32.06%	64.19	65.18	49.91	77.48
	SSPL	+46.59%	64.42	65.33	50.68	77.23
	<b>RAISE(Ours)</b>	<b>+69.83%</b>	<b>64.77</b>	65.32	<b>51.28</b>	<b>77.72</b>

Table 1: **Performance comparison on MMLU, ARC-Challenge, and CommonsenseQA.** All methods are trained on Alpaca-52K. We report results for two versions of Llama-3.2 (3B and 1B) and Qwen-2.5-3B. “0%” denotes the base model and “100%” denotes full Alpaca dataset, and otherwise we select 1% of the data or equivalent number of training steps. “Avg” denotes the average metric across these three benchmarks. “Avg.Q” denotes the additional performance gain achieved by each method, relative to the improvement obtained by using the 100% data baseline. It is computed as:  $\text{Avg.Q}(\cdot) = \frac{\text{Avg}(\cdot) - \text{Avg}(0\%)}{\text{Avg}(100\%) - \text{Avg}(0\%)}$ . Bold numbers denotes the best performing on its column.

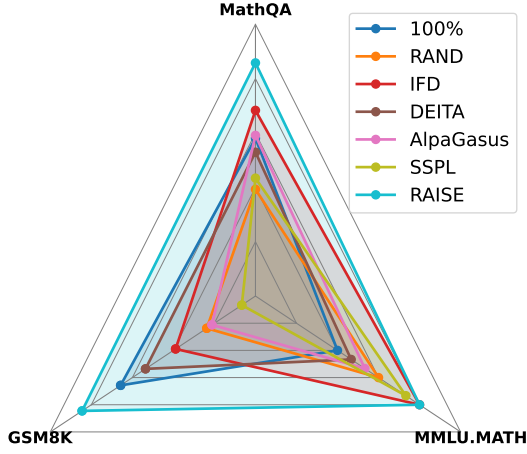


Figure 2: **Performance of GSM8K-targeted training.** All methods are still trained on Alpaca dataset but use GSM8K as validation set (different from the GSM8K evaluation set). Besides, We use MathQA and MMLU.Math (MMLU.college\_mathematics) for OOD evaluations.

ules: **state fusion** and **diversity-constrained selection**. Finally, we analyze the distribution of data selected by RAISE at different stages of training.

### 5.1 Ablation on State Fusion

**A small instruction semantic dimension suffices.** We vary the dimension of the semantic embedding ( $\{8, 16, 32, 64\}$ ; default: 32) and report the results in Figure 3. Even though 32 is much smaller than the original embedding size (e.g., 4096), it consistently yields solid performance on MMLU, ARC-C, and ComQA. Increasing the semantic dimension leads to modest gains overall, but notably,

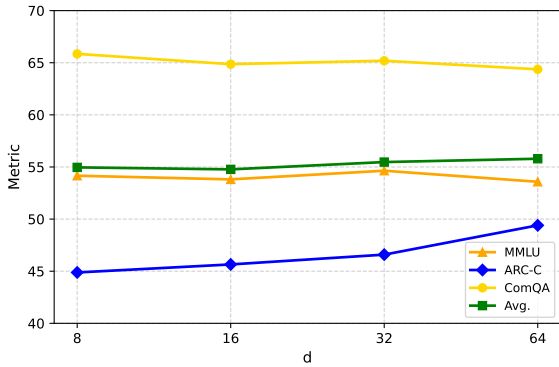


Figure 3: Performance with different instruction semantic dimensions  $d_{sem}$ .

ARC-C benefits the most from higher-dimensional representations, suggesting a stronger reliance on richer feature spaces for reasoning. Although performance on MMLU and ComQA slightly declines at 64 dimensions, the improvement on ARC-C compensates, keeping the overall average competitive.

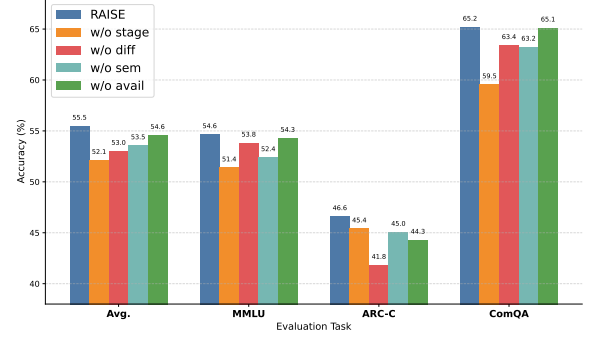


Figure 4: Ablation results on different components in state fusion.

**Stage State is the most critical.** Ablation results in Figure 4 show that removing the Stage State leads to the largest performance drop, confirming its role as the key temporal controller in RAISE. It dynamically guides training by integrating model status and progress, enabling early semantic exploration and late-stage sample refinement. Other states (difficulty, semantic, availability) also contribute—e.g., Difficulty State is essential for complex tasks like ARC-C, and Semantic State helps in knowledge-rich domains like MMLU—but their impact is more task-specific. Availability plays a minor role but ensures diversity with minimal cost. We discuss in detail in Appendix B the impact of each component on the performance of RAISE.

### 5.2 Ablation on Diversity-Constrained Selection

In the diversity-constrained selection, all data are first clustered into  $C$  classes via K-means, and the model then selects top-scoring samples within each class. We study how different values of  $C$  affect performance. As shown in Figure 5, we vary  $C \in \{1, 2, 4, 8, 16, 32\}$ . When  $C$  is small (1 or 2), the model achieves relatively strong overall performance when selecting 1% data, whereas larger  $C$  leads to a downward trend. While this might seem counterintuitive—given that diversity often boosts performance—the key factor here is that RAISE uses only 1% of the training steps compared to full-data training. Under such a tight budget, the

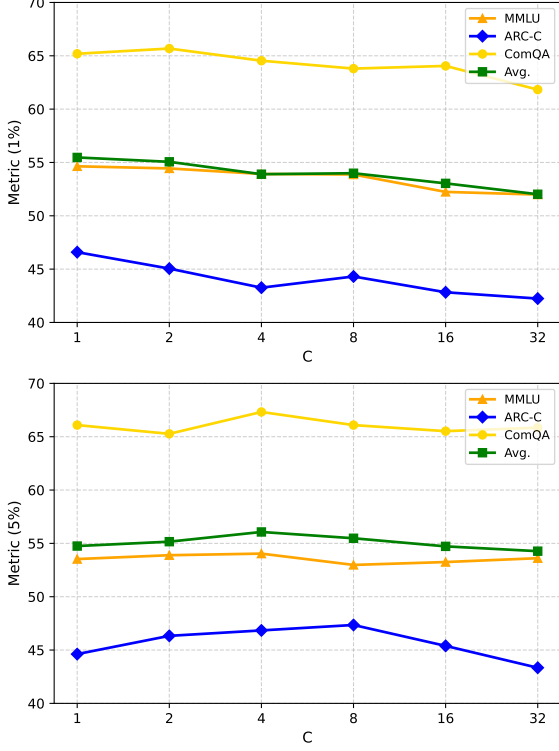


Figure 5: Performance with different class counts  $C$  when using only 1% and 5% data. The larger the  $C$ , the finer the class, and the fewer instructions each class selects.

model must rapidly focus on data most aligned with the target objective. These valuable samples may all fall into a single class, and the diversity constraint then limits how many can be selected from that cluster ( $B/C$ ), thereby hurting performance. The experiment on 5% data proves this (achieving best performance at  $C = 4$ ), suggesting that ensuring data diversity remains a beneficial measure for training when there is a large amount of data samples.

### 5.3 Distribution of Selected Instructions

In this section, we investigate the data selected by RAISE. We split the total  $T$  training steps into three stages (Stage 0, Stage 1, and Stage 2), representing the early, middle, and late stages of training. We then visualize the distribution of the selected samples at each stage. As shown in Figure 6, the data chosen in the early and middle stages are widely scattered, whereas in the final stage they become tightly clustered. This indicates that the most beneficial data for the model changes over time. In the early and middle stages of training, various patterns of data are helpful for the training process.

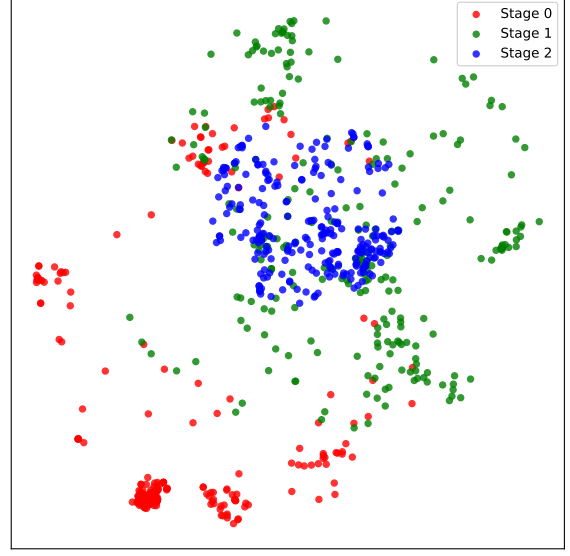


Figure 6: Distribution of selected instructions in different stage.

However, in the later stage, the data that is most beneficial for training is concentrated on certain patterns.

## 6 Conclusion

In this paper, we present **RAISE**, a dynamic instruction selection method that adaptively selects beneficial instructions for LLM fine-tuning. RAISE employs a task-objective-driven acquisition function and a cluster-based diversity mechanism to identify high-utility data. Our experiments on multiple benchmarks demonstrate that RAISE outperforms static selection baselines, achieving strong performance while using only a small fraction of training steps. We hope this work inspires further research on adaptive data selection and fine-tuning strategies for large language models.

### Limitation

RAISE incurs linear memory overhead in the replay buffer during RL training of the acquisition function. Specifically, when storing states for RL optimization, each instruction’s fused state vector (dimension  $M$ ) requires  $O(M)$  memory. For a dataset of size  $N$ , the total buffer storage scales as  $O(N \times M)$ . This becomes prohibitive for composite datasets where  $N \geq 200,000$ —common in current instruction tuning. Furthermore, when sampling batches from the buffer, multiple state vectors must be simultaneously loaded into memory, exacerbating peak memory pressure. Consequently,



RAISE faces scalability challenges for very large-scale instruction datasets, necessitating future work on state compression or distributed buffer strategies.

## References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Richard Bellman. 1966. Dynamic programming. *science*, 153(3731):34–37.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

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

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

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

Pei Chen, Soumajyoti Sarkar, Leonard Lausen, Balasubramaniam Srinivasan, Sheng Zha, Ruihong Huang, and George Karypis. 2024b. Hytrel: Hypergraph-enhanced tabular data representation learning. *Advances in Neural Information Processing Systems*, 36.

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

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

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

Dan Hendrycks, Collin Burns, Steven Basart, Andrew Critch, Jerry Li, Dawn Song, and Jacob Steinhardt. 2021a. Aligning ai with shared human values. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021b. Measuring massive multitask language understanding. *Proceedings of the International Conference on Learning Representations (ICLR)*.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hananeh Hajishirzi. 2020. Unifiedqa: Crossing format boundaries with a single qa system. *arXiv preprint arXiv:2005.00700*.

Diederik P Kingma. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

M Kumar, Benjamin Packer, and Daphne Koller. 2010. Self-paced learning for latent variable models. *Advances in neural information processing systems*, 23.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2023. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. *arXiv preprint arXiv:2308.12032*.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. 2024. [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)*, pages 7602–7635, Mexico City, Mexico. Association for Computational Linguistics.

Fuxiao Liu, Xiaoyang Wang, Wenlin Yao, Jianshu Chen, Kaiqiang Song, Sangwoo Cho, Yaser Yacoob, and Dong Yu. 2023. Mmc: Advancing multimodal chart understanding with large-scale instruction tuning. *arXiv preprint arXiv:2311.10774*.

Wei Liu, Weihao Zeng, Keqing He, Yong Jiang, and Junxian He. 2024a. [What makes good data for alignment? a comprehensive study of automatic data selection in instruction tuning](#). In *The Twelfth International Conference on Learning Representations*.

Xiaoyu Liu, Paiheng Xu, Junda Wu, Jiaxin Yuan, Yifan Yang, Yuhang Zhou, Fuxiao Liu, Tianrui Guan, Hao-liang Wang, Tong Yu, et al. 2024b. Large language models and causal inference in collaboration: A comprehensive survey. *arXiv preprint arXiv:2403.09606*.

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

- James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA.
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.
- Chatgpt OpenAI. 2023. Optimizing language models for dialogue, 2022. URL: <https://openai.com/blog/chatgpt>.
- Xingyuan Pan, Luyang Huang, Liyan Kang, Zhicheng Liu, Yu Lu, and Shanbo Cheng. 2024. G-dig: Towards gradient-based diverse and high-quality instruction data selection for machine translation. *arXiv preprint arXiv:2405.12915*.
- Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
- John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Wangtao Sun, Haotian Xu, Xuanqing Yu, Pei Chen, Shizhu He, Jun Zhao, and Kang Liu. 2024. Itd: Large language models can teach themselves induction through deduction. *arXiv preprint arXiv:2403.05789*.
- Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. **CommonsenseQA: A question answering challenge targeting commonsense knowledge**. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.
- Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. [https://github.com/tatsu-lab/stanford\\_alpaca](https://github.com/tatsu-lab/stanford_alpaca).
- Xin Wang, Yudong Chen, and Wenwu Zhu. 2021. A survey on curriculum learning. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):4555–4576.
- Yizhong Wang, Swaroop Mishra, Pegah Alipoor-molabashi, Yeganeh Kordi, Amirreza Mirzaei, Anjana Arunkumar, Arjun Ashok, Arut Selvan Dhanasekaran, Atharva Naik, David Stap, et al. 2022. Super-naturalinstructions: Generalization via declarative instructions on 1600+ nlp tasks. *arXiv preprint arXiv:2204.07705*.
- Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. 2021. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*.
- Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. 2024. LESS: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning (ICML)*.
- Qinyuan Ye, Bill Yuchen Lin, and Xiang Ren. 2021. Crossfit: A few-shot learning challenge for cross-task generalization in nlp. *arXiv preprint arXiv:2104.08835*.
- Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2024. Lima: Less is more for alignment. *Advances in Neural Information Processing Systems*, 36.

## A Details of Experiments Setup

### A.1 Evaluation Datasets

MMLU covers 57 tasks ranging from elementary math and U.S. history to computer science and law, primarily measuring knowledge breadth and reasoning (Hendrycks et al., 2021b,a). ARC-C is a challenging subset of the AI2 Reasoning Challenge, featuring multiple-choice questions that demand complex reasoning and scientific knowledge (Clark et al., 2018). ComQA is a common-sense reasoning benchmark requiring real-world knowledge and inference (Talmor et al., 2019). GSM8K contains 8,000 grade-school math problems focusing on multi-step numeric reasoning (Cobbe et al., 2021). Table 2 provides more detailed information about these evaluation datasets.

Dataset	$ \mathcal{D}_{\text{val}} $	$ \mathcal{D}_{\text{test}} $	Answer Type	Metric
MMLU	285	14,042	Letter options	Acc
ARC-C	299	1,172	Letter options	Acc
ComQA	280 <sup>3</sup>	1,140	Letter options	Acc
GSM8K	256 <sup>4</sup>	1,319	COT and answer	Acc

Table 2: Statistics of evaluation datasets.

### A.2 Hyperparameters

When training the LLM in each round, we set the learning rate to  $2\text{e-}5$ , use a cosine learning rate scheduler, and have no warm-up steps. When ppo training, we set actor learning rate, critic learning rate, weight decay,  $\gamma$ ,  $\lambda$ ,  $K$  respectively to  $1\text{e-}1$ ,  $2\text{e-}1$ ,  $1\text{e-}2$ ,  $0.99$ ,  $1.0$ ,  $20$ . In our **state fusion** pipeline, we pool the instruction embedding vector to a dimension of 32. Consequently, the fused state dimension is  $2$  (stage) +  $4$  (diff) +  $32$  (sem) +  $1$  (avail) =  $39$ . As a practical measure of model performance  $\mathcal{P}(\mathcal{M}_t)$ , we use  $-\text{Loss}(\mathcal{M}_t, \mathcal{D}_{\text{val}})$  for computational efficiency. When evaluating the trained LLM, we use the lm\_eval framework and set max\_length to 512.

### A.3 SSPL Baseline

In SSPL baseline, all training examples are sorted by their loss values and divided into max\_steps buckets of approximately equal size, such that each bucket contains instructions with similar difficulty (as measured by loss). During training, the model

<sup>3</sup>In ComQA, we randomly select 280 samples from the original 1,221 validation data.

<sup>4</sup>For GSM8K, which does not have a dedicated validation set, we sample 256 examples from its 7,473 training data.

sequentially takes data batches from these buckets in ascending order of difficulty, moving from simpler to more challenging tasks to progressively enhance its capabilities.

## B Effects of Different Components in State Fusion

The ablation experimental results in Figure 4 systematically evaluate the 4 state components, revealing the following importance ranking: **Stage > Instruction Difficulty > Instruction Semantic > Instruction Availability**. We provide a more detailed analysis here.

### B.1 Stage State

As the core component of RAISE’s dynamic selection, Stage State integrates current model performance and training progress to provide global temporal awareness. Figure 4 results show removing this component causes the largest average performance drop, as it directly controls training rhythm: guiding broad semantic category exploration in early stages (dispersed distributions in early/mid phases in Figure 6) and focusing on high-value samples later (dense distributions in later phases). Its global and dynamic nature makes it most significant in multi-task scenarios, serving as the key source of temporal sensitivity for dynamic value estimation.

### B.2 Instruction Difficulty State

Drives progressive learning by quantifying instruction complexity. Ablation experiments show significant performance drops on ARC-Challenge when removed, as this task heavily relies on complex reasoning (e.g., scientific knowledge inference). Works synergistically with Stage State: selecting easier samples early for stable training, then gradually introducing harder ones. However, in knowledge-intensive tasks (e.g., MMLU), difficulty metrics show weaker correlation with task objectives, making its impact relatively limited. This indicates Difficulty State’s effectiveness varies significantly by task type, being crucial for complex reasoning tasks.

### B.3 Instruction Semantic State

Captures semantic relevance through embedding vectors. Ablation shows significant impact on MMLU and CommonsenseQA but limited effect on ARC-Challenge. This difference stems from varying task requirements: MMLU needs cross-domain

knowledge generalization where semantic features prevent bias toward frequent domains (e.g., law, medicine). CommonsenseQA requires connecting similar common sense questions (e.g., causal reasoning) to extract patterns. ARC-Challenge’s scientific reasoning depends more on logical chain completeness than semantic discrimination. Thus, Semantic State offers limited benefits for reasoning tasks.

#### B.4 Instruction Availability State

Tracks selection frequency to prevent oversampling. Ablation shows minimal impact, with analysis revealing <10% cases of repeated sampling (>3 times) of same data. Its function becomes weaker with higher cluster counts (C). However, since it adds negligible overhead (just 1 extra dimension) and handles extreme oversampling cases, retaining it remains meaningful.

### C Dynamic Value

#### C.1 Static Value Framework

Existing approaches rely on *static value* computation. Let  $f : \mathcal{D} \rightarrow \mathbb{R}$  be the scoring function where  $\forall d \in \mathcal{D}$ ,  $f(d)$  represents its general utility. The subset selection objective is:

$$\mathcal{D}' = \arg \max_{N'} \{f(d) \mid d \in \mathcal{D}\} \quad (14)$$

$$\mathcal{J} = \max_f \mathcal{P}(\mathcal{M}[\mathcal{D}']) \quad (15)$$

where  $N'$  is the selection size and  $\mathcal{P}$  denotes evaluation metrics. Representative implementations include:

- **AlpaGasus** (Chen et al., 2023):  $f_{\text{GPT-4}}(d) = \text{Complexity}(d) + \text{FormatScore}(d)$

- **IFD** (Li et al., 2024):  $f_{\text{Llama}}(d) = \frac{\log p(y|x)}{\log p(y)}$

#### C.2 Dynamic Value Definition

We extend static values to *dynamic values* by introducing temporal dependency:

$$f : \mathcal{M}_{t-1} \times \mathcal{D} \rightarrow \mathbb{R}, \quad \text{where } t \in \{1, \dots, T\} \quad (16)$$

The training dynamics are governed by:

$$\mathcal{D}_t = \arg \max_B \{f_t(\mathcal{M}_{t-1}, d) \mid d \in \mathcal{D}\} \quad (17)$$

$$\mathcal{M}_t = \text{Update}(\mathcal{M}_{t-1}, \mathcal{D}_t) \quad (18)$$

with final objective:

$$\mathcal{J} = \max_f \mathcal{P}(\mathcal{M}_T) \quad (19)$$

### C.3 Conceptual Comparisons

- **Vs. Reward:** Reward  $R_t$  measures immediate performance improvement  $\Delta \mathcal{P}_t$ , whereas  $f(\mathcal{M}_{t-1}, \mathcal{D})$  estimates long-term utility of each sample.
- **Vs. Curriculum Learning:** Dynamic values adapt to emergent model states rather than pre-defined difficulty schedules.
- **Vs. Active Learning:** Focuses on final model capability rather than immediate uncertainty reduction.

### C.4 Example

Consider  $\mathcal{D} = \{d_1, d_2\}$  with  $B = 1$  at step  $t$ :

$$f(\mathcal{M}_{t-2}, d_1) = 0.8$$

$$f(\mathcal{M}_{t-2}, d_2) = 0.1$$

$$\Rightarrow \mathcal{D}_t = \{d_1\} \quad (\text{via Eq. 17})$$

The model updates as  $\mathcal{M}_{t-1} \rightarrow \mathcal{M}_t$  following Eq. 18. The value predictor  $f$  (we call it **acquisition function**) is optimized via RL.

### C.5 Parameter-to-Time Simplification

Directly using model parameters in  $f(\mathcal{M}_t, d)$  is computationally prohibitive. We simplify to  $f(t, d)$  because optimizers like Adam (Kingma, 2014) induce smooth parameter updates, making  $t$  a sufficient temporal index. The simplification reduces complexity from  $\mathcal{O}(|P| \times |\mathcal{D}|)$  to  $\mathcal{O}(|\mathcal{D}|)$ , enabling real-time computation.

## D Selection Policy Optimization

The optimization algorithm for the data selection strategy (acquisition function) is illustrated in Algorithm 2. This framework employs a bi-level iterative optimization process:

- The inner loop trains the LLM and adaptively updates the data selection strategy based on observed performance improvements during training.
- The outer loop iteratively refines the selection strategy over multiple rounds, ultimately achieving global optimization of the policy  $\pi_\theta$ .



---

**Algorithm 2** Selection Policy Optimization

---

```
1: Input: Training dataset  $\mathcal{D}$ , Validation set  $\mathcal{D}_{\text{val}}$ , Initial  
   LLM  $\mathcal{M}_0$ , Number of rounds  $K$ , Steps per round  $T$ ,  
   Batch size  $B$ , Fusion function  $\mathcal{H}$ , PPO epochs  $K_2$   
2: Output:  $\pi_{\theta_K}$   
3: Initialize  $\theta_0, \phi_0$   
4: for  $k = 1$  to  $K$  do  
5:   // Data Collection Phase  
6:   Initialize buffer  $W \leftarrow \emptyset$   
7:   for  $t = 1$  to  $T$  do  
8:      $S_t \leftarrow \mathcal{H}(\mathcal{D}, \mathcal{M}_{t-1}, t)$   
9:      $\mathcal{D}_t \leftarrow \pi_{\theta_{k-1}}(S_t, B)$   
10:     $V_t \leftarrow V_{\phi_{k-1}}(S_t)$   
11:     $\mathcal{M}_t \leftarrow \text{Update}(\mathcal{M}_{t-1}, \mathcal{D}_t)$   
12:     $R_t \leftarrow \mathcal{P}(\mathcal{M}_t) - \mathcal{P}(\mathcal{M}_{t-1})$   
13:     $W \leftarrow W \cup \{(S_t, \mathcal{D}_t, R_t, V_t, S_{t+1})\}$   
14:   end for  
15:   // Policy Optimization Phase  
16:    $\theta, \theta_{\text{old}}, \psi = \theta_{k-1}, \theta_{k-1}, \psi_{k-1}$   
17:   for  $k_2 = 1$  to  $K_2$  do  
18:     for  $(S'_t, \mathcal{D}'_t, R'_t, V'_t, S'_{t+1}) \subseteq W$  do  
19:       Update  $\theta$  by Eq.12  
20:       Update  $\psi$  by Eq.13  
21:     end for  
22:   end for  
23:    $\theta_k, \psi_k = \theta, \psi$   
24: end for  
25: return  $\pi_{\theta_K}$ 
```

---

## E Analysis of Computational Cost

RAISE introduces moderate additional overhead due to its two-stage structure: instruction preprocessing and acquisition function training. We denote  $\Phi$  as the cost of full-data LLM training.

**Instruction Preprocessing.** This stage computes instruction difficulty and semantic representations using an auxiliary model. Each instruction requires two forward passes. Assuming the auxiliary model is  $3\times$  larger than the target model, and training is  $5\times$  more expensive than inference per epoch, the preprocessing cost is:

$$\text{Preprocess} \approx \frac{2 \times 3}{5 \times 3} \Phi = 0.4\Phi$$

**Acquisition Function Training.** The acquisition function is trained via a bi-loop structure over  $K$  rounds. Each round performs one LLM training using a  $p$ -proportion data subset, costing  $p\Phi$ . Additional PPO and selection overheads are negligible ( $< 80$ s per round). Total training cost is:

$$\text{Train} \approx K \cdot p \cdot \Phi$$

Using  $p = 0.05$ ,  $K = 30$ , we obtain  $1.5\Phi$ .

**Total Cost.** The total computational cost of RAISE is approximately:

$$\text{Total} \approx 0.4\Phi + 1.5\Phi = 1.9\Phi$$

Despite a 90% increase over full-data training, RAISE enables targeted instruction selection and facilitates constructing high-quality, task-aligned datasets. For instance, aligning Alpaca with GSM8K as validation allows extraction of reasoning-specific data for specialized downstream training.

## F Memory Optimization Strategy

During RL training of acquisition function, RAISE maintains a replay buffer to store fused state vectors for each instruction. Given a state dimension of  $M$  and dataset size  $N$ , the memory cost scales linearly as  $O(N \times M)$ . This becomes prohibitive for large-scale instruction datasets (e.g.,  $N \geq 200,000$ ), where simultaneous access to multiple state vectors during batch sampling can exacerbate peak memory pressure.

To address this scalability issue, we propose a simple strategy: performing data selection every  $M$  steps instead of every training step. During intermediate steps, random sampling is used in place of selection. This reduces the frequency of policy updates and state maintenance, decreasing both computational and memory costs to approximately  $1/M$  of the original per-step selection setup.

This optimization leverages the observation that model parameters typically change slowly between adjacent training steps. Thus, the stage state remains similar over short horizons, and infrequent updates (e.g., every  $M$  steps) preserve most of the benefits of dynamic selection while significantly reducing overhead.

The interval parameter  $M$  provides a tunable trade-off between adaptivity and efficiency:

- **Fully Dynamic Selection** ( $M = 1$ ): Per-step selection with maximum adaptivity but highest memory and compute cost.
- **Fully Static Selection** ( $M = N$ ): Single selection before training with no runtime adaptation and minimal overhead.

This parameterization allows explicit control over the adaptation-efficiency trade-off in practical deployments. For very large-scale instruction tuning, this strategy enables RAISE to scale more efficiently without significant loss in performance.

## G Why “Less Is More”?

We observe that full-data training incurs significantly higher computational cost—up to  $100\times$

compared to selective training baselines—while resulting in comparable or only marginally improved performance. This finding may appear to contradict existing scaling laws. However, as noted in prior work (Zhou et al., 2024), scaling laws do not directly apply to the alignment phase, for the following reasons:

- The majority of LLMs’ core capabilities are acquired during pretraining.
- The alignment phase primarily teaches output patterns that activate these pretrained capabilities.
- Thus, only a small amount of high-quality data is sufficient for effective alignment.

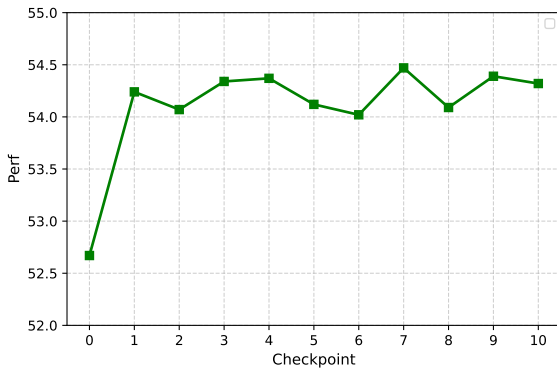


Figure 7: Model performance during training.

**Checkpoint Analysis.** To further examine this behavior, we conducted full-data training and saved 10 equally spaced checkpoints. Figure 7 reports performance across checkpoints. The results show that performance plateaus after early training steps, with only minor fluctuations throughout the rest of training. This reinforces the hypothesis that only a small subset of the full data contributes meaningfully to alignment.

## H Bias of Auxiliary Model

The Instruction Difficulty and Instruction Semantic features computed by auxiliary models are crucial for RAISE’s performance. We emphasize these components exhibit strong generalizability, while absolute values may differ significantly between architecturally distinct models (e.g., GPT vs. Llama), their feature distributions remain similar. To validate this experimentally, we implemented RAISE-Q using Qwen-2.5-7B-Instruct as the auxiliary model. The experimental results are shown in

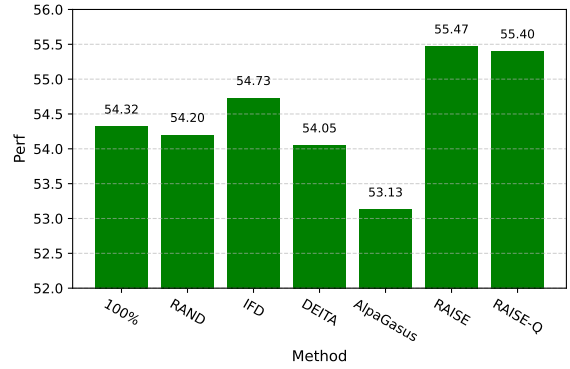


Figure 8: Experiment result of using Qwen-2.5-7B-Instruct as auxiliary model (RAISE-Q).

Figure 8. It can be observed that even when using an auxiliary model of another architecture, RAISE still exhibits excellent performance, indicating that RAISE’s dependence on the reference model is not particularly strong.

## I Case Study

In the experiments where GSM8K was used as the task objective, we were surprised to find that the selected instruction data were typical CoT data, which were highly aligned with GSM8K. Here we present two data cases for reference.

Solve for the inverse of the matrix

### Instruction:

Find the inverse of the matrix.

### Input:

$$\begin{bmatrix} 1 & -3 & 2 \\ 3 & 2 & 1 \\ 4 & 0 & -1 \end{bmatrix}$$

### Response:

We can find the inverse of the input matrix using the adjugate or classical adjoint method. Here are the steps:

**Solution:** First, we need to find the matrix of minors. The matrix of minors is a matrix where each element is the determinant of the minor for the corresponding entry in the original matrix. The minor for an entry is the determinant of the matrix that we get by removing the entry’s row and column.

The matrix of minors of  $\begin{bmatrix} 1 & -3 & 2 \\ 3 & 2 & 1 \\ 4 & 0 & -1 \end{bmatrix}$

is  $\begin{bmatrix} \begin{vmatrix} 2 & 1 \\ 0 & -1 \end{vmatrix} & \begin{vmatrix} 3 & 1 \\ 4 & -1 \end{vmatrix} & \begin{vmatrix} 3 & 2 \\ 4 & 0 \end{vmatrix} \\ \begin{vmatrix} -3 & 2 \\ 0 & -1 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 4 & -1 \end{vmatrix} & \begin{vmatrix} 1 & -3 \\ 4 & 0 \end{vmatrix} \\ \begin{vmatrix} -3 & 2 \\ 2 & 1 \end{vmatrix} & \begin{vmatrix} 1 & 2 \\ 3 & 1 \end{vmatrix} & \begin{vmatrix} 1 & -3 \\ 3 & 2 \end{vmatrix} \end{bmatrix} =$

$$\begin{bmatrix} -2 & -1 & -8 \\ 3 & -6 & -12 \\ -7 & -1 & 11 \end{bmatrix}.$$

Then we need to turn this matrix of minors into a matrix of cofactors by negating the elements at the positions where the sum of the row number and the column number is odd, i.e., alternate signs in the matrix of minors.

The matrix of cofactors is  $\begin{bmatrix} -2 & 1 & -8 \\ -3 & -6 & 12 \\ -7 & 1 & 11 \end{bmatrix}.$

Next, we have to find the adjugate of the matrix is by taking the transpose of the matrix of cofactors.

### 3.Boolean function:

```
def is_safe(board, row, col):
    for i in range(col):
        # Check row and columns
        if board[row][i] == 1:
            return False
    # Check diagonals
    for i, j in zip(range(row, -1, -1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    for i, j in zip(range(row, N, 1), range(col, -1, -1)):
        if board[i][j] == 1:
            return False
    return True
```

4. If a queen can be placed, move to the next row and repeat step 2. If no safe position is found for the current row, go back to the previous row and move the queen to the next safe position.

5. If all eight queens have been placed on the chessboard without attacking each other, the solution is found. Print the solution.

5. Do this for rows from 1-7 since the first row already has a queen.

6. Continue the process until all possible solutions are found, or if the solutions are exhausted, and none is found.

## Solve the Eight Queens puzzle

### ### Instruction:

Design an algorithm to solve the Eight Queens puzzle

### ### Response:

The Eight Queens puzzle is a classic problem where the goal is to place eight queens on an 8x8 chessboard in such a way that no two queens threaten each other. This means that no two queens can be placed in the same row, column, or diagonal.

One solution to this problem is to use backtracking. Here is an algorithm to solve the Eight Queens puzzle using backtracking:

1. Start with an empty 8x8 chessboard and add the first queen to the first row in the first column.

2. Move to the next row and try placing a queen in each column. Check if the position is safe by verifying if any of the previously placed queens is attacking the current position. If it's a safe position, place a queen in that column.

## Solve a Tower of Hanoi problem

### ### Instruction:

Create an algorithm for solving a Tower of Hanoi problem

### ### Response:

The Tower of Hanoi is a classic puzzle game with a goal of moving a stack of disks from one peg to another with the following three constraints:

1. Only one disk can be moved at a time.
2. Each move consists of taking the upper disk from one peg and moving it to another peg.
3. No disk may be placed on top of a smaller disk.

Given these constraints, we can use a recursive algorithm to solve the Tower of Hanoi puzzle.

Let's define the function 'towerOfHanoi(n, source, destination, auxiliary)' where 'n' is the number of disks, 'source' is the peg where the disks are initially stacked, 'desti-

nation' is the peg where the disks need to be transferred to and 'auxiliary' is the third peg that can be used as a temporary peg for moving the disks.

1. If 'n == 1', then move the top disk from 'source' to 'destination' peg. 2. If 'n > 1', then: a. Recursively call 'towerOfHanoi(n-1, source, auxiliary, destination)' to move the top 'n-1' disks from the 'source' peg to the 'auxiliary' peg using the 'destination' peg as the temporary peg. b. Move the 'n'th disk from the 'source' peg to the 'destination' peg. c. Recursively call 'towerOfHanoi(n-1, auxiliary, destination, source)' to move the 'n-1' disks from the 'auxiliary' peg to the 'destination' peg using the 'source' peg as the temporary peg.

Below is sample code for the algorithm in Python:

```
def towerOfHanoi(n, source, destination,
    auxiliary):
    if n == 1:
        print(f'Move disk 1 from {source}
            to {destination}')
    else:
        towerOfHanoi(n-1, source,
            auxiliary, destination)
        print(f'Move disk {n} from {source}
            to {destination}')
        towerOfHanoi(n-1, auxiliary,
            destination, source)
```

When the function is called with the appropriate initial values, it will print out the step-by-step solution to the Tower of Hanoi puzzle. For example:

```
towerOfHanoi(3, 'A', 'B', 'C')
```