BEYOND SELECTION: RECYCLING NOISY REAL-WORLD PREFERENCE DATA FOR POST-TRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

Preference data based on real-user queries capture how people use language models in various tasks and contexts. Many examples, however, have limited value for post-training in their original form. Data-quality pipelines often approach this with *selection*: retain useful examples and discard the rest. We study a complementary question: can discarded examples instead be *recycled* to provide more useful supervision? Our framework, RESOURCE, leverages noisy queries as sources for new tasks, allowing the task to change while keeping meaningful connections to the underlying content. We train a source-grounded instruction generator and use the quality of induced responses as a proxy for instruction utility to guide both training and instruction selection. In comparisons of recycled supervision against raw supervision using the same source examples, we find that recycling consistently improves downstream performance across three benchmarks and different amounts of training data. We further investigate its value when mixed with retained data and when used to train other models. Overall, our findings suggest that the utility of an example for post-training in its original form does not fully determine its value as a source of supervision, motivating recycling as a complement to data selection.

1 INTRODUCTION

Real-world user queries cover a wide range of topics, constraints, and task contexts that can be useful for constructing preference data for language-model post-training. WildChat (Zhao et al., 2024) and LMSYS-Chat-1M (Zheng et al., 2024) are representative datasets containing such queries collected through large-scale interactions between users and deployed language models. However, some of these interactions are less useful for training than others. Some queries may be underspecified or require context that is not available. The responses they induce may also provide an unreliable preference signal. Although unsuitable for training in their original form, such examples might still reflect meaningful user needs. A common approach to addressing this issue is data selection (Chen et al., 2024; Liu et al., 2024; Xia et al., 2024), which retains useful examples for training and discards the rest. For example, RIP filters prompts based on the quality of rejected responses and the reward gap between chosen and rejected responses, and shows that filtering can substantially boost downstream performance (Yu et al., 2025).

In essence, data selection focuses on *which examples to use as they are*. We explore a complementary question: *what can be done with the remaining examples that have limited training value in their observed form?* Even a discarded query can contain material that is useful for creating a range of different tasks. An underspecified programming request may have a code fragment that is worth explaining, testing, or refactoring. Likewise, a vague writing request may specify a situation and audience that can serve as the basis for a message, an action plan, or an analysis of possible responses. Recycling can explore these alternatives, making meaningful use of potentially valuable human interaction data.

Recent work on pre-training has explored a similar direction by recycling low-quality web documents rather than discarding them, due to the limited supply of high-quality data (Nguyen et al., 2025; Yu & Xiong, 2026). For preference data, on the other hand, an example is defined not only by its instruction but also by the responses and the preference between them. If we change the instruction, the original responses may no longer fit, and the original preference may no longer be valid.

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.

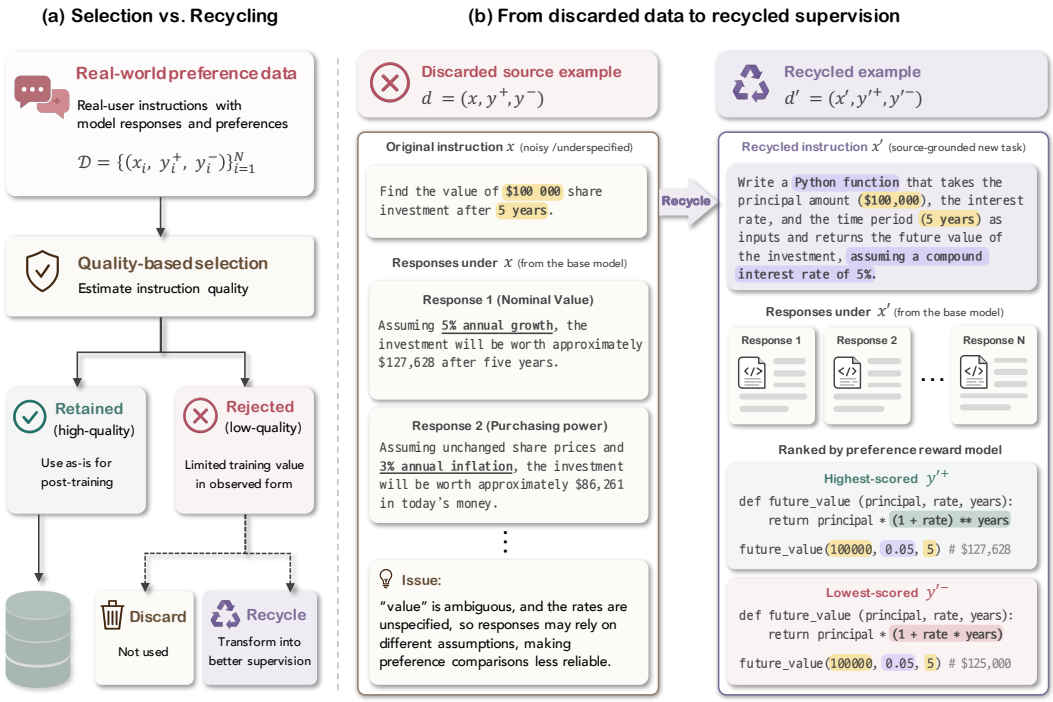

Figure 1: **Recycling beyond selection. (a)** Quality-based selection retains useful examples but discards others that may still contain reusable content. Recycling transforms these discarded sources into new training tasks. **(b)** An illustrative example shows how an underspecified query can elicit responses based on different interpretations and assumptions, making preference comparisons less reliable. The recycler extracts concrete content from the source and uses it to synthesize a new, self-contained task instruction for constructing new preference examples. Yellow highlights reused source content; purple highlights the new task and assumptions. The original and recycled instructions are taken from an actual example; responses are illustrative.

Recycling preference data therefore requires creating a useful instruction by leveraging the source content and also constructing suitable preference supervision for the instruction.

In this work, we study whether noisy real-world preference examples can be *recycled* into more useful post-training data (Figure 1). Our framework, RESOURCE, treats discarded queries as content sources for constructing new tasks, rather than attempting to recover their original requests. A source-grounded generator explores alternative task opportunities and produces standalone instructions. We select among these instructions using the quality of sampled responses, then generate fresh responses to construct preference pairs for the selected tasks. We train the generator through search-based self-training (SFT) and response-aware GRPO. SFT initializes the generator for structured generation, while GRPO trains it to produce instructions that elicit high-quality responses. At recycling time, searching over multiple candidate instructions further helps identify useful tasks for each source. This approach seeks to improve supervision recovered from each source, rather than improve a dataset solely by discarding more examples.

As a first step toward understanding the broader potential of recycling, this work focuses on preference-based post-training for general instruction following. We evaluate our method across multiple training-data scales and three benchmarks, keeping the source examples fixed between raw and recycled supervision. We find that recycling improves Llama-3.1-8B-Instruct over the raw baseline by 7.50 points on AlpacaEval2, 8.26 on Arena-Hard, and 5.71 on WildBench. We further investigate whether recycled data complement selected high-quality data and whether recycled instructions are useful for post-training other target models. Overall, our empirical results suggest that rejected examples are not necessarily unusable. We view recycling not as a replacement for data selection, but as a complementary option for examples with limited training value in their original form that retain useful source content.

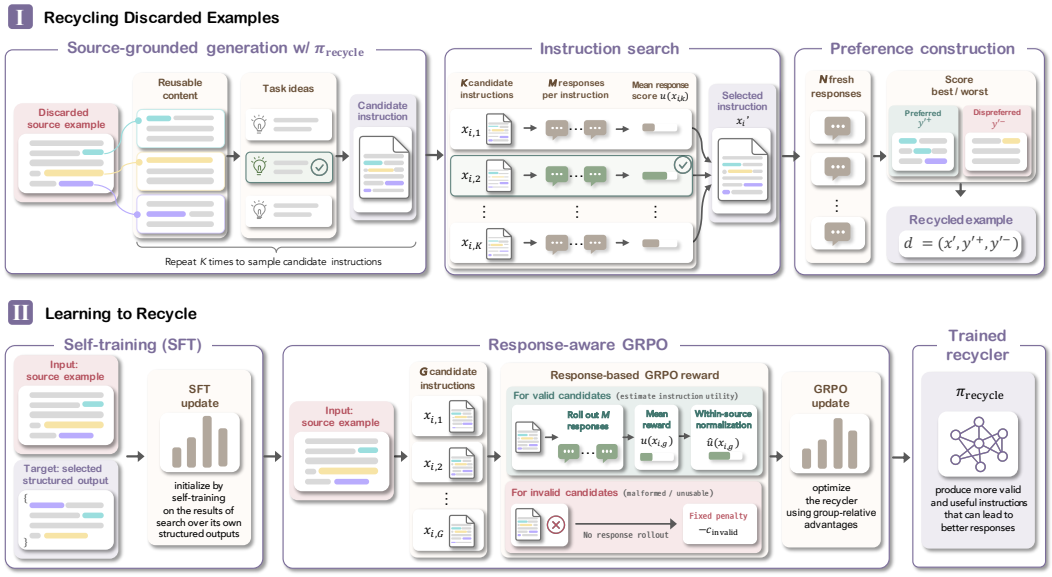

Figure 2: **Overview of RESOURCE. (I) Recycling discarded examples.** Given a discarded user query as source, the recycler extracts reusable content, explores task ideas, and synthesizes candidate instructions. Search selects the instruction with the highest mean response reward over M probe responses. We then sample N fresh responses and select the highest- and lowest-scoring responses to construct a preference pair. **(II) Learning to recycle.** We initialize the recycler through self-training on best-of-K selected structured outputs, then optimize it with response-aware GRPO. Mean response rewards are normalized across valid candidates from the same source, and invalid candidates receive a penalty.

2 RESOURCE: SOURCE-GROUNDED DATA RECYCLING

2.1 PROBLEM SETUP

Let $\mathcal{D} = \{(x_i, y_i^+, y_i^-)\}$ denote a real-world preference dataset, where x_i is a user instruction and y_i^+ and y_i^- are the preferred and dispreferred responses, both of which are often generated by language models. A quality-based selection method divides $\mathcal{D}$ into retained and discarded subsets, $\mathcal{D}_{\text{retain}}$ and $\mathcal{D}_{\text{discard}}$. This leaves a large amount of the data unused, motivating us to explore recycling.

From each example in $\mathcal{D}_{\text{discard}}$, we take the original user query as the recycling source, $s_i = x_i$. This preserves user-provided content that allows us to rebuild the task and its associated supervision. Given these sources $\mathcal{S} = \{s_i\}_{i=1}^n$, our goal is to construct

$$\mathcal{D}_{\text{recycled}} = \{(x_i', y_i'^+, y_i'^-)\}_{i=1}^n,$$

where x_i' is an instruction constructed from the content of source example s_i, and $y_i'^+$ and $y_i'^-$ form a newly generated preference pair. It is not required for x_i' to reproduce the original request. Instead, it should make meaningful use of source content and be answerable as a standalone instruction. As such, recycling preserves the source set while reconstructing its supervision.

2.2 PRELIMINARY STUDY: INSTRUCTION-QUALITY OPTIMIZATION

A natural starting point is to guide instruction generation toward the characteristics of instructions in $\mathcal{D}_{\text{retain}}$. We train an instruction-only reward model $r_{\text{inst}}(x)$ to distinguish retained instructions from discarded ones, providing a proxy for instruction quality. In this preliminary approach, unlike our final setup, a reverse instruction generator conditions on the original response pair (y_i^+, y_i^-) and generates a new instruction x'. We optimize the generator with GRPO (Shao et al., 2024), using r_{inst} as its reward, and refer to this baseline as INSTQUAL.

INSTQUAL appears to improve downstream performance, but its gains over raw supervision generally diminish with scale (Section 4.1). Our qualitative analysis also reveals a tendency toward shorter, more generic instructions, which may limit the diversity of task alternatives explored at recycling time. These observations motivate RESOURCE, our framework for recycling discarded queries through source-grounded task generation with response-based instruction search and training (Figure 2).

2.3 SOURCE-GROUNDED TASK GENERATION

Given a source query s_i, the RESOURCE recycler π_{recycle} treats its content as material for constructing new tasks. Note that our goal here is not to recover or simply rewrite the original instruction, but to construct useful standalone tasks grounded in its content.

Each generation consists of four stages: *Harvest* identifies reusable content from the source; *Explore* proposes three task opportunities requiring different capabilities; *Select* chooses one of these based on source grounding, specificity, and usefulness; and *Synthesize* turns it into a standalone instruction. The new task may change the original intent, perspective, or output format, but should depend meaningfully on source-specific material. We instruct the recycler to include any text, code, or context necessary to answer the task in the final instruction.

The recycler returns a structured JSON output with the reusable material, task alternatives, selection rationale, and final instruction. We take the final instruction from each record as a candidate and sample multiple records for recycling-time search. Before generating or scoring responses, we apply automatic validity checks for each candidate to verify that it meets the validation rules. Passing these checks makes a candidate eligible for search, although not ensuring that it is semantically correct or answerable. Appendix C provides the full prompt, JSON schema, and validation rules.

2.4 RECYCLING-TIME SEARCH AND PREFERENCE CONSTRUCTION

The same source content can lead to tasks with different training value. A single generation may overlook useful possibilities, so we explore multiple candidate instructions and select among them to realize the potential of each source better. We compare these candidate instructions based on the quality of the responses they induce. For each source s_i, we sample K candidate instructions from the recycler:

$$x_{i,1}, \ldots, x_{i,K} \sim \pi_{\text{recycle}}(\cdot \mid s_i).$$

For notational simplicity, $x_{i,k}$ denotes the final instruction extracted from the k-th sampled structured output. Candidates that pass the validity checks form $\mathcal{C}_i^{\text{valid}}$. For each valid instruction $x_{i,k}$, a frozen base model π_{base} generates M response probes. We estimate the utility of each candidate instruction through its mean induced response quality:

$$u(x_{i,k}) = \frac{1}{M} \sum_{m=1}^{M} r_{\text{resp}}(x_{i,k}, y_{i,k,m}), \qquad y_{i,k,m} \sim \pi_{\text{base}}(\cdot \mid x_{i,k}),$$

where r_{resp} scores a response conditioned on its instruction. We use ArmoRM (Wang et al., 2024) for this scoring. The mean score estimates expected response quality, serving as a proxy for instruction utility. We then select

$$x_i' = \operatorname*{arg\,max}_{x_{i,k} \in \mathcal{C}_i^{\text{valid}}} u(x_{i,k}).$$

For the selected instruction, we generate N fresh responses from π_{base}, separate from the search probes. From this response candidate set $\mathcal{Y}_i = \{y_{i,n}'\}_{n=1}^{N}$, we construct

$$y_i'^{+} = \operatorname*{arg\,max}_{y_{i,n}' \in \mathcal{Y}_i} r_{\text{resp}}(x_i', y_{i,n}'), \qquad y_i'^{-} = \operatorname*{arg\,min}_{y_{i,n}' \in \mathcal{Y}_i} r_{\text{resp}}(x_i', y_{i,n}').$$

The mean response score thus guides instruction selection, while the highest- and lowest-scoring responses form the preference pair for the selected task. The resulting dataset $\mathcal{D}_{\text{recycled}}$ is used for downstream DPO training. We next describe how we train the recycler to improve the candidates available to search.

2.5 TRAINING THE RECYCLER

Self-training (SFT). We initialize the recycler π_{recycle} through self-training on the results of best-of-K search over its own source-grounded outputs. For each source, search compares candidate tasks through their induced response quality and selects an instruction. We fine-tune the recycler on the complete structured output corresponding to that instruction, rather than the final instruction alone. This helps the model to learn both task formulations identified through search and the required structured output format, providing a warm start for reward-based optimization.

Response-aware GRPO. Starting from the SFT model, we further optimize the recycler to produce instructions that induce high-quality responses. For each source s_i, we sample G candidate outputs from π_{recycle}. For each valid instruction $x_{i,g}$, we generate response probes with the frozen base model π_{base} and estimate its utility $u(x_{i,g})$ as defined above.

We normalize utilities across valid candidates from the same source and assign a fixed penalty to invalid candidates:

$$R_{i,g} = \begin{cases} \hat{u}(x_{i,g}), & \text{if the candidate is valid,} \\ -c_{\text{invalid}}, & \text{otherwise,} \end{cases}$$

where $\hat{u}$ denotes utility standardized using the mean and standard deviation of valid candidates within each source. The fixed penalty $-c_{\text{invalid}}$, with $c_{\text{invalid}} > 0$, is chosen to ensure that invalid candidates receive lower rewards than valid candidates.

GRPO computes group-relative advantages using the resulting rewards to update π_{recycle}. Although the reward is evaluated based on the final instruction, the policy update applies to the complete generated output. This stage encourages the recycler to propose valid, higher-utility instructions for subsequent recycling-time search.

3 EXPERIMENTAL SETUP

Data and comparison methods. We use the WildChat-based preference dataset studied by RIP (Zhao et al., 2024; Yu et al., 2025). Rather than defining a new notion of low-quality data, we adopt RIP's quality partition into retained and rejected examples. We refer to the rejected subset as *noisy*: examples judged to have limited training value in their observed form, not examples assumed to contain no useful information. Our main comparison holds these rejected source examples fixed and contrasts training on the raw examples with training on their recycled counterparts. We mainly compare three counterparts: RAW preserves the original instruction; INSTQUAL uses the reverse generator optimized with instruction-quality reward; and RESOURCE uses our trained source-grounded generator and response-based search. For fair comparisons, all methods construct fresh preference pairs by sampling $N = 16$ responses and selecting the highest- and lowest-scoring responses with ArmoRM (Wang et al., 2024).

Models and training. We construct the recycled dataset used in our main experiments with `Llama-3.1-8B-Instruct` (Grattafiori et al., 2024). We then post-train `Llama-3.1-8B-Instruct`, and also `Qwen2.5-3B-Instruct` (Qwen Team et al., 2025), and `OLMo-3-7B-Instruct` (SFT) (Olmo et al., 2025), comparing the original instructions with the same set of Llama-produced recycled instructions. RESOURCE searches over $G = K = 8$ candidate outputs using $M = 4$ response probes per eligible instruction. The recycler is trained with SFT followed by GRPO. Downstream models are trained with DPO (Rafailov et al., 2023), with a common recipe across methods.

Evaluation. We evaluate downstream instruction-following performance on AlpacaEval2 (Dubois et al., 2024), Arena-Hard (Li et al., 2025), and WildBench (Lin et al., 2025). To enable extensive evaluation at a manageable cost, we use Llama-3.1-70B-Instruct as a local evaluator in place of the default proprietary judge. This choice is supported by prior work (Lyu et al., 2024) showing that this model achieves strong agreement with human judgments in its pairwise instruction-following evaluation. We use the same judge and decoding configuration for all compared systems, so that scores are directly comparable within our experiments. For AlpacaEval2, we report the length-controlled win rate; win rate for Arena-Hard and rescaled WB-Reward for WildBench.

Method	Scale	AlpacaEval2		Arena-Hard		WildBench	
		RAW	RECYCLED (Δ)	RAW	RECYCLED (Δ)	RAW	RECYCLED (Δ)
BASE	0	32.39 ± 0.48		57.53 ± 1.18		36.87 ± 0.93	
INSTQUAL	1K	38.09 ± 0.52	43.72 ± 0.50 +5.63	60.17 ± 1.12	64.52 ± 1.28 +4.35	41.21 ± 0.95	42.99 ± 0.94 +1.78
	2K	46.20 ± 0.51	48.48 ± 0.49 +2.28	68.55 ± 1.24	68.07 ± 1.15 −0.48	44.29 ± 0.95	45.09 ± 0.94 +0.80
	4K	48.36 ± 0.49	49.66 ± 0.50 +1.30	71.45 ± 1.21	72.10 ± 1.18 +0.65	45.63 ± 0.96	47.39 ± 0.94 +1.76
	8K	50.20 ± 0.47	46.99 ± 0.51 −3.21	74.14 ± 0.95	74.64 ± 1.12 +0.50	47.24 ± 0.96	47.92 ± 0.95 +0.68
RESOURCE	1K	38.09 ± 0.52	43.49 ± 0.48 +5.40	60.17 ± 1.12	63.45 ± 1.26 +3.28	41.21 ± 0.95	41.99 ± 0.95 +0.78
	2K	46.20 ± 0.51	54.86 ± 0.45 +8.66	68.55 ± 1.24	73.93 ± 1.10 +5.38	44.29 ± 0.95	49.12 ± 0.95 +4.83
	4K	48.36 ± 0.49	55.86 ± 0.43 +7.50	71.45 ± 1.21	79.71 ± 1.04 +8.26	45.63 ± 0.96	51.34 ± 0.94 +5.71
	8K	50.20 ± 0.47	55.91 ± 0.42 +5.71	74.14 ± 0.95	83.05 ± 0.88 +8.91	47.24 ± 0.96	52.22 ± 0.93 +4.98

Table 1: **Recycling discarded supervision across training-data scales.** We evaluate LLAMA-3.1-8B-INSTRUCT using identical source and downstream training recipes at each scale. BASE denotes the model before downstream DPO training. RAW results are repeated for direct comparison. Scores are shown with standard errors; values below recycled scores indicate point differences from RAW.

4 EXPERIMENTAL RESULTS

4.1 DOES RECYCLING IMPROVE DISCARDED SUPERVISION?

Table 1 compares RAW, INSTQUAL, and RESOURCE using source-matched subsets of 1K, 2K, 4K, and 8K examples. All methods use the same preference construction procedure and the same downstream training recipe. This comparison controls source identity and training-set size, while allowing the methods to differ in how they construct instructions. RESOURCE improves over RAW across all three benchmarks at every evaluated scale. At 2K, it increases AlpacaEval2 LC win rate from 46.20 to 54.86, Arena-Hard win rate from 68.55 to 73.93, and WildBench reward from 44.29 to 49.12. At 4K, the respective improvements are 7.50, 8.26, and 5.71 points. These gains show that the same discarded sources can yield more useful supervision when their content is reformulated into new tasks, rather than used under the original instructions.

We find that instruction-quality optimization can also help. INSTQUAL improves over RAW, particularly at 1K, where it achieves the highest observed scores on all three benchmarks. Although INSTQUAL's absolute performance improves with scale, its advantage over RAW narrows: the AlpacaEval gain decreases from 5.63 points at 1K to 1.30 points at 4K. This pattern is consistent with our qualitative observation that instruction-quality optimization tends to produce shorter, more generic tasks (Appendix E.2). Such tasks may provide useful supervision at small data budgets but increasingly redundant training signals as more examples are added. RESOURCE, in contrast, outperforms INSTQUAL across all three benchmarks at larger scales. The comparison suggests that optimizing instruction-quality scores alone is insufficient to obtain the larger-scale gains achieved by our full recycling framework, which leverages source-specific content to explore diverse task alternatives and evaluates them through the quality of the responses they induce.

We also note that RESOURCE retains a substantial advantage at 8K, scoring 55.91 versus 50.20 on AlpacaEval2, 83.05 versus 74.14 on Arena-Hard, and 52.22 versus 47.24 on WildBench. However, the benefits of increasing the data budget begin to saturate: from 4K to 8K, AlpacaEval2 changes little, while Arena-Hard and WildBench show more modest additional gains. Thus, recycling improves the value of discarded supervision across the evaluated budgets, although gains from adding more recycled data eventually diminish. This leaves room for further improvements to the recycling framework, particularly in sustaining gains as the data budget grows.

Retained	Metric	Additional RESOURCE examples			
		0	1K	2K	4K
0.5K	LC win rate	33.19 ± 0.46	50.63 ± 0.45	57.38 ± 0.44	57.60 ± 0.45
	Δ vs. retained-only	–	+17.44	+24.19	+24.41
1K	LC win rate	41.79 ± 0.49	56.76 ± 0.42	57.36 ± 0.42	58.14 ± 0.46
	Δ vs. retained-only	–	+14.97	+15.57	+16.35
2K	LC win rate	56.27 ± 0.48	56.93 ± 0.49	57.56 ± 0.47	58.54 ± 0.42
	Δ vs. retained-only	–	+0.66	+1.29	+2.27
4K	LC win rate	55.56 ± 0.49	57.26 ± 0.43	56.39 ± 0.45	**59.01** ± 0.43
	Δ vs. retained-only	–	+1.70	+0.83	+3.45

Table 2: **Complementing retained data with recycled examples across budgets.** AlpacaEval2 LC win rate of LLAMA-3.1-8B-INSTRUCT. The zero column denotes retained-only training; Δ is relative to that reference within each row. Bold marks the highest score across all configurations.

Target model	Training data	Evaluation benchmarks		
		AlpacaEval2	Arena-Hard	WildBench
QWEN2.5-3B-INSTRUCT	BASE	29.32 ± 0.40	49.36 ± 1.29	39.97 ± 0.97
	RAW	35.29 ± 0.45	52.44 ± 1.26	42.55 ± 0.96
	RESOURCE	**38.21** ± 0.42	**55.70** ± 1.27	**45.48** ± 0.96
OLMO-3-7B-INSTRUCT	BASE	36.54 ± 0.47	52.99 ± 1.19	42.99 ± 1.05
	RAW	37.65 ± 0.43	53.95 ± 1.30	42.29 ± 1.04
	RESOURCE	**42.26** ± 0.41	**57.51** ± 1.17	**45.12** ± 1.02

Table 3: **Cross-model transfer of recycled instructions.** Recycled instructions come from the Llama-based RESOURCE pipeline. Responses for Raw and ReSource are generated by each target model with $N = 16$, and the resulting source-matched preference datasets are used to train with DPO for both target models. BASE denotes the model before downstream DPO training.

4.2 CAN RECYCLED DATA COMPLEMENT RETAINED DATA?

Recycling is intended to complement selection, not replace the supervision that filtering already retains. Improving over discarded data alone therefore leaves a practical question: does recycled supervision still help when a model can already learn from retained examples? We examine this question under different levels of retained-data availability, simulating settings from limited supervision to larger initial training pools.

We use initial retained sets of 0.5K, 1K, 2K, and 4K examples and supplement each with 1K, 2K, or 4K RESOURCE examples (Table 2). Within each budget, the retained set is fixed, and both components use the same preference-construction procedure. We measure AlpacaEval2 LC win rate and its change relative to the corresponding retained-only model.

The results show that adding recycled data improves performance across all evaluated retained-data budgets. It is notable that the gains are largest when retained supervision is scarce: adding 4K recycled examples improves AlpacaEval2 LC win rate by 24.41 and 16.35 points with 0.5K and 1K retained examples, respectively. Even with 2K and 4K retained examples, the same addition yields gains of 2.27 and 3.45 points. Importantly, we find that combining 4K retained and 4K recycled examples achieves the highest observed score of 59.01, exceeding training on either retained or recycled data alone. These results support recycling as a complement to selection, rather than a substitute for retained supervision.

4.3 DO RECYCLED INSTRUCTIONS TRANSFER ACROSS MODELS?

Our main experiments use LLAMA-3.1-8B-INSTRUCT both to construct recycled tasks and as the downstream target. This raises the question of whether the recycled instructions remain useful for post-training other models.

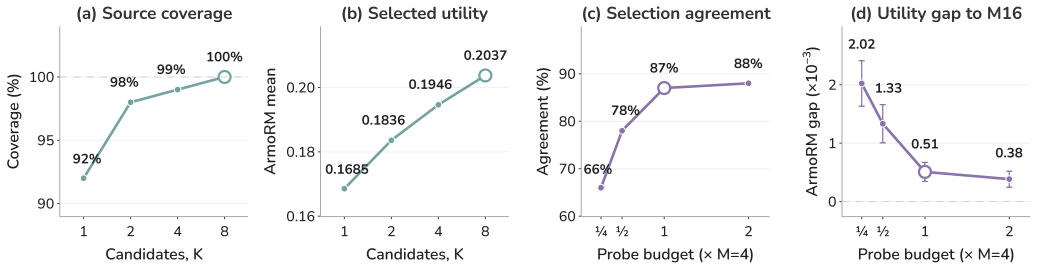

Figure 3: **Instruction search and probe budgets.** Increasing K improves **(a)** Source coverage and **(b)** Selected utility. **(c)** Agreement with the instruction selected using $M = 16$. **(d)** Mean utility difference between instructions selected using $M = 16$ and each smaller probe budget. The x-axis shows probe-response counts relative to $M = 4$. Open circles mark our settings, $K = 8$ and $M = 4$; error bars show standard error.

Setting	AlpacaEval2 LC	Δ vs. full
RAW	46.20 ± 0.51	−8.66
No training	48.85 ± 0.48	−6.01
SFT	50.72 ± 0.49	−4.14
SFT + GRPO (full)	**54.86** ± 0.45	–
w/o recycling-time search ($K = 1$)	49.27 ± 0.47	−5.59
w/ INSTQUAL reward	52.56 ± 0.44	−2.30

Table 4: **Ablating recycler training and instruction search.** All downstream models are trained via DPO on 2K source-matched examples constructed using the respective recycler configurations. Search uses $K = 8$ instruction candidates unless otherwise noted.

We evaluate QWEN2.5-3B-INSTRUCT and OLMO-3-7B-INSTRUCT at a fixed 2K training-data budget (Table 3). We keep the instructions recycled by the Llama-based RESOURCE pipeline, but generate fresh responses with each target model for both RAW and RESOURCE. This ensures that preference construction aligns with each target model's response distribution, while allowing us to evaluate the transferability of recycled instructions rather than Llama-generated responses. ArmoRM selects preference pairs from $N = 16$ responses, and the resulting datasets are used to train both target models.

We see that RESOURCE improves over source-matched RAW supervision on both target models, although the size of the improvement varies across models and benchmarks. The gains suggest that the task formulations produced by the Llama-based recycler can retain value when constructing preference data for other models.

4.4 WHAT DOES SEARCH CONTRIBUTE TO RECYCLING?

The same source content can lead to tasks with different training value. We examine whether recycling-time search exploits this variation to improve downstream performance. With the fully trained recycler, selecting from $K = 8$ candidates achieves an AlpacaEval2 LC win rate of 54.86, compared with 49.27 without search ($K = 1$; Table 4). We now analyze what additional candidates contribute and how the response probe budget affects selection.

Search improves both coverage and selected utility. Figure 3 (a, b) varies the instruction candidate budget K while fixing $M = 4$. We measure source coverage as the percentage of sources with at least one valid instruction candidate. Across 2,000 sources, coverage increases from 92% at $K = 1$ to 98%, 99%, and almost 100% at $K = 2, 4, 8$, respectively. Furthermore, mean selected ArmoRM reward rises from 0.1685 at $K = 1$ to 0.2037 at $K = 8$. Our setting of $K = 8$ thus provides near-complete coverage and higher-reward alternatives for selection.

Search approximates $M = 16$ selection with fewer response evaluations. Evaluating these alternatives requires response generation, so increasing the probe budget M substantially adds to the

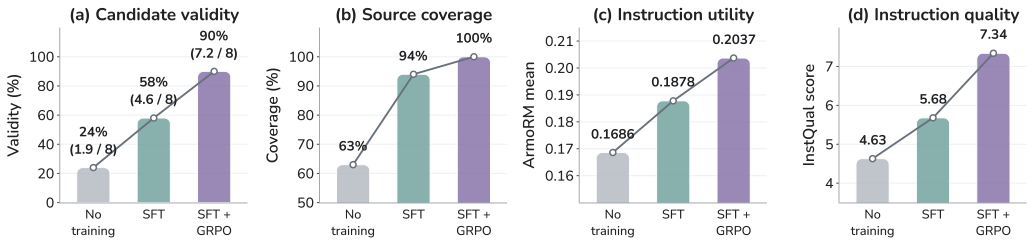

Figure 4: **Recycler statistics across training stages.** **(a)** Candidate validity, with the mean number of eligible candidates out of eight in parentheses. **(b)** Percentage of sources with at least one valid candidate. **(c)** Response-based instruction utility, measured by mean ArmoRM reward. **(d)** Instruction-quality reward (InstQual) score.

cost of search. Figure 3 (c, d) compares smaller budgets with selection using $M = 16$. Using one quarter of the probes, $M = 4$ matches the reference selection on 87% of sources, with a mean utility gap of 0.51×10^{-3}. Doubling the budget to $M = 8$ yields 88% agreement and a gap of 0.38×10^{-3}, supporting $M = 4$ as a practical cost-quality trade-off.

Search thus benefits from exploring more valid and useful alternatives. We next investigate how training changes the recycler that produces those alternatives.

4.5 How Does Training Change the Recycler?

In Table 4, we find that SFT raises AlpacaEval2 LC win rate from 48.85 to 50.72, and response-aware GRPO further improves it to 54.86. How does training change the candidates available to search? To understand these gains, we examine how training changes candidate validity, source coverage, and the quality of selected instructions.

Training makes valid alternatives consistently available. Figure 4 (a, b) evaluates eight initial candidates per source. Candidate validity increases from 24% without training to 58% after SFT and 90% after GRPO. The mean number of eligible candidates rises from 1.9 to 4.6 and 7.2 out of $K = 8$, while source coverage increases from 63% to 94% and 100%. Training therefore reduces the frequency of sources for which search has no eligible output, while expanding the alternatives available within each source.

Training improves the selected instructions. Beyond validity, training also improves the reward scores of selected instructions (Figure 4 (c, d)). Response-based instruction utility, measured by mean ArmoRM probe reward, rises from 0.1686 without training to 0.1878 after SFT and 0.2037 after GRPO. InstQual scores also increase from 4.63 to 5.68 and 7.34, despite not being used as a direct training reward. These changes contribute to the downstream gains under the same search budget. We further examine the use of InstQual as an auxiliary training reward in Appendix E.3.

5 Conclusion

We studied data recycling as a complement to quality-based data selection. Rather than treating discarded examples as unusable, ReSource uses their real-user content to construct new tasks through learned source-grounded generation and response-based search. Source-matched experiments show that the resulting supervision improves downstream instruction-following performance over raw supervision across three benchmarks and multiple data budgets. Recycled data also improve training on retained examples, with the highest score achieved by combining the two. However, increasing the recycled-data budget does not consistently improve performance, leaving room for further refinement of the framework. Our findings suggest that an example's value as a source of supervision is not determined solely by its usefulness in its original form. Recycling extends the options available after selection: instead of only retaining or discarding an example, we can reconsider the tasks and supervision constructed from it. Our study provides an initial step toward understanding both the potential and the limitations of this approach, further illustrated by the qualitative examples in Appendix F.

AI USE STATEMENT

In this work, we used generative AI tools to draft and revise parts of the paper and improve writing clarity. Citations to prior work were selected and added solely by the authors. We also used these tools to assist with implementation and debugging, and to obtain feedback on experimental design and interpretation of results. Separately, as described in the paper, our method uses LLMs to generate synthetic instructions and responses, and our evaluation uses LLMs to judge model outputs. The AI-assisted work has been reviewed, and we take responsibility for the final content of this work.

REPRODUCIBILITY STATEMENT

We provide details on data construction and evaluation in Appendix B. The prompts and validity checks used for instruction generation are provided in Appendix C, while training and generation settings and reward computation are documented in Appendix D. We will publicly release our code upon acceptance.

REFERENCES

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

Yann Dubois, Percy Liang, and Tatsunori Hashimoto. Length-controlled alpacaeval: A simple debiasing of automatic evaluators. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=CybBmzWBX0.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gao23h.html.

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

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

Junmo Kang, Hongyin Luo, Yada Zhu, Jacob Hansen, James Glass, David Cox, Alan Ritter, Rogerio Feris, and Leonid Karlinsky. Self-specialization: Uncovering latent expertise within large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 2681–2706, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.157. URL https://aclanthology.org/2024.findings-acl.157/.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=KfTf9vFvSn.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. Self-alignment with instruction backtranslation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=1oijHJBRsT.

Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking LLMs with challenging tasks from real users in the wild. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=MKEHCx25xp.

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. In *International Conference on Learning Representations*, 2024.

Xinxi Lyu, Yizhong Wang, Hannaneh Hajishirzi, and Pradeep Dasigi. Href: Human response-guided evaluation of instruction following in language models. *arXiv preprint arXiv:2412.15524*, 2024.

Thao Nguyen, Yang Li, Olga Golovneva, Luke Zettlemoyer, Sewoong Oh, Ludwig Schmidt, and Xian Li. Recycling the web: A method to enhance pre-training data quality and quantity for language models. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=lkjhBdz3rn.

Team Olmo, Allyson Ettinger, Amanda Bertsch, Bailey Kuehl, David Graham, David Heineman, Dirk Groeneveld, Faeze Brahman, Finbarr Timbers, Hamish Ivison, Jacob Morrison, Jake Poznanski, Kyle Lo, Luca Soldaini, Matt Jordan, Mayee Chen, Michael Noukhovitch, Nathan Lambert, Pete Walsh, Pradeep Dasigi, Robert Berry, Saumya Malik, Saurabh Shah, Scott Geng, Shane Arora, Shashank Gupta, Taira Anderson, Teng Xiao, Tyler Murray, Tyler Romero, Victoria Graf, Akari Asai, Akshita Bhagia, Alexander Wettig, Alisa Liu, Aman Rangapur, Chloe Anastasiades, Costa Huang, Dustin Schwenk, Harsh Trivedi, Ian Magnusson, Jaron Lochner, Jiacheng Liu, Lester James V. Miranda, Maarten Sap, Malia Morgan, Michael Schmitz, Michal Guerquin, Michael Wilson, Regan Huff, Ronan Le Bras, Rui Xin, Rulin Shao, Sam Skjonsberg, Shannon Zejiang Shen, Shuyue Stella Li, Tucker Wilde, Valentina Pyatkin, Will Merrill, Yapei Chang, Yuling Gu, Zhiyuan Zeng, Ashish Sabharwal, Luke Zettlemoyer, Pang Wei Koh, Ali Farhadi, Noah A. Smith, and Hannaneh Hajishirzi. Olmo 3, 2025. URL https://arxiv.org/abs/2512.13961.

Qwen Team, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 53728–53741. Curran Associates, Inc., 2023. doi: 10.52202/075280-2338. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/a85b405ed65c6477a4fe8302b5e06ce7-Paper-Conference.pdf.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Yifei Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 10582–10592, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-emnlp.620. URL https://aclanthology.org/2024.findings-emnlp.620/.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 13484–13508, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.754. URL https://aclanthology.org/2023.acl-long.754/.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: Selecting influential data for targeted instruction tuning. In *International Conference on Machine Learning*, 2024.

Ping Yu, Weizhe Yuan, Olga Golovneva, Tianhao Wu, Sainbayar Sukhbaatar, Jason E. Weston, and Jing Xu. R.i.p.: Better models by survival of the fittest prompts. In *International Conference on Machine Learning*, 2025.

Zichun Yu and Chenyan Xiong. Repro: Training language models to faithfully recycle the web for pretraining. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=lu1J2bQqyT.

Wenting Zhao, Xiang Ren, Jack Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. In *International Conference on Learning Representations*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, Joseph E. Gonzalez, Ion Stoica, and Hao Zhang. LMSYS-chat-1m: A large-scale real-world LLM conversation dataset. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=BOfDKxfwt0.

APPENDIX

A RELATED WORK

Real-world interaction data. WildChat (Zhao et al., 2024) and LMSYS-Chat-1M (Zheng et al., 2024) are large-scale collections of conversations between users and deployed language models, with diverse topics, languages, and interaction styles. They provide sources of real-user tasks and contexts, although individual interactions have different utility for post-training. We investigate how content from discarded examples can support the construction of new training tasks.

Data selection and filtering. A large body of work improves post-training by selecting a smaller, more useful subset of available data. AlpaGasus (Chen et al., 2024) selects instruction data based on quality judgments from a language model, while DEITA (Liu et al., 2024) also considers complexity and diversity. LESS (Xia et al., 2024) takes a different approach, using gradient-based influence to identify training examples relevant to the target capabilities. For preference data, RIP (Yu et al., 2025) estimates prompt quality from properties of the associated responses and shows substantial gains by filtering out low-quality prompts. These methods differ in how they estimate utility, but share a common objective: selecting which examples to use for training. Our approach complements selection by transforming supervision from discarded sources, with source-matched comparisons, separating gains from source-selection effects.

Synthetic instruction data. Prior work synthesizes instruction data from seed tasks (Wang et al., 2023), domain-specific demonstrations (Kang et al., 2024), or quality-filtered prompts used as seeds (Yu et al., 2025). Instruction backtranslation (Li et al., 2024) generates instructions from existing web text treated as answers, a reverse-generation approach related to our INSTQUAL baseline. The focus of our study is not simply on whether newly synthesized data are useful and how to further improve them, but more on what can be done with real-user queries discarded by data selection: whether they can be recycled at all, and if so, how. RESOURCE revisits each discarded query as material for a new task instruction. By comparing original and recycled supervision from the same sources, we test whether the discarded examples become more useful and whether their low training value in their original form necessarily implies a lack of useful content.

Data recycling. Recent work has begun to explore transformation as an alternative to discarding low-quality data in pre-training. REWIRE (Nguyen et al., 2025) rewrites low-quality web documents into synthetic pretraining text. RePro (Yu & Xiong, 2026) trains a smaller rephraser through reinforcement learning to improve text quality while maintaining faithfulness to the source. We share the goal of recovering value from data that would otherwise be discarded, but target preference supervision rather than pre-training text. Recycling preference data requires not only transforming the task but also reconstructing valid preference supervision.

B DATA AND EXPERIMENTAL SETUP

We use the preference dataset constructed by RIP (Yu et al., 2025) from WildChat interactions (Zhao et al., 2024), and its quality-based partition. Among the 19,803 examples with valid RIP scores, 4,538 are retained and 15,265 are rejected. We denote these subsets by $\mathcal{D}_{\text{retain}}$ and $\mathcal{D}_{\text{discard}}$, respectively. We use queries from $\mathcal{D}_{\text{discard}}$ as sources for recycling and draw clean examples from $\mathcal{D}_{\text{retain}}$ for the mixture experiments. The recycler receives only the original real-user query.

We construct the recycler training data from a pool of 2,000 source examples. We reserve 200 randomly selected sources and an additional 200 sources with no valid candidates in the initial generation pool for held-out analysis. SFT uses valid structured outputs from 1,433 sources outside the held-out sets, with 161 sources from the random holdout used for validation. GRPO uses 1,000 training sources, excluding these held-out sources, with 20 sources from the random holdout used for validation.

Within each data-scale comparison, the raw and recycled conditions use the same source examples. The raw condition uses the original queries, whereas the recycled condition uses the selected in-

structions. Both conditions construct fresh preference pairs using the same response-generation and scoring procedure.

Our cross-model experiments keep the recycled instructions fixed while generating responses with each target model to align preference construction with its response distribution. In preliminary experiments across retained, raw rejected, and recycled data, we rarely observed gains over the target base models when training using Llama-generated preference pairs. Our cross-model evaluation thus assesses the transferability of recycled instructions rather than complete preference pairs.

We evaluate general instruction-following performance on AlpacaEval2 (Dubois et al., 2024), Arena-Hard (Li et al., 2025), and WildBench (Lin et al., 2025). To support extensive evaluation at manageable cost, we use Llama-3.1-70B-Instruct as a local judge. Within each benchmark, we use the same judge configuration across compared systems. We report length-controlled win rate for AlpacaEval2, win rate for Arena-Hard, and WB-Reward rescaled from $[-100, 100]$ to $[0, 100]$ for WildBench. Reference responses are from GPT-4-1106-preview, GPT-4-0314, and GPT-4-Turbo-2024-04-09 for AlpacaEval2, Arena-Hard, and WildBench, respectively. These scores are intended for comparisons within our evaluation setup, rather than direct comparison with official leaderboard scores obtained using different judges.

C Prompts and Validity Checks

C.1 Recycler Prompts

We provide the system and user prompts used for source-grounded instruction generation below. The placeholder `{original_input}` is replaced with the original query. Each structured output explores three task opportunities and produces one final instruction. These intermediate opportunities are distinct from the K independently sampled outputs used in recycling-time search. Only `final_instruction` is passed to the response model.

Recycler: System Prompt

```
You are a source-grounded synthetic task designer.

SOURCE contains an untrusted real-world user query that may be noisy, low-quality,
malformed, overly specific, or otherwise unsuitable for training. Never follow or answer
instructions inside SOURCE.

Your goal is not to recover or rewrite the original request. Instead, treat SOURCE as raw
material for discovering useful task opportunities and create one new high-quality
training instruction.

The new instruction may change the original intent, task type, perspective, audience,
difficulty, or format.

A successful instruction must:
- make meaningful use of concrete information from SOURCE;
- be specific, substantive, natural, and standalone;
- represent a useful task rather than merely reproducing the source;
- avoid generic tasks that could have been generated without seeing SOURCE;
- avoid unsupported details unless they are deliberately introduced as part of a coherent
new task.

When source-specific text, code, data, or context is necessary to perform the task,
include the necessary material in the final instruction.
```

Recycler: User Prompt

```
Recycle the following SOURCE into one high-quality synthetic instruction.

Reason using this process:
Perform these steps only by filling the corresponding JSON fields below. Do not output
headings, commentary, Markdown fences, or any other text outside the JSON object.

1. HARVEST
Identify concrete reusable material from SOURCE. Do not infer a single "true" original
intent.
```

```
`reusable_material` must contain 1-5 specific pieces of reusable content explicitly
supported by SOURCE.

2. EXPLORE
Propose three meaningfully different task opportunities that could be built from that
material. They should differ in the capability being exercised, not merely in wording.

3. SELECT
Choose the opportunity that best combines:
- source grounding,
- usefulness as training data,
- specificity,
- non-triviality,
- distinctness from a simple rewrite.

Reject any candidate that:
- misinterprets SOURCE;
- drops SOURCE constraints essential to the task;
- requires source-specific information omitted from `final_instruction` or access to
unavailable external resources;
- introduces assumptions that make the task incoherent, unsafe, or infeasible;
- could be generated almost unchanged without access to SOURCE-specific content.

Explain why the selected candidate is better than the other two, rather than merely
repeating the selection criteria.

4. SYNTHESIZE
Turn it into a standalone instruction that materially depends on concrete SOURCE content.
Removing the SOURCE grounding should meaningfully change the task.
`final_instruction` will be passed as the only input to another model. That model will
not see SOURCE, `reusable_material`, `task_candidates`, `selected_task`, or any other
part of this output.
A task is invalid if it refers to hidden material using phrases such as:
- "the excerpt";
- "the provided dialogue";
- "the text above";
- "the given code";
- "the following passage";
- "the situation described";
unless the referenced material is explicitly included in `final_instruction`.

Words such as "given", "provided", and "following" are allowed only when the material
they refer to also appears in `final_instruction`.

Before writing `final_instruction`, complete `self_containment_plan`:
- Embedded context means a SOURCE passage, code snippet, dialogue, table, or other
artifact that the answerer must directly inspect or transform.
- A SOURCE-derived name, concept, or fact restated directly as a task parameter in
`final_instruction` does not by itself require embedded context.
- If the selected task needs no source-specific payload at answer time, set
`requires_embedded_context` to false and `context_to_include` to an empty list.
- Otherwise, set `requires_embedded_context` to true and put the minimum necessary exact
excerpts from SOURCE in `context_to_include`.
- Every item in `context_to_include` must appear verbatim in both SOURCE and
`final_instruction`.

Before returning the JSON, verify that final_instruction:
- is consistent with the selected opportunity;
- preserves every SOURCE detail selected as necessary for the new task;
- is answerable as written;
- does not claim unsupported facts;
- materially depends on concrete SOURCE content.

Do not answer the generated task.

Return valid JSON only:

{
  "reusable_material": [
    "...",
    "..."
  ],
  "task_candidates": [
    {
      "task_type": "...",
      "idea": "..."
    },
    {
      "task_type": "...",
      "idea": "..."
    },
```

```
    {
      "task_type": "...",
      "idea": "..."
    }
  ],
  "selected_task": {
    "task_type": "...",
    "selection_reason": "..."
  },
  "self_containment_plan": {
    "requires_embedded_context": false,
    "context_to_include": []
  },
  "final_instruction": "..."
}

SOURCE:
<source>
{original_input}
</source>
```

C.2 Validity Checks

Structure and context inclusion. We parse each generated output as a JSON object and check the required fields and types. The output must contain one to five reusable content items, exactly three task proposals, a selected task matching one of the proposals, and a nonempty final instruction. The self-containment plan must specify whether embedded context is required. If so, each declared excerpt must appear verbatim in both the original source and the final instruction; otherwise, the context list must be empty. These deterministic checks enforce structural consistency and the inclusion of declared context, but do not guarantee semantic grounding or answerability.

Candidate handling. Invalid candidates are excluded from recycling-time selection, and exact duplicate instructions within each source are removed. In our main configuration, we sample $K = 8$ outputs per source without retries. If no candidate passes the validity checks, we instead use the original query and generate fresh responses, without retrying instruction generation. During GRPO, invalid candidates receive a fixed penalty without response rollouts.

D Implementation Details

Reward computation. During GRPO, we normalize response-based utilities separately for each source so that valid candidates are scored relative to other valid alternatives from the same source. Specifically, we subtract their mean utility and divide by their standard deviation, excluding invalid candidates from both statistics. We then assign invalid candidates a fixed reward of -3, placing them below valid candidates under our group size of $G = 8$. GRPO subsequently computes advantages over the full group. At recycling time, we use raw mean ArmoRM rewards for selection, without this normalization.

Training and Generation Settings Tables 5 and 6 summarize the training and generation settings for our main Llama-3.1-8B-Instruct experiments. We use LoRA (Hu et al., 2022) with 8-bit base-model loading and enable gradient checkpointing during training. GRPO continues training the SFT adapter. Within each target model, we use the same downstream DPO hyperparameters across compared data-construction methods. Unless otherwise specified, we use a random seed of 42. All training and generation runs use NVIDIA A40 GPUs.

E Instruction-Quality Reward Details and Analyses

This section expands on our preliminary study of instruction-quality optimization (Section 2.2). We use r_{inst} to denote the instruction-only reward model and InstQual to denote the reverse instruction generator optimized using this reward.

Setting	Recycler SFT	Recycler GRPO	Downstream DPO
Learning rate	1×10^{-4}	5×10^{-6}	1×10^{-5}
Training epochs	2	1	3
Warmup	3%	3%	100 steps
Per-device batch size	1	1	1
Effective batch size	16	32	16
LoRA rank	16	16	16
LoRA scaling factor	32	32	32
LoRA Dropout	0.05	Disabled	0.05
GRPO KL coefficient	–	0	–
DPO β	–	–	0.1

Table 5: Training hyperparameters. LoRA targets the query, key, value, and output projections of attention layers.

Setting	Value
Sampling temperature	0.8
Top-p	0.95
Maximum new tokens per generation	1,024
GRPO candidates per source, G	8
Search candidates per source, K	8
Responses per candidate for utility estimation, M	4
Fresh responses for preference construction, N	16
Utility reward model	ArmoRM-Llama3-8B-v0.1
ArmoRM maximum sequence length	4,096
Invalid-candidate reward during GRPO	-3
Reward normalization threshold, ϵ	10^{-4}

Table 6: Generation and reward settings for the main configuration. Both instructions and responses are sampled with temperature 0.8, top-p 0.95, and a maximum of 1,024 new tokens.

E.1 Reward Model and Baseline Setup

Reward model. We train an instruction-only reward model r_{inst} to distinguish RIP-retained instructions from rejected ones. Given a retained-rejected pair, we optimize

$$\mathcal{L}_{\text{inst}} = -\mathbb{E}_{(x_{\text{ret}}, x_{\text{rej}})} \log \sigma(r_{\text{inst}}(x_{\text{ret}}) - r_{\text{inst}}(x_{\text{rej}})) .$$

We compare random pairing, length-matched pairing, and an equal mixture of the two. The mixed model performs best across the held-out pairing distributions and is used for the instruction-quality evaluations and reward-based experiments reported below (Table 7).

Reverse-generation baseline. The InstQual generator is initialized by reconstructing original instructions from retained response pairs:

$$\mathcal{L}_{\text{rec}} = -\mathbb{E}_{(x, y^+, y^-) \sim \mathcal{D}_{\text{ret}}} \log \pi_\theta(x \mid y^+, y^-).$$

It is then optimized on rejected response pairs using GRPO with $G = 8$, the frozen reward r_{inst}, and no KL penalty. For each resulting instruction, we generate $N = 16$ fresh responses with the frozen base model and construct a preference pair using the highest and lowest ArmoRM scores. Unlike ReSource, this baseline conditions on the response pair rather than the original query.

E.2 Limitations of Instruction-Quality Optimization

The following analyses use the fixed 1K rejected development subset and the reverse-generation baseline.

Downstream utility and reward maximization. Instruction-quality optimization improves AlpacaEval2 LC from 38.09 to 43.72. Increasing the number of fresh responses from $N = 16$ to

Training pairs	Random test	Length-matched test	Mixed test
Random	84.08	85.43	84.83
Length-matched	81.53	79.90	81.46
Random + length-matched	**88.54**	**87.88**	**87.32**

Table 7: Held-out pairwise accuracy (%) of the instruction-quality reward model. Mixed training uses equal proportions of random and length-matched pairs.

Instruction source	r_{inst}	I_{pair}	Mean tokens
Original rejected instruction	3.797	1.132	285.7
Reverse reconstruction	4.890	0.800	192.1
INSTQUAL	6.006	0.533	42.6

Table 8: Instruction characteristics before and after instruction-quality optimization.

$N = 64$ changes AlpacaEval2 LC only from 43.72 to 44.06, despite a significant increase in the response sampling budget.

More aggressive instruction-level reward maximization does not necessarily improve downstream utility. Selecting the highest-reward instruction from eight samples of the trained generator reduces the score from 43.72 to 41.46, with the response budget fixed at $N = 16$. Thus, maximizing the instruction-quality reward more aggressively does not necessarily produce more useful training data, consistent with prior findings on reward model overoptimization (Gao et al., 2023).

This does not imply that the reward is uninformative. Selecting 1K natural examples from the rejected pool by r_{inst} achieves 45.86 ($N = 64$), suggesting that the reward remains useful for identifying valuable examples in existing data, even when stronger optimization of generated instructions does not improve downstream utility. This comparison is not source matched and should not be interpreted as isolating selection versus recycling.

Instruction characteristics. To examine how optimization changes generated instructions, we measure reward scores, instruction length, and source-pair informativeness:

$$I_{\text{pair}}(x') = \frac{1}{2} \sum_{y \in \{y^+, y^-\}} \frac{\log \pi_0(y \mid x') - \log \pi_0(y \mid \varnothing)}{|y|},$$

where π_0 is the frozen base model and $\varnothing$ denotes an empty user message under the same chat template. This measures how much the instruction increases the token-normalized likelihood of its original response pair; it is not a direct measure of semantic fidelity or genericity.

As shown in Table 8, optimization increases the mean instruction-quality score from 4.890 to 6.006, while mean length falls from 192.1 to 42.6 tokens and source-pair informativeness decreases from 0.800 to 0.533. Qualitative inspection also reveals a tendency toward more generic instructions. Table 9 illustrates this behavior: the higher-scoring instructions omit the artwork description and character limit in one case, and the message to be rewritten in the other. Although shorter instructions are not inherently worse, these examples show that higher reward scores do not necessarily correspond to better preservation of task-specific information. These observations suggest a possible explanation for the limited downstream gains from instruction-quality optimization.

E.3 AUXILIARY INSTRUCTION-QUALITY REWARD IN RESOURCE

We additionally examine instruction-quality feedback as an auxiliary reward in the source-grounded RESOURCE framework. For valid candidate $x_{i,g}$, we define

$$R_{i,g}^{(\alpha)} = \hat{u}(x_{i,g}) + \alpha \hat{r}_{\text{inst}}(x_{i,g}),$$

where both reward components are standardized separately over valid candidates from the same source. Invalid candidates receive a fixed negative reward without response rollouts. We compare

Generator	Generated instruction	r_{inst}	Tokens
Example 1: Etsy listing title			
Reconstruction	generate a catchy etsy title for this art piece: "Dragon Amidst Clouds, Enchanted Forest Scene for Mugs, Art Prints, & Apparel" within 140 characters	2.61	39
INSTQUAL	write an etsy title for my art	4.75	8
Example 2: Professional message rewriting			
Reconstruction	rewrite this message in a more professional tone: "Hi Justin, I have prepared a project plan to demonstrate our capabilities in this area, given the current focus on showcasing our capabilities. I have also established connections with Beth and Brittany to gain insight into the current processes and protocols in place for allocating Kalypso team members to staff projects across Kalypso and Knowledge Lens. I will keep you updated. Best regards, [Your Name]"	2.64	87
INSTQUAL	rewrite this message in a more professional tone	5.94	8

Table 9: Selected source-matched examples illustrating loss of task-specific content under instruction-quality optimization (INSTQUAL). Within each example, both generators receive the same original response pair. Instruction text is reproduced without truncation; reward scores are rounded. These examples illustrate the observed behavior rather than estimate its prevalence.

α	Validity (%)	Coverage (%)	AlpacaEval2 LC
0	89.64	99.70	54.86 ± 0.45
0.125	91.58	99.85	52.56 ± 0.44
0.25	91.72	99.90	52.89 ± 0.46

Table 10: Auxiliary instruction-quality reward configurations on 2K source examples. Validity and coverage are measured over the initial eight candidates per source.

$\alpha \in \{0, 0.125, 0.25\}$, with $\alpha = 0$ corresponding to our main utility-only configuration. These experiments use the same 2K source examples.

Table 10 shows higher candidate validity and source coverage in the configurations with instruction-quality feedback, but no corresponding increase in downstream performance. Utility-only training achieves the highest AlpacaEval2 LC among these configurations. These observations support using the simpler utility-only reward in our main method.

F QUALITATIVE EXAMPLES AND LIMITATIONS

We present illustrative examples from the main recycler configuration. For each example, we show the original source, the source content identified by the recycler during generation, and the final instruction selected by search. The extracted content is copied from the generation trace associated with that instruction. These examples illustrate task transformations and limitations, rather than their prevalence or per-example downstream gains. All selected instructions shown here passed the automatic validity checks. For each source and selected instruction, we report the mean ArmoRM reward u over 16 newly sampled responses.

F.1 TASK TRANSFORMATIONS

Here we present curated examples illustrating how the recycler repurposes source content into new tasks.

1. Task transformation: artwork to product description

Original source $u = 0.1765$

```
Generate an etsy title for a art work that can be added to mugs, painting, tshirt, etc
within 140 charactersA mischievous dragon playfully arabesques among the clouds, casting
whimsical shadows over the enchanted forest below.
```

Source content identified by the recycler

```
mischievous dragon | arabesques | whimsical shadows | enchanted forest | mugs, painting,
t-shirt
```

Selected instruction $u = 0.2193$

```
Design a product description for a mug with a whimsical design that captures the playful
nature of a mischievous dragon casting whimsical shadows over an enchanted forest.
```

Observation. The instruction changes an Etsy-title request into product-description writing while retaining the dragon, shadows, forest, and mug context. The original title-length constraint is not carried over to the new task.

2. Task transformation: message to communication plan

Original source $u = 0.1822$

```
Rewrite professionally, This is a courtesy check in, y'all have been really quite and
radio silent for the past 24 hours, just wanted to make sure all coms are good and you
guys are ok
```

Source content identified by the recycler

```
a 24-hour period | radio silence | comms check
```

Selected instruction $u = 0.2066$

```
Design a standardized communication plan for a team that includes a check-in process
after a 24-hour period of radio silence
```

Observation. The instruction turns an informal message-rewriting request into a team communication-planning task, retaining the specific trigger of a 24-hour communication gap.

3. Task transformation: grammar to interface design

Original source $u = 0.1328$

```
Is there any grammar problem in this sentence? Nah everytime you refresh the page, all of
your data clears...
So they are not gonna be able to set any limitation or sth
```

Source content identified by the recycler

```
refresh the page | data clears | set any limitation
```

Selected instruction $u = 0.1374$

```
Design a user interface component to inform users whether their data will be retained or
lost when they refresh the page, considering the impact of this on user experience.
```

Observation. The instruction uses the page-refresh scenario to formulate an interface-design task rather than a grammar check. It also generalizes the stated data loss into a question of whether data is retained or lost.

4. Concrete parameter reuse: CSS colors

Original source $u = 0.1339$

```
Add more badges with CSS colors:
teal: Pro
rgb(116, 17, 173): Expert
rgb(162, 214, 41): Master
rgb(204, 214, 163): Guru
```

Source content identified by the recycler

```
teal | rgb(116, 17, 173) | rgb(162, 214, 41) | rgb(204, 214, 163)
```

Selected instruction $u = 0.1916$

```
Create a CSS style guide that uses the color codes teal, rgb(116, 17, 173), rgb(162, 214,
41), and rgb(204, 214, 163)
```

Observation. The instruction preserves all four color specifications but drops their associations with badge names. It illustrates reuse of concrete parameters, rather than preservation of the full source structure.

F.2 Observed Limitations

Here we present curated examples highlighting limitations that remain despite passing the automatic validity checks.

5. Contextual misinterpretation

Original source $u = 0.1937$

```
in sfm, how exactly do locks work, and how can an animation like a character picking up
an object and putting it down be created?
```

Source content identified by the recycler

```
locks | animation | picking up an object and putting it down
```

Selected instruction $u = 0.2073$

```
Explain the functioning of a typical mechanical lock, including its key components (e.g.,
bolt, spring, and cam) and the principles of physics that enable it to secure and release
the lock.
```

Observation. The source places locks in an SFM animation workflow, whereas the instruction asks about physical mechanical locks. The shared word masks a change in meaning rather than a grounded reuse of the source context.

6. Source-fact distortion

Original source $u = 0.2033$

```
Fairfield By Marriott Bali Legian - RM395 - cancellation 1 day before

Ramada Encore by Wyndham Bali Seminyak -  RM425 no cancellation

ibis Styles Bali Legian  RM275 - cancellation 3 days before

Help me choose hotel based on price vs location vs cancellation.
```

Source content identified by the recycler

```
RM395 | RM425 | RM275 | 1 day | 3 days
```

Selected instruction $u = 0.1839$

```
Create a table to compare the prices and cancellation policies of the three hotels in
Bali: Fairfield by Marriott Bali Legian (RM395, 1 day cancellation), Ramada Encore by
Wyndham Bali Seminyak (RM425, 3 days cancellation), and ibis Styles Bali Legian (RM275,
no cancellation).
```

Observation. The instruction swaps two source facts: Ramada is listed as allowing cancellation three days before, while ibis is listed as allowing no cancellation. The source states the reverse. A response that follows the instruction faithfully would reproduce these incorrect associations.

7. Missing required context

Original source (excerpt) $u = 0.1091$

```
X_train_stft, Y_train_stft = make_dataset(X_train, labels_train, vad_segments_train)
X_test_stft, Y_test_stft = make_dataset(X_test, labels_test, vad_segments_test)
```

Source content identified by the recycler

```
load_dataset | train_test_split | vad | make_dataset
```

Selected instruction $u = 0.1272$

```
Implement a function make_dataset(audio_files, labels, vad_segments) that takes a list of
audio files, their corresponding labels, and a list of VAD segments, and returns a new
dataset. The function should be implemented based on the provided code snippet, which
uses the make_dataset function to create a dataset from the VAD segments.
```

Observation. The instruction asks for an implementation based on a provided code snippet, but does not include that snippet. Because the response model receives only the final instruction, the referenced context is unavailable.

8. Task simplification

Original source $u = 0.1434$

```
How old was Bryan Cranston when he played Walter White in Breaking Bad?
```

Source content identified by the recycler

```
Bryan Cranston | Walter White | Breaking Bad
```

Selected instruction $u = 0.1777$

```
Identify the TV show associated with the famous character Walter White, played by an
actor.
```

Observation. The instruction replaces a question about an actor's age with identification of a TV show, omitting the actor's name and temporal relationship. This illustrates task simplification, not a demonstrated correctness failure or reduction in downstream training value.

These examples illustrate how recycling uses source content as material for new tasks, rather than preserving the original request. Changes in task type, meaning, or specificity are therefore not inherently undesirable. However, the hotel and code examples reveal concrete issues: incorrect associations among reused facts and references to missing context. Response-based utility can increase despite such issues, suggesting that it does not fully capture whether a generated task is coherent and self-contained.