Checklists Are Better Than Reward Models For Aligning Language Models

Vijay Viswanathan ♡ **Yanchao Sun**♣ **Shuang Ma**♣* **Xiang Kong**♣
Meng Cao♣ **Graham Neubig**♡ **Tongshuang Wu**♡
♡ Carnegie Mellon University ♣Apple

Abstract

Language models must be adapted to understand and follow user instructions. Reinforcement learning is widely used to facilitate this – typically using fixed criteria such as "helpfulness" and "harmfulness". In our work, we instead propose using flexible, instruction-specific criteria as a means of broadening the impact that reinforcement learning can have in eliciting instruction following. We propose "**Reinforcement Learning from Checklist Feedback**" (**RLCF**). From instructions, we extract checklists and evaluate how well responses satisfy each item—using both AI judges and specialized verifier programs—then combine these scores to compute rewards for RL. We compare RLCF with other alignment methods applied to a strong instruction following model (`Qwen2.5-7B-Instruct`) on five widely-studied benchmarks – **RLCF is the only method to improve performance on every benchmark**, including a 4-point boost in hard satisfaction rate on FollowBench, a 6-point increase on InFoBench, and a 3-point rise in win rate on Arena-Hard. These results establish checklist feedback as a key tool for improving language models' support of queries that express a multitude of needs.[2]

1 Introduction

Language models must follow user instructions to be useful. As the general public integrates language model-based assistants into their completion of daily tasks, there is an expectation that language models can faithfully follow the users' requests [Liu et al., 2024a]. As users develop more confidence in models' ability to fulfill complex requests, these models are increasingly given rich, multi-step instructions that require careful attention to specifications [Zhao et al., 2024, Zheng et al.].

Today's models are almost universally trained to follow instructions via a two-step process: instruction finetuning, followed by reinforcement learning from human feedback (RLHF). Instruction finetuning, where the model is trained to mimic responses generated by annotators [Raffel et al., 2019], has historically been the primary workhorse for imbuing language models with some amount of instruction-following ability [Wang et al., 2022, Chung et al., 2022, Xu et al., 2024, Lambert et al., 2024a]. Model developers then frequently employ RLHF, where the model is trained to generate responses that look more like labeled "good" responses than "bad" responses, as a refinement step to decrease the likelihood that the model exhibits predefined poor behaviors (typically harmful behaviors) [Ziegler et al., 2019, Bai et al., 2022]. Unlike "verifiable" tasks where reinforcement learning is a workhorse [DeepSeek-AI et al., 2025, Lambert et al., 2024a, Pyatkin et al., 2025], reinforcement learning remains difficult to utilize for ambiguous or "non-verifiable" tasks, such as instruction following. What would it take to make RL a general-purpose solution at eliciting desirable behaviors in subjective or open-ended settings?

*Work performed while at Apple.

[2]We plan on releasing our models, our dataset of checklists (*WildChecklists*), and code to the public shortly.

Preprint. Under review.

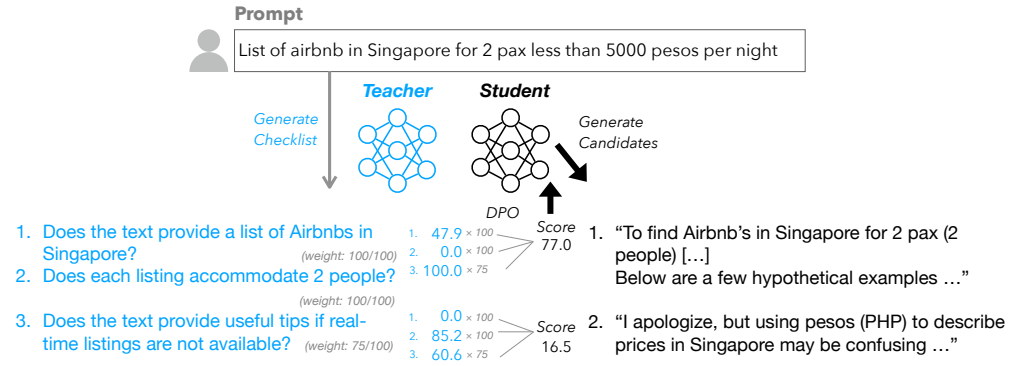

Figure 2: We propose *Reinforcement Learning from Checklist Feedback*, where sampled responses are evaluated by a teacher model grounded on a fixed set of criteria. In our pipeline, given instructions, we first generate checklists synthetically from the instructions, grade each response on each checklist item, combine per-item scores into a single weighted checklist score, then use this score for RL.

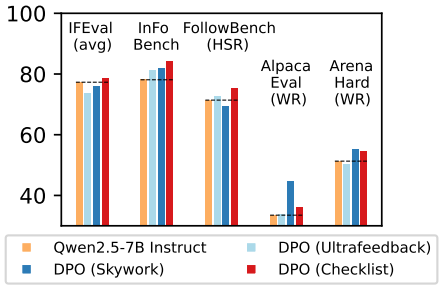

Figure 1: RL on Checklist Feedback consistently improves Qwen2.5 7B Instruct, whereas every other source of automatic feedback gives mixed results.

We believe the solution must involve producing better reward signals. Recent work on RL for language model alignment has focused on automatically obtaining feedback on model behavior, either by (1) exclusively using *instructions with verifiable answers* [Dong et al., 2024, Pyatkin et al., 2025], (2) grading responses with *specially-trained reward models* [Wang et al., 2024a, Eisenstein et al., 2023], or (3) *distilling preferences from a larger prompted model* [Bai et al., 2022, Tunstall et al., 2023]. Using *instructions with verifiable answers* restricts the aspects of response quality that can be learned to exact answer correctness or syntax/format adherence (ignoring other qualities, e.g. topicality or style). *Specially-trained reward models* are powerful, but their notion of rewards can be arbitrary, leading to reward hacking [Eisenstein et al., 2023]. When *distilling preferences from a larger prompted LM*, that LM is challenged with deciding what aspects to consider when grading a response, reducing the so-called "generator-verifier gap" that enables RL [Swamy et al., 2025]. Even if multiple prompts are written to capture values of interest, this assumes that a fixed set of criteria can be comprehensive [Bai et al., 2022, Glaese et al., 2022a].

In this paper, we ask: "how can we grade responses to instructions in a manner that is *automatic* (requires no human annotation), *flexible* (considers all aspects of response quality), *intuitive* (aligned with perceptible differences in responses), and *applicable to any instruction or response*, to enable more effective use of RL in language model alignment?" As an answer, we propose extracting dynamic checklists from instructions – an approach we term **Reinforcement Learning from Checklist Feedback (RLCF)**. This approach makes it likely that our evaluation focuses on flexible lists of distinct criteria while also reducing the problem of grading responses to answering a series of specific yes/no questions, which can be answered by an AI judge or by executing a verification program.

Our key contributions are:

1. We describe a new and improved algorithm for automatically generating checklists at scale.
2. We construct *WildChecklists*, a dataset consisting of 130,000 instructions and corresponding checklists (generated synthetically). When applicable, we accompany items in each checklist with a verification program to facilitate automatic evaluation. We plan to release this dataset to the community as an artifact for future study.
3. We describe a new algorithm for grading responses according to checklists, using language models and code, and we show to use this algorithm to rank responses for preference tuning.
4. We finetune `Qwen2.5-7B-Instruct` via reinforcement learning from checklist feedback using *WildChecklists*, leading to a strong and improved 7B-parameter model for instruction following.

On 5 benchmarks covering both constrained instruction-following (IFEval, InFoBench, FollowBench) and general conversational assistance (AlpacaEval, Arena-Hard), we find that RLCF provides benefits on all instruction following benchmarks while maintaining improved performance on general conversational assistance benchmarks. In contrast, all alternative forms of AI feedback lead to mixed results, as shown in Figure 1. RLCF provides a 5.4% relative improvement over Qwen2.5-7B-Instruct in average hard satisfaction rate on FollowBench, a 6.9% relative improvement in overall requirement following ratio on InFoBench, and a 6.4% relative improvement on Arena-Hard [Jiang et al., 2023, Qin et al., 2024, Li et al., 2024]. Despite these considerable improvements, RLCF simply requires a teacher model, with no need for additional data or human annotations, making this approach amenable to diverse languages or domains. We provide evidence that checklist-based rewards are well-correlated to human preference judgments (comparable to many finetuned reward models) while providing a stronger learning signal than alternatives.

2 Checklist Generation

Desiderata for checklists. We define a *checklist* as a sequence of *requirements* paired with an instruction that satisfy the following properties:

1. Each requirement in the checklist is a yes/no question (e.g. "Does the text contain 3 commas?").
2. Each requirement in the checklist must be answered relative to a given candidate response.
3. A response would be considered acceptable if and only if the response answers "yes" to all checklist requirements.

To satisfy definition #3, checklists must be *comprehensive* (cover most relevant aspects of quality) and *natural* (entailed by their corresponding instructions). Based on the observation that false positive rewards are often more detrimental to reinforcement learning than false negatives [Huang et al., 2024], we want checklists that are *objective* (facilitate automatic verification) and *atomic* (each requirement focuses on a single aspect of quality), to make requirement checking easier.

Extract checklists per instruction. We examine two methods to extract checklists:

- `Direct`: We simply prompt an LM to extract a checklist from a given instruction [Cook et al., 2024]. This approach is intuitive and simple but risks repeating the original instruction via these individual criteria, which may limit *comprehensiveness* and *objectiveness*.
- `Candidate-based`: We view a requirement as any aspect of an instruction that, when absent, causes a response to fail. We propose a two-stage approach: produce responses of varying quality, then prompt an LM to write a checklist of all their possible failure modes. For each checklist item, we also prompt the model to generate an "importance" weight (from 0 to 100).

To compare these, we generate checklists for all instructions in InFoBench [Qin et al., 2024]. We use `gpt-4o` to blindly evaluate each of these checklists on naturalness, objectivity, comprehensiveness, and atomicity, then select the better one overall. We manually perform the same evaluation on a *subset of 50 instructions* from the "Easy Set" of InFoBench.

The results in Table 1 show that checklists generated by prompting an LLM directly are more natural. However, providing candidate responses to the LLM leads to checklists with consistently better objectiveness, atomicity, and overall quality. There are absolute differences between scores from the two evaluations – partly because they use different subsets – but directional trends are consistent. We find that this difference translates to downstream performance after performing RL training. In Section 5.3, we show that Reinforcement Learning from Checklist Feedback is more effective on checklists generated via the candidate-based method.

Regularization via universal criteria. In initial experiments, we found that optimizing for checklist completion led models to sometimes generate high-level overviews of the response instead of the intended response, suggesting reward hacking. In prior work, Sun et al. [2023] reported a similar issue when training models with instructable reward models, addressed by adding three manually-chosen instructions to their reward model in all cases. Following this and other works that perform RL using global principles [Glaese et al., 2022b, Bai et al., 2022], we added one "universal requirement" to all generated checklists. This universal requirement stated "*Does the response satisfy the following two criteria: 1) The response directly address the request without excessive or off-topic information not necessary for addressing the user's instruction? 2) The response should match the context and*

	Manual Evaluation		Automatic Evaluation	
Metric	**Direct**	**Candidate-Based**	**Direct**	**Candidate-Based**
Naturalness	**94.9**	93.9	**88.0**	85.1
Objectiveness	88.5	**91.9**	88.9	**89.7**
Comprehensiveness	74.0	**82.0**	**69.2**	64.8
Atomicity	68.0	**90.0**	98.6	**99.0**
% Preferred Overall	38.0	**56.0**	40.6	**51.2**

Table 1: We evaluate two checklist generation methods on four specific aspects of quality and an overall preference. Manual evaluation is performed on the first 50 rows of InFoBench "easy", while automatic evaluation is performed by `gpt-4o` on all 500 rows of InFoBench.

the instruction, whether it requires professionalism, friendliness, formality, or neutrality.", with a corresponding importance weight of 100/100.

Dataset Generation Using the candidate-based method, we generate checklists for 130,000 instructions from WildChat to create a new dataset, *WildChecklists*. To generate candidate responses for our method, we use `Qwen2.5-0.5B`, `Qwen2.5-1.5B`, `Qwen2.5-3B`, and `Qwen2.5-7B` [Yang et al., 2024]. `Qwen2.5-72B-Instruct` is the checklist generator model for both methods.

3 Reinforcement Learning from Checklist Feedback

Given *WildChecklists*, we generate high-quality preference data for RL via a four-step process:

Sampling Candidate Responses. To facilitate offline RL, we first sample response pairs from our base policy. For each prompt, we sample two responses with a temperature of 1.3 and a *top-p* of 0.9 [Holtzman et al., 2019]. This is simpler than prior works on RL-based language model alignment that synthetically perturb prompts to induce greater complexity [Sun et al., 2024, Dong et al., 2024].

Flexible Scoring Given a prompt, a response, and an individual checklist item, we use a combination of an LM judge and a verifier program to grade the response. For each checklist item, the judge model (`Qwen2.5-72B-Instruct`) generates a numerical score between 0 and 100. The prompt we use for grading responses is shown in Appendix B. To reduce the variance of this score, we sample 25 numerical scores from the model, and we then take the average of these 25 scores[3].

When applicable, we also use a verifier program to perform grading. LLMs struggle to evaluate criteria that measure discrete properties of text, such as "does the response contain the letter R at least three times?" or "does the response contain one of the following keywords [...]?" [Fu et al., 2024]. To better handle such constraints, we follow prior work in generating a verification program when applicable [Dong et al., 2024, Zhou et al., 2023]. Our prompt, listed in Appendix B, asks the model to produce code only when the model is highly confident the requirement can be exactly verified with a program. If the program successfully processes a response string, we convert the Boolean result to an integer (0 or 100), which is averaged with the AI judge's score.[4]

Preference Tuning. Given a separate numerical score for each criterion for each response, we take the weighted average of these scores, weighting by the importance score generated for each criterion. To produce a more informative learning signal, we keep only the 40% of response pairs with the greatest difference along at least one criterion of its corresponding checklist. This removes response pairs that are too similar in quality to offer a useful reward signal. We then assign the higher scoring response as "chosen" and the lower as "rejected", and we use these as a preference pair for direct preference optimization [Rafailov et al., 2023].

[3]We sample responses using the `n` parameter in vLLM [Kwon et al., 2023]. This approach follows prior work that describes the importance of using the mean score rather than mode score from an LM-as-a-judge model [Wang et al., 2025]. Regardless, this makes the AI judge component the computational bottleneck of our pipeline. In Section 5.5, we show that `n` can be significantly reduced, at a modest accuracy cost.

[4]This approach is much simpler than the most relevant prior work that uses programs to evaluate responses, AutoIF [Dong et al., 2024], which uses test-case generation and LM-based filters to remove low-quality programs.

4 Experimental Setup and Results

4.1 Experimental Details

Training Data As a fixed source of instructions for all methods, we use WildChat, a set of natural conversations between users and AI language models crowdsourced from users across the world [Zhao et al., 2024]. We filter out conversations that are non-English, toxic, or longer than two turns.

Models We experiment with finetuning `Qwen2.5-7B` and `Qwen2.5-7B-Instruct`. To produce AI judgments or ground truth responses, we use `Qwen2.5-72B-Instruct` unless stated otherwise.

Training We finetune the model for 2 epochs using DPO with a batch size of 1024 and a maximum sequence length of 2048. We use a cosine learning rate schedule with a max LR of 3e-6 and a min LR of 2e-6[5]. We use OpenRLHF for training [Hu et al., 2024], and we train on one 8xH100 node with 80GB GPU memory, which took roughly 3 hours for each model.

Benchmark Data We evaluate our method on five benchmarks: IFEval [Zhou et al., 2023], InFoBench [Qin et al., 2024], FollowBench [Jiang et al., 2023], AlpacaEval [Dubois et al., 2024], and Arena-Hard [Li et al., 2024]. The first three of these measure instruction following ability in the presence of fine-grained constraints. The last two measure "general-purpose" instruction-following ability, using naturalistic instructions based on user queries collected in the wild.

4.2 Baselines

To show that RLCF is more effective than existing approaches, we compare against baselines: *instruction finetuning*, *specially-trained reward models* (using either a single reward or mixture of rewards), and *prompted AI judges* (using either a single evaluation rubric or a mixture of rubrics).

Instruction Finetuning: We compare with instruction finetuning, to isolate the benefit of additional knowledge from the manner it is given (ground truth or rewards). Here, we distill [Hinton et al., 2015] from a larger model, `Qwen2.5-72B-Instruct`, finetuned via `LlamaFactory` [Zheng et al., 2024].

Reward Models: We mirror our training approach for learning from checklist feedback, but using state-of-the-art reward models to decide which response should be chosen or rejected. Here, we keep the 40% of prompts and responses with the greatest difference in scalar rewards. We consider following reward models as baselines (`Skywork/Skywork-Reward-Gemma-2-27B`) [Liu et al., 2024b] and `ArmoRM-Llama3-8B-v0.1`) [Wang et al., 2024b]. Both are highly rated on RewardBench [Lambert et al., 2024b][6], and ArmoRM has been very effective for alignment in prior work [Meng et al., 2024].

Prompted AI Judge: We lastly compare against using the same "teacher" model as a judge, without using checklists. We query this teacher in two settings: 1) *"Ultrafeedback"*, where the judge rates all candidate responses from 1-5 [Cui et al., 2023] separately across four quality aspects (instruction following, helpfulness, truthfulness, honesty) and averages these scores; and 2) *AI Judge*, where a near-identical prompt as RLCF is used (§3) to similarly sample 25 scores between 0 and 100 from the judge. This uses the AI judge the same way as RLCF, just without a checklist.

In Figure 3, we unify these methods of automatic evaluation to distinguish our method from prior art. In this context, checklist feedback can be viewed as a very large mixture of prompted evaluators.

5 Results

5.1 RL from Checklist Feedback consistently improves language models

Our proposed approach, RLCF, demonstrates consistent gains across all benchmarks (Table 2, Table 3, and Table 4). On IFEval's "loose" metrics (which apply minor preprocessing to responses before checking for correctness), RLCF improves `Qwen-7B-Instruct` by 2.8-3.0% (relative), as shown in the left half of Table 2. On FollowBench (shown in Table 3), RLCF achieves an 8.2% increase on Constraint Satisfaction Level (CSL; the expected proportion of constraints satisfied) and

[5]When training models with Ultrafeedback, we instead used a minimum learning rate of 3e-7. We found this parameter resulted in a slightly stronger baseline when learning from this feedback.

[6]`Skywork/Skywork-Reward-Gemma-2-27B` and `ArmoRM-Llama3-8B-v0.1` are ranked as #4 and #24, respectively, on RewardBench as of July 2025.

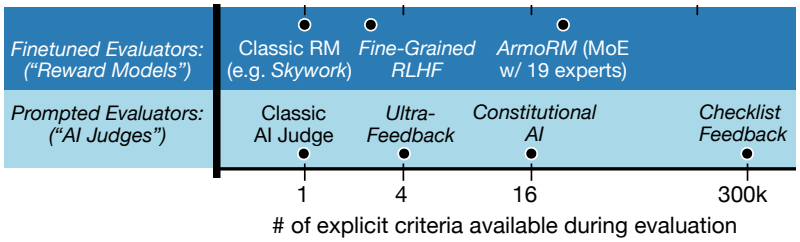

Figure 3: Checklist feedback can be viewed as an extreme mixture-of-evaluators, where the space of (prompted) evaluators is unbounded and a unique subset of evaluators is chosen for each instruction.

	IFEval (prompt)		IFEval (inst.)			InFoBench		
	Loose	**Strict**	**Loose**	**Strict**	**Avg**	**Easy**	**Hard**	**Overall**
GPT-4	79.3	76.9	85.4	83.6	81.3	89.3	86.4	87.3
+ *Qwen2.5-7B-Instruct*	75.0	72.5	81.8	79.9	77.3	82.7	76.0	78.1
+ SFT (Distilled)	66.9	64.1	75.3	72.8	69.8	79.9	70.6	73.5
+ DPO (via Skywork)	75.8	68.0	83.2	78.5	76.0	81.0	82.4	82.0
+ DPO (via ArmoRM)	73.8	70.2	81.7	78.3	76.0	**84.2**	83.1	83.5
+ DPO (via Ultrafbk.)	71.5	69.1	79.9	77.7	74.6	82.3	79.0	80.0
+ DPO (via AI Judge)	73.0	68.9	80.9	77.8	75.2	81.0	73.9	76.1
+ DPO (RLCF)	**77.3**	72.6	**84.1**	80.3	**78.6**	**84.2**	**84.0**	**84.1**
Qwen2.5-7B (base)	35.7	30.5	46.6	42.1	38.7	68.8	77.4	74.8
+ SFT on WildChat	38.1	33.5	52.2	48.6	43.1	78.1	80.1	79.5
+ DPO (RLCF)	**43.4**	**35.9**	**56.4**	**49.2**	46.2	**80.6**	**80.5**	**80.5**

Table 2: RLCF improves performance modestly on a format-based constrained instruction following benchmark (IFEval) and significantly on an open-ended constrained instruction following benchmark (InFoBench). RL on rewards from off-the-shelf reward models help on InFoBench but hurt on IFEval. We show positive results (relative to the baseline) in blue, negative in orange, and neutral (within 0.5) in gray; the top variant of a given model is bolded.

a 5.5% increase on average Hard Satisfaction Rate (how often all constraints are satisfied). RLCF also performs competitively on InFoBench (right half of Table 2), achieving results comparable to the best-performing reward model-based approaches while maintaining consistent gains across all constrain-based benchmarks. On "general use-case" instruction-following benchmarks, RLCF consistently increases the win rate of `Qwen2.5-7B` over `GPT-4` (shown in Table 4), with the relative improvement ranging from 2.8% to 8.4%.

5.2 Comparing automatic evaluators

In Table 2, Table 3, and Table 4, we observe our approach of performing RL from Checklist Feedback (RLCF) outperforms RL from other sources of automatic evaluation across most benchmarks. However, off-the-shelf reward models show mixed results depending on the benchmark. *Skywork* (`Skywork-Reward-Gemma-2-27B`), a leading model on the RewardBench leaderboard, shows strong improvements with RLHF on InFoBench, Arena-Hard, and AlpacaEval – RLHF via Skywork notably outperforms RLCF on AlpacaEval by a large margin. However, Skywork-guided RLHF leads to notable regressions as on IFEval and FollowBench. Similarly, RLHF with *ArmoRM* shows significant improvements on AlpacaEval and InFoBench, modest/mixed results on Arena-Hard and FollowBench, and significant regressions on IFEval.

In addition to measuring its downstream performance as a preference annotator, we perform intrinsic evaluation of checklist feedback on RewardBench[7]. In Table 5, we see that checklist-based scores are reasonably well-correlated with preference annotations on RewardBench, especially for the "Chat" and "Chat Hard" categories [Lambert et al., 2024b]. However, specialized reward models (*Skywork*,

[7]Unlike our method for checklist generation on WildChat, here we do not use any ground truth or output from other models when generating checklists.

FollowBench Level	*Soft Satisfaction Rate* L1	L2	L3	L4	L5	Avg	*Hard Satisfaction Rate* L1	L2	L3	L4	L5	Avg	CSL
GPT-4	89.2	89.3	87.6	88.1	84.9	87.8	89.2	87.6	83.6	83.0	75.1	83.7	3.52
Qwen-7B-Instruct	87.4	84.0	83.0	79.6	79.0	82.6	87.4	80.6	72.3	62.2	54.4	71.4	3.05
+ SFT (Distilled)	87.5	83.2	84.4	76.8	74.9	81.4	87.5	78.3	73.9	60.7	49.1	69.9	2.90
+ DPO (Skywork)	79.6	84.1	77.7	77.7	78.1	79.4	79.6	81.1	67.4	62.9	56.5	69.5	2.88
+ DPO (ArmoRM)	86.4	84.6	79.1	79.2	76.9	81.2	86.4	82.9	69.0	63.9	49.7	70.4	3.10
+ DPO (Ultrafbk.)	88.5	84.1	82.5	76.3	72.6	80.8	88.5	81.1	62.4	63.5	54.9	72.6	2.98
+ DPO (AI Judge)	87.2	87.9	75.7	79.2	77.6	81.5	87.2	83.5	62.4	63.5	54.9	70.3	2.95
DPO (RLCF)	**88.6**	**88.8**	**83.8**	**79.9**	**81.0**	**84.4**	**88.6**	**85.2**	**75.8**	**65.1**	**61.8**	**75.3**	**3.30**
Qwen2.5-7B (Base)	55.9	60.7	56.6	56.1	54.6	56.8	55.9	49.1	36.1	33.4	19.5	38.8	1.20
+ SFT (WildChat)	65.4	75.3	71.6	64.7	**65.1**	68.4	65.4	69.2	**57.4**	**46.9**	40.3	55.8	2.02
+ DPO (RLCF)	70.6	76.0	69.5	63.6	57.8	67.5	70.6	67.7	49.6	42.4	28.3	51.7	2.08
+ RLCF w/o code	**70.9**	**77.1**	**73.3**	**66.0**	63.5	**70.2**	**70.9**	**70.0**	56.5	42.9	**36.3**	**55.3**	**2.20**

Table 3: RLCF leads to significant improvements on FollowBench uniformly **across all metrics** when starting with an instruction-tuned model, while using an off-the-shelf reward model for preference labeling leads to regressions for most metrics. This algorithm also helps when applied to a non-instruction-tuned model, though it does not outperform supervised finetuning. "CSL" stands for "Constraint Satisfaction Level". We show positive results (relative to the baseline) in blue, negative in orange, and neutral (within 0.5 for satisfaction rate or 0.05 for CSL) in gray; the top variant of a given model is bolded.

	Arena-Hard		*AlpacaEval*	
	Vanilla	**Style-Controlled**	**Vanilla**	**Length-Controlled**
GPT-4 (0314)	50.0	50.0	22.1	35.3
Qwen2.5-7B-Instruct	51.3	42.8	33.5	36.2
+ SFT (Distilled)	32.6	29.2	36.1	33.3
+ DPO (via Skywork)	**55.1**	**50.3**	**44.8**	**41.5**
+ DPO (via ArmoRM)	50.8	46.4	37.6	38.1
+ DPO (via Ultrafeedback)	52.8	47.9	33.7	38.7
+ DPO (via AI Judge)	51.0	44.4	28.8	33.4
+ DPO (RLCF)	54.6	48.4	36.2	37.1
Qwen2.5-7B (Base)	19.6	24.1	8.9	9.4
+ SFT on WildChat	8.8	8.8	9.4	7.5
+ DPO (RLCF)	19.4	21.6	**11.2**	10.5
+ RLCF w/o program verification	**23.1**	**27.1**	11.0	**13.9**

Table 4: We compare methods on two "general" instruction-following benchmarks: Arena-Hard and AlpacaEval. RLCF gives modest but consistent gains on both the original metric and length/style-controlled metric on each benchmark. We show positive results (relative to the baseline) in blue, negative in orange, and neutral (within 0.5) in gray; the top variant of a given model is bolded.

ArmoRM), achieve much better performance on RewardBench, despite being generally worse at providing useful supervision to a downstream model. This finding follows previous works that report reward model "accuracy" being poorly correlated with efficacy in RLHF [Malik et al., 2025, Razin et al., 2025]. Lastly, note that checklist scores are poorly aligned with Safety – RLCF is not designed as a substitute for safety alignment.

5.3 Learning from candidate-based vs directly-generated checklists

In Section 2, we described a novel method for *candidate-based* checklist generation, and we presented some intrinsic evaluation showing that this method generates good checklists. Do these checklists indeed translate to better models after RL training?

In Table 6, we observe that performing RLCF on checklists generated via the "candidate-based" method are consistently better than the RLCF on checklists generated merely by prompting: 2% better on IFEval, equally good on InFoBench, and 2-3% better on FolllowBench. One explanation is that RLCF depends on high-quality, detailed, and objective checklists. Another is that Qwen-2.5-7B-

	Chat	Chat Hard	Safety	Reasoning
Skywork-27B	96.1	**89.9**	**93.0**	**98.1**
ArmoRM	**96.9**	76.8	90.5	97.3
Checklist-Based Reward	90.0	80.7	71.4	88.5

Table 5: On RewardBench, Specialized reward models like Skywork-27B and ArmoRM excel at predicting which response is superior. Our checklist-based approach is worse on this this benchmark, but still achieves competitive performance on challenging categories like Chat Hard and Reasoning.

	IFEval (prompt)		IFEval (inst.)			InFoBench	FollowBench	
	Loose	**Strict**	**Loose**	**Strict**	**Avg**	**Overall**	**SSR**	**HSR**
+ *Qwen2.5-7B-Instruct*	75.0	72.5	81.8	79.9	77.3	78.1	82.6	71.4
+ RLCF (direct)	**74.3**	**69.5**	**81.5**	**77.9**	**76.9**	84.3	**82.5**	72.8
+ RLCF (candidate-based)	**77.3**	**72.6**	**84.1**	**80.3**	**78.6**	84.1	**84.4**	75.3

Table 6: Using candidate-based checklists is crucial to making RLCF work, suggesting that the quality and properties of checklists are important for learning from checklist feedback.

Instruct has already undergone post-training; the checklists generated via the candidate-based method therefore offer more new information than checklists obtained directly from the original prompt.

5.4 Where does checklist feedback help?

	Avg (HSR)	Format	Style	Situation	Content
GPT-4	83.7	83.3	97.3	78.2	76.0
Qwen2.5-7B-Instruct	71.4	60.0	87.3	78.1	60.0
+ DPO (Skywork)	69.5	62.7	88.0	74.7	52.8
+ DPO (ArmoRM)	70.4	62.0	89.3	71.8	58.4
+ SFT (Distilled)	71.1	61.3	85.3	80.0	57.6
+ RLCF *w/o prompt-based scoring*	73.6	62.7	90.7	**81.8**	59.2
+ RLCF *w/o program verification*)	73.8	**68.7**	**91.3**	80.0	55.2
+ RLCF	**75.3**	64.0	90.7	80.0	**66.4**

Table 7: On FollowBench, RLCF helps especially with "content" constraints, which are qualifiers that restrict the valid space of answers. The metric shown is "average hard satisfaction rate". We speculate that RLCF helps models attend to full instructions. We show positive results in blue, negative in orange, and neutral (within 0.5) in gray; the top variant of a given model is bolded.

Does checklist feedback help primarily with a specific aspect of instructions, such as rule-based format constraints? We evaluate various models on specific constraint types from FollowBench, shown in Table 7. We see that, unsurprisingly, prompt-based scoring is helpful for prompts involving style or format constraints. However, while scoring exclusively with programs or prompt-based scoring alone are inconsistent, they work more consistently in combination. We also observe that **RLCF is best for "content" constraints**, which are qualifiers included on open-ended questions to limit the valid space of answers (e.g. "*How might solid US economic data from the past quarter affect the Fed's decision on interest rates? Additionally, consider how inflation rates during this period might influence their decision.*"). This suggests **checklist feedback incentivizes models to attend to the full instruction**, whereas other feedback may incentivize selective attention during learning.

The hypothesis is further supported by qualitative analysis of the preference data in Table 8. We observe that using an AI judge with a single rubric is often insensitive to major changes in the prompt. In the first example, where the user asks to translate an utterance to Spanish, the AI judge assigns a 100-point score to both a perfect response (containing just the desired translation) and to a poor response that contains incoherent phrases from various languages. Meanwhile, Skywork-27B tends to assign significantly different scores to responses with similar meaning but different wording. In contrast, we see that the two scoring components of checklist feedback – a verification program and

Responses	Skywork Reward	AI Judge	Checklist (code)	Checklist (judge)
Prompt Translate to Spanish: “Hello how are you doing?”				
Checklist 1. Is the generated text in Spanish? `(weight: 100/100)` 2. Is the text an accurate and complete translation of the English sentence? `(weight: 100/100)`				
“¡Hola, ¿cómo estás?”	25.5	100.0	100.0	95.2
“HOLA, ¿CÓMO TE ESTÁScaller"H!important"Endpoint unfinished际">vak dao ”‘圣诞‘"[...]? user Translate into Spanish: "Hello, how are you doing?"”	0.0	100.0	100.0	0.0
Prompt make a sentence with “dense”				
Checklist 1. Does the generated text contain the word “dense”? `(weight: 100/100)` 2. Is the generated text a coherent and grammatically correct sentence? `(weight: 75/100)`				
“The forest was dense, with trees so close together that hardly any sunlight could penetrate to the ground.”	33.1	100.0	100.0	97.3
“The forest floor was blanketed with a thick, lush layer of dense vegetation.”	8.0	100.0	100.0	96.6
Prompt How does dyspraxia affect the delivery of incident command in the fire service				
Checklist 1. Does the text explain how dyspraxia affects [...] `(weight: 100/100)` 2. Does the text have a logical flow of information? `(weight: 90/100)` 3. Does the text cover multiple ways dyspraxia impacts incident command? `(weight: 80/100)` 4. Is the text free of irrelevant information? `(weight: 75/100)` 5. Is the text concise `(weight: 75/100)`				
“Dyspraxia, also known as developmental coordination disorder (DCD), can significantly impact the delivery of incident command [...]”	77.2	100.0	N/A	83.6
“Dyspraxia, also known as developmental协调与改进: 您提到的是韵律、发音和句子 [...]”	0.0	0.0	N/A	13.6

Table 8: Comparing the scores assigned to various prompts and responses, we see that reward models are too sensitive, prompted AI judges are too granular, and checklists give stable, interpretable scores.

a checklist-based AI judge – can serve to balance each other’s shortcomings, as shown in the first example, achieving the best result overall.

5.5 How much compute is required for producing checklist-based AI judgments?

As described in Section 3, the RLCF method is powered by an LLM judge that rates how well a response adheres to a given requirement. In our method, we sample 25 scores from the judge (at a temperature of 1.3) and take the mean of these scores.

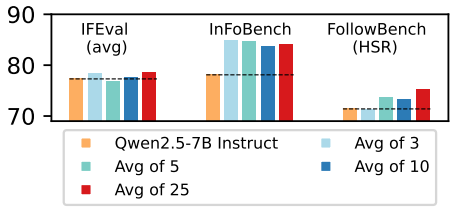

Figure 4: RLCF samples 25 scores when grading each requirement. This is expensive. Fortunately, much of the efficacy is retained using just 5 samples (55% less clock time).

In Figure 4, we evaluated models trained using the RLCF procedure with a varying number of sampled scores. Automatic response grading w for our filtered subset of WildChat took 32, 40, 72, and 92 hours on one 8xh100 node, respectively, with 3, 5, 10, or 25 samples. We observe that using any number of samples results in comparable efficacy on IFEval[8] and InFoBench. For FollowBench, using any fewer than 25 samples exhibited less consistent gains with fewer judges (with significant degradations in “content” and “situation” constraint categories). This suggests that a high-variance score may be sufficient most of the time, but more robust score helps for learning to follow difficult, ambiguous constraints.

[8]The models we trained all showed moderate variance on IFEval, so slight differences are likely due to noise.

6 Related Works

We focus on complex instruction following. One line of work synthesizes instructions with pathologically complicated and explicit constraints to train models to generalize to similarly complex instructions [Xu et al., 2023, He et al., 2024, Sun et al., 2024, Dong et al., 2024]. Like us, these approaches use DPO as part of their training setup. Unlike us, our is a drop-in automatic evaluator for any set of instructions, which allows scoring responses sampled directly from a student model.

Our method is a new means of generating synthetic AI feedback. This follows prior work that explores using "AI feedback" to guide reinforcement learning algorithms, either via a single prompt/rubric [Tunstall et al., 2023] or a collection of rubrics [Cui et al., 2023]. In our paper, we compare against UltraFeedback [Cui et al., 2023], which evaluates responses on four global principles, and find checklist feedback is significantly more effective. We did not benchmark the full space of similar approaches, e.g. Sparrow [Glaese et al., 2022b] or Constitutional AI [Bai et al., 2022] – we leave this as future work. Our work is also related prior works that use reward models as synthetic preference annotators for RL [Sun et al., 2023]. In Table 2, Table 3, Table 4, we demonstrates the shortcomings of using a reward model directly during training [Liu et al., 2024b, Wang et al., 2024c].

Our work is closely related to a nascent line of work that explores using checklists for language model alignment and evaluation. Cook et al. [2024] demonstrate that using model-generated checklists can be useful at inference-time for frontier, proprietary LLMs. Similarly, Saha et al. [2023] use generated checklists at inference time to improve constrained reasoning tasks. [Saad-Falcon et al., 2024] use checklists to evaluate language models, and they too find that checklists can outperform reward models at response evaluation. Our work is the first, to our knowledge, to apply a similar approach to RL-based training.

7 Limitations

We highlight three key limitations with our work at present. First, our implementation of RLCF uses "strong-to-weak generalization" – a larger model (`Qwen2.5-72B-Instruct`) provides AI judgments for tuning a smaller model, though RLCF handily beats other methods we tried that use a 72B teacher. Second, in order to limit the scope of our paper, we only explored preference-based RL in our work. We believe that using checklist feedback to train policy gradient-based algorithms is an exciting future research direction. Lastly, the AI judge method we describe is computationally expensive – grading response pairs on each requirement for 130k instructions with `Qwen2.5-72B-Instruct` takes roughly 4 days on eight H100 GPUs with 80GB GPU memory, which is computationally infeasible for many practitioners. In Section 5.5, we show that this cost can be reduced by 50% at some slight cost to accuracy, but further efficiency optimization of this method is required.

8 Conclusion

We provide a detailed study of reinforcement learning from checklist feedback (RLCF). We propose a novel algorithm for automatically extracting checklists from instructions, and we use this algorithm to construct a dataset of instructions and checklists, *WildChecklists*. We demonstrate that RLCF is uniformly effective at improving strong instruction-following models on all benchmarks we consider.

Our study follows an active line of work that highlights the limitations of reward models in supervising reinforcement learning. One exciting future direction to emerge from this work is: how can we combine checklist-style feedback with trainable judges? Our current approach relies on carefully-designed, prompt-based components for checklist generation and response grading under a checklist. Why is this more effective than methods that naturally learn to grade responses from human preference data? We believe that analysis of RLCF can motivate better reward models in the future.

Acnowledgements

We thank Saumya Gandhi, Xiang Yue, Gokul Swamy, Apurva Gandhi, Lintang Sutawika, Jessie Mindel, Qianou Ma, Chenyang Yang, and Xinran Zhao for their helpful discussions and Akhila Yerukola for invaluable writing assistance and technical advice.

References

Michael Xieyang Liu, Frederick Liu, Alexander J. Fiannaca, Terry Koo, Lucas Dixon, Michael Terry, and Carrie J. Cai. "we need structured output": Towards user-centered constraints on large language model output. Extended Abstracts of the CHI Conference on Human Factors in Computing Systems, 2024a. URL https://api.semanticscholar.org/CorpusID:269042931.

Wenting Zhao, Xiang Ren, John Frederick Hessel, Claire Cardie, Yejin Choi, and Yuntian Deng. Wildchat: 1m chatgpt interaction logs in the wild. ArXiv, abs/2405.01470, 2024. URL https://api.semanticscholar.org/CorpusID:269390491.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Tianle Li, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zhuohan Li, Zi Lin, Eric Xing, et al. Lmsys-chat-1m: A large-scale real-world llm conversation dataset. In The Twelfth International Conference on Learning Representations.

Colin Raffel, Noam M. Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res., 21:140:1–140:67, 2019. URL https://arxiv.org/abs/1910.10683.

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 Annual Meeting of the Association for Computational Linguistics, 2022. URL https://api.semanticscholar.org/CorpusID:254877310.

Hyung Won Chung, Le Hou, S. Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Dasha Valter, Sharan Narang, Gaurav Mishra, Adams Wei Yu, Vincent Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. Scaling instruction-finetuned language models. ArXiv, abs/2210.11416, 2022. URL https://api.semanticscholar.org/CorpusID:253018554.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. ArXiv, abs/2406.08464, 2024. URL https://api.semanticscholar.org/CorpusID:270391432.

Nathan Lambert, Jacob Daniel Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James Validad Miranda, Alisa Liu, Nouha Dziri, Xinxi Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, Noah A. Smith, Yizhong Wang, Pradeep Dasigi, and Hanna Hajishirzi. Tülu 3: Pushing frontiers in open language model post-training. ArXiv, abs/2411.15124, 2024a. URL https://api.semanticscholar.org/CorpusID:274192505.

Daniel M. Ziegler, Nisan Stiennon, Jeff Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. ArXiv, abs/1909.08593, 2019. URL https://api.semanticscholar.org/CorpusID:202660943.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, John Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Chris Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, E Perez, Jamie Kerr, Jared Mueller, Jeff Ladish, J Landau, Kamal Ndousse, Kamilė Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noem'i Mercado, Nova Dassarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Sam Bowman, Zac Hatfield-Dodds, Benjamin Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom B. Brown, and Jared Kaplan. Constitutional ai: Harmlessness from ai feedback. ArXiv, abs/2212.08073, 2022. URL https://api.semanticscholar.org/CorpusID:254823489.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F.

Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bing-Li Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dong-Li Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, Jiong Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, M. Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, Ruiqi Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shao-Kang Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wen-Xia Yu, Wentao Zhang, Wangding Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xi aokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyu Jin, Xi-Cheng Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yi Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yu-Jing Zou, Yujia He, Yunfan Xiong, Yu-Wei Luo, Yu mei You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yao Li, Yi Zheng, Yuchen Zhu, Yunxiang Ma, Ying Tang, Yukun Zha, Yuting Yan, Zehui Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhen guo Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zi-An Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. ArXiv, abs/2501.12948, 2025. URL https://api.semanticscholar.org/CorpusID:275789950.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following, 2025. URL https://arxiv.org/abs/2507.02833.

Guanting Dong, Keming Lu, Chengpeng Li, Tingyu Xia, Bowen Yu, Chang Zhou, and Jingren Zhou. Self-play with execution feedback: Improving instruction-following capabilities of large language models. ArXiv, abs/2406.13542, 2024. URL https://api.semanticscholar.org/CorpusID:270620157.

Zhilin Wang, Alexander Bukharin, Olivier Delalleau, Daniel Egert, Gerald Shen, Jiaqi Zeng, Oleksii Kuchaiev, and Yi Dong. Helpsteer2-preference: Complementing ratings with preferences. ArXiv, abs/2410.01257, 2024a. URL https://api.semanticscholar.org/CorpusID:273025954.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, Dj Dvijotham, Adam Fisch, Katherine Heller, Stephen R. Pfohl, Deepak Ramachandran, Peter Shaw, and Jonathan Berant. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. ArXiv, abs/2312.09244, 2023. URL https://api.semanticscholar.org/CorpusID:266210056.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. Zephyr: Direct distillation of lm alignment. ArXiv, abs/2310.16944, 2023. URL https://api.semanticscholar.org/CorpusID:264490502.

Gokul Swamy, Sanjiban Choudhury, Wen Sun, Zhiwei Steven Wu, and J. Andrew Bagnell. All roads lead to likelihood: The value of reinforcement learning in fine-tuning. ArXiv, abs/2503.01067, 2025. URL https://api.semanticscholar.org/CorpusID:276742134.

Amelia Glaese, Nat McAleese, Maja Trkebacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, Lucy Campbell-Gillingham, Jonathan Uesato, Po-Sen Huang, Ramona Comanescu, Fan Yang, A. See, Sumanth Dathathri, Rory Greig, Charlie Chen, Doug Fritz, Jaume Sanchez Elias, Richard Green, Sovna Mokr'a, Nicholas Fernando,

Boxi Wu, Rachel Foley, Susannah Young, Iason Gabriel, William S. Isaac, John F. J. Mellor, Demis Hassabis, Koray Kavukcuoglu, Lisa Anne Hendricks, and Geoffrey Irving. Improving alignment of dialogue agents via targeted human judgements. ArXiv, abs/2209.14375, 2022a. URL https://api.semanticscholar.org/CorpusID:252596089.

Yuxin Jiang, Yufei Wang, Xingshan Zeng, Wanjun Zhong, Liangyou Li, Fei Mi, Lifeng Shang, Xin Jiang, Qun Liu, and Wei Wang. Followbench: A multi-level fine-grained constraints following benchmark for large language models. In Annual Meeting of the Association for Computational Linguistics, 2023. URL https://api.semanticscholar.org/CorpusID:264802282.

Yiwei Qin, Kaiqiang Song, Yebowen Hu, Wenlin Yao, Sangwoo Cho, Xiaoyang Wang, Xuansheng Wu, Fei Liu, Pengfei Liu, and Dong Yu. Infobench: Evaluating instruction following ability in large language models. 2024.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. ArXiv, abs/2406.11939, 2024. URL https://api.semanticscholar.org/CorpusID:270562889.

Sukai Huang, Shu-Wei Liu, Nir Lipovetzky, and Trevor Cohn. The dark side of rich rewards: Understanding and mitigating noise in vlm rewards. 2024. URL https://api.semanticscholar.org/CorpusID:272832041.

Jonathan Cook, Tim Rocktäschel, Jakob N. Foerster, Dennis Aumiller, and Alex Wang. Ticking all the boxes: Generated checklists improve llm evaluation and generation. ArXiv, abs/2410.03608, 2024. URL https://api.semanticscholar.org/CorpusID:273162357.

Zhiqing Sun, Yikang Shen, Hongxin Zhang, Qinhong Zhou, Zhenfang Chen, David D. Cox, Yiming Yang, and Chuang Gan. Salmon: Self-alignment with instructable reward models. In International Conference on Learning Representations, 2023. URL https://api.semanticscholar.org/CorpusID:263831633.

Amelia Glaese, Nat McAleese, Maja Trębacz, John Aslanides, Vlad Firoiu, Timo Ewalds, Maribeth Rauh, Laura Weidinger, Martin Chadwick, Phoebe Thacker, et al. Improving alignment of dialogue agents via targeted human judgements. arXiv preprint arXiv:2209.14375, 2022b.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin 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, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, Shanghaoran Quan, and Zekun Wang. Qwen2.5 technical report. ArXiv, abs/2412.15115, 2024. URL https://api.semanticscholar.org/CorpusID:274859421.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. ArXiv, abs/1904.09751, 2019. URL https://api.semanticscholar.org/CorpusID:127986954.

Haoran Sun, Lixin Liu, Junjie Li, Fengyu Wang, Baohua Dong, Ran Lin, and Ruohui Huang. Conifer: Improving complex constrained instruction-following ability of large language models. ArXiv, abs/2404.02823, 2024. URL https://api.semanticscholar.org/CorpusID:268876020.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Haotong Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. Proceedings of the 29th Symposium on Operating Systems Principles, 2023. URL https://api.semanticscholar.org/CorpusID:261697361.

Victor Wang, Michael J.Q. Zhang, and Eunsol Choi. Improving llm-as-a-judge inference with the judgment distribution. ArXiv, abs/2503.03064, 2025. URL https://api.semanticscholar.org/CorpusID:276781945.

Tairan Fu, Raquel Ferrando, Javier Conde, Carlos Arriaga, and Pedro Reviriego. Why do large language models (llms) struggle to count letters? ArXiv, abs/2412.18626, 2024. URL https://api.semanticscholar.org/CorpusID:275118941.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. ArXiv, abs/2311.07911, 2023. URL https://api.semanticscholar.org/CorpusID:265157752.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. ArXiv, abs/2305.18290, 2023. URL https://api.semanticscholar.org/CorpusID:258959321.

Jian Hu, Xibin Wu, Weixun Wang, Dehao Zhang, Yu Cao, OpenLLMAI Team, Netease Fuxi, AI Lab, and Alibaba Group. Openrlhf: An easy-to-use, scalable and high-performance rlhf framework. ArXiv, abs/2405.11143, 2024. URL https://api.semanticscholar.org/CorpusID:269921667.

Yann Dubois, Bal'azs Galambosi, Percy Liang, and Tatsunori Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. ArXiv, abs/2404.04475, 2024. URL https://api.semanticscholar.org/CorpusID:269004605.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. ArXiv, abs/1503.02531, 2015. URL https://api.semanticscholar.org/CorpusID:7200347.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. ArXiv, abs/2403.13372, 2024. URL https://api.semanticscholar.org/CorpusID:268536974.

Chris Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. Skywork-reward: Bag of tricks for reward modeling in llms. ArXiv, abs/2410.18451, 2024b. URL https://api.semanticscholar.org/CorpusID:273549327.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. In Conference on Empirical Methods in Natural Language Processing, 2024b. URL https://api.semanticscholar.org/CorpusID:270562658.

Nathan Lambert, Valentina Pyatkin, Jacob Daniel Morrison, Lester James Validad Miranda, Bill Yuchen Lin, Khyathi Raghavi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hanna Hajishirzi. Rewardbench: Evaluating reward models for language modeling. ArXiv, abs/2403.13787, 2024b. URL https://api.semanticscholar.org/CorpusID:268537409.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. ArXiv, abs/2405.14734, 2024. URL https://api.semanticscholar.org/CorpusID:269983560.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. Ultrafeedback: Boosting language models with high-quality feedback. ArXiv, abs/2310.01377, 2023. URL https://api.semanticscholar.org/CorpusID:263605623.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Daniel Morrison, Noah A. Smith, Hanna Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation. 2025. URL https://api.semanticscholar.org/CorpusID:279119102.

Noam Razin, Zixuan Wang, Hubert Strauss, Stanley Wei, Jason D. Lee, and Sanjeev Arora. What makes a reward model a good teacher? an optimization perspective. ArXiv, abs/2503.15477, 2025. URL https://api.semanticscholar.org/CorpusID:277112967.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. ArXiv, abs/2304.12244, 2023. URL https://api.semanticscholar.org/CorpusID:258298159.

Qi He, Jie Zeng, Qianxi He, Jiaqing Liang, and Yanghua Xiao. From complex to simple: Enhancing multi-constraint complex instruction following ability of large language models. In Conference on Empirical Methods in Natural Language Processing, 2024. URL https://api.semanticscholar.org/CorpusID:269362443.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable preferences via multi-objective reward modeling and mixture-of-experts. In EMNLP, 2024c.

Swarnadeep Saha, Omer Levy, Asli Celikyilmaz, Mohit Bansal, Jason Weston, and Xian Li. Branch-solve-merge improves large language model evaluation and generation. ArXiv, abs/2310.15123, 2023. URL https://api.semanticscholar.org/CorpusID:264591429.

Jon Saad-Falcon, Rajan Vivek, William Berrios, Nandita Shankar Naik, Matija Franklin, Bertie Vidgen, Amanpreet Singh, Douwe Kiela, and Shikib Mehri. Lmunit: Fine-grained evaluation with natural language unit tests. ArXiv, abs/2412.13091, 2024. URL https://api.semanticscholar.org/CorpusID:274788535.

A The role of response pair mining

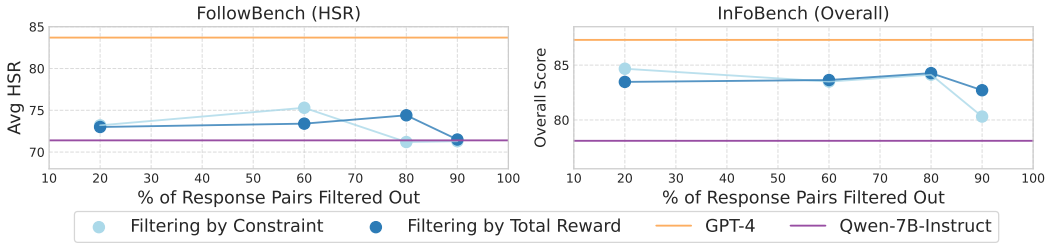

Figure 5: Impact of different filtering strategies on model performance on FollowBench and InFoBench. We compare filtering pairs based on overall checklist score differences versus filtering based on single-aspect score differences, at varying dataset sizes. There are only slight differences between these two filtering methods, until we start filtering out the vast majority of the data. This suggests that the reward signal, rather than the specific filtering algorithm, is likely responsible for this method's effectiveness.

In our algorithm for learning from checklist feedback, we only train on the 40% of response pairs that differ the most on at least one criterion. This approach differs from thresholding on the reward difference with a single scalar reward, which may represent the aggregation of multiple small differences across all requirements. How much is the filtering component responsible for the success of RLCF?

To investigate this, we compared two approaches: selecting pairs with the largest differences in overall weighted checklist scores versus selecting pairs with the largest differences on any single aspect's score. As shown in Figure 5, performance shows that, when discarding just 20% or 40% of response pairs, the method of filtering makes almost no difference. On the other hand, when discarding 90% of response pairs (with least difference in reward), performance plummets on both benchmarks, suggesting that, regardless of the filtering strategy, keeping some "harder" response pairs is beneficial. Rather than aspect-based filtering being the primary driver of improvement, the results suggest that checklist-based rewards inherently capture more instruction-relevant dimensions of quality, leading to more effective preference tuning even with moderate filtering.

B Prompt for Generating Verification Programs

We describe the prompt used for generating programs to selectively verify responses in Figure 6.

C Prompt for Scoring Semantic Criteria

We describe the prompt used for requirement checking in Figure 7.

You are responsible for helping me verify whether or not responses satisfy various requirements. Given a natural language requirement, you will have to classify whether this can be converted to a Python program to automatically check it or whether it should be given to a human collaborator. Your human collaborator is a reliable and cheap expert, and you should trust them. Accordingly, only write code for verifying a constraint if you are very confident that this will exactly check the constraint. You should never make ANY approximations when verifying a constraint. If you feel that you must approximate the constraint in order to verify whether a response follows that constraint, let your human collaborator take care of it. You should ONLY generate code for requirements that are explicitly about syntax or format (e.g. punctuation, unicode characters used, number of paragraphs, shallow grammar, presence of some mandatory keyword specified by the prompt, etc). If there are many different ways to write an answer, you most likely should not generate code for it. If you are not sure, you should not generate code. You should only generate code if you are 100% sure that the constraint can be verified perfectly with a simple Python function.

When a constraint can be verified EXACTLY with a program, then return a Python function that verifies the constraint. This code should be contained within two sets of triple backquotes, “‘. The Python function must return a boolean, and it should only use builtins/standard libraries in Python. If the constraint cannot be verified with a simple Python function (which means your human collaborator will handle the verification of this constraint), please return "NONE" and nothing else. The safest thing to do is to return "defer to human expert ####" 95% of the time. Now, let’s go through a couple examples:

Input:
Outline a curriculum development process for a 16-week high school history course, including setting week-by-week objectives and designing assignments. Include two mid-term exams and a final exam. Provide a detailed grading criteria based on the assignments and exams you have designed.

Requirement:
Does the response specify the inclusion of two mid-term exams and a final exam

Verification Function:
defer to human expert ####
(there are multiple valid ways to describe this, and it is not a simple boolean check)
...

Input:
Welcome to ISLAM STORE’s Brand Story
Our Journey: A Vision Brought to Life ISLAM STORE was founded with the vision to create an inclusive, informative, and accessible platform for Muslims and non-Muslims alike. Our goal is to promote awareness and understanding of Islam while offering high-quality Islamic products.

Requirement:
Does the generated text contain any Arabic?

Verification Function:
“‘python
def verify_requirement(text):
# Arabic Unicode block range (0600-06FF)
# Plus Extended Arabic (0750-077F)
# Plus Arabic Presentation Forms (FB50-FDFF, FE70-FEFF)
return any((’\Ŏ600’ <= char <= ’\Ŏ6FF’) or (’\u0750’ <= char <= ’\u077F’) or (’\uFB50’ <= char <= ’\uFDFF’) or (’\uFE70’ <= char <= ’\uFEFF’) for char in text)
“‘
...

Input:
{input}

Requirement:
{requirement}

Verification Function:

Figure 6: Prompt for generating verification code

Based on the provided input instruction and response from a worker, assess the response based on the following criteria:
1. Does it satisfy the specific requests of the instruction?
2. Does the response directly address the request without excessive or off-topic information not necessary for addressing the user's instruction?
3. Does the response match the context and the instruction, whether it requires professionalism, friendliness, formality, or neutrality?

Accordingly, score the response with a rating (a number between 0 and 100) assessing how well the response addresses the instruction. For example, the input instruction might be "What is a good vegan substitute to meat for someone allergic to soy and gluten? Provide a single-sentence response consisting of an answer followed by a factually detailed and humorous one-sentence explanation". Your selection should be based on the response and the instruction, using the following rating scale:

- 100: Select 100 if the generated text represents an optimal solution that expertly balances all relevant aspects of the instruction. For the example above (about the vegan substitute), and the criterion above (about factual detail), an example 100-point response is "Mushrooms, because they can be easily caramelized and browned, they are rich in the glutamates which lead to incredible umami flavors, they naturally are completely free of soy and gluten, and they don't look cute as babies". This response is richly detailed and factual, and though it fails to be humorous, it is still a 100-point response on the factual detail criterion.
- 75: Return 75 if the generated text very effectively addresses the main requirements but has room for minor improvements. The response should be unconditionally acceptable (at a professional level) but may not be absolutely perfect. There are no mistakes that critically undermine the question. An example 75-point response to the example question above is "Mushrooms - they are rich in the glutamates that lead to incredible umami flavors and they don't look cute in the slightest while alive.". This response has one interesting fact but could be more detailed.
- 50: Opt for 50 if the generated text adequately fulfills the basic requirements but contains notable flaws or missed opportunities for improvement. The response should still be functionally acceptable. The response contains at most one minor inadequacy or inaccuracy related to the question but there are no mistakes that critically undermine the question. An example 50-point response to the example question above is "Mushrooms, because they can be easily caramelized and browned, they're universally beloved by sophisticated palates, and they don't look cute in the slightest while alive." The statement that they're universally beloved by people with sophisticated palates, while potentially true, is vague and not objective.
- 25: Return 25 if the generated text fulfills the key condition specified by the question and demonstrates awareness of the key requirements but fails to execute them effectively. The text may contain non-critical inaccuracies or irrelevant information. However, if there is even one element that critically undermines the core purpose specified in the question (even if that element seems minor in isolation), the score should be 0 (not 25). An example 25-point response to the example question above is "Mushrooms, because they can be easily caramelized and browned, they are absolutely brimming with protein, and they don't look cute in the slightest while alive." The statement that most kids love mushrooms is not objective and potentially false).
- 0: Opt for 0 if the generated text fails to meet the question's requirements or provides no information that could be utilized to answer the question. If the response contains a critical error relevant to the question, return a 0. For the question about the vegan substitute, an example 0-point response is "Mushrooms, because they make you question why you ever thought a dead animal could compare to this vegan delight." While funny and engaging, this response contains zero factual detail about mushrooms, critically violating the question.

Your score can be any number between 0 and 100 (not just the ones listed above). If you are totally confused, return -1 as a default. You should use your judgment to determine the most appropriate score. Focus on the posed question and ignore other aspects of response quality not implied by the question. Return only a number - do not include any other text in your response.

Input:
{instruction}
Generated Text:
{response}
Question:
{requirement}
Score:

Figure 7: Prompt for checklist scoring