REBUTTAL TO REVIEWER AC2P

Summary. We sincerely thank the reviewer for the appreciation of **eva** and the constructive feedback. We have made every effort to throughly address the concerns. Specifically, we have:

- added experiments on implementing **different evolving methods** and discussed relevant strengths and weaknesses in § D.1;
- added **visualization on the learning curriculum** in § E;
- provided detailed discussion on **scaling up eva** with million-level data on larger-scale seed sets and/or inference-time scaling for synthesizing prompts.

> **Q1 (Choice of the Evolving Method)**: Could you explain more about the particular choice of evolution algorithm used in your implementation of eva and different potential strengths and weaknesses related to this choice?

TL;DR: We use EvolInstruct (Xu et al., 2023a) as it is among the most easy-to-implement methods. We added new experiments w/ other methods, including SelfInstruct (Wang et al., 2022), EvolQuality and EvolComplexity (Liu et al., 2023b), and show that **eva** remains to be effective in § D.1.

Answer: As an addition to Table 1, we have experimented with three different `evolve`(·) methods:

- **SelfInstruct** (Wang et al., 2022): Given seed prompts, variations are created based on criteria such as verb diversity and style blending (mixing interrogative and imperative styles). Unlike EvolInstruct (Xu et al., 2023a), which generates prompt variations sequentially, this approach generates independently. We follow the one-shot implementation in `self_instruct.py` of `distilabel==1.4.1` and modified the instruction on conciseness so that newly generated prompts have similar lengths compared to the seed prompts.
- **EvolQuality** and **EvolComplexity** (Liu et al., 2023b): The two methods use the same evolutionary approach (*i.e.*, sequential generation), but with slightly different meta-instructions for prompt generation, where EvolQuality asks to improve the quality (*i.e.*, helpfulness, relevance, etc) of the seed prompt and EvolComplexity asks to improve the complexity (*i.e.*, increased reasoning steps, etc) of the seed prompt. We follow the implementation in `evol_quality/utils.py` and `evol_complexity/utils.py` of `distilabel==1.4.1`.

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) / Metric (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO	51.6	651
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `EvolInstruct`)	60.1	733
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `EvolQuality`)	58.7	721
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `EvolComplexity`)	**60.6**	749
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `SelfInstruct`)	57.2	725

Table 23: Results of using different evolving methods.

eva is effective under different evolving methods. As shown in Table 10, our method brings strong performance gain without training with additional human prompts. Among the experimented methods, we find EvolComplexity shows better results.

We believe the main strength of such method is its **simplicity**. Viewing the evolving process as $\mathbf{x}' \leftarrow p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}, \texttt{meta\_prompt})$, one can easily tune the meta prompt in natural language for improved performance. However, such simplicity comes at a price: (i) the main weakness is that the default method does not take **environmental feedback** into account (*e.g.*, rewards received, verbal critique on responses, etc) and relies on the pre-defined meta prompt, thus the evolving may be less directional; we encourage practitioners to consider incorporating richer feedback during evolving (one way to formulate this is by generative optimization (Yuksekgonul et al., 2024; Cheng et al., 2024; Nie et al., 2024)); (ii) another weakness is that existing method is single-shot (*i.e.*, we evolve based on a single $\mathbf{x}$ each time), thus the **diversity** of the generation may be limited – we anticipate future works improving this with multi-shot evolving by graph-based sampling or including diversity-related rewards in generation. In this regard, the evolving process can be viewed as $\{\mathbf{x}'\}_{i=1}^{N} \leftarrow p_{\boldsymbol{\theta}}(\cdot \mid \{\mathbf{x}\}_{i=1}^{M}, \texttt{meta\_prompt}, \texttt{env\_feedback})$.

> **Q2 & Q3 (Empirical Evidence on Learning Progress and Curriculum)**: Do you see empirical evidence of your intuition about learning progress discussed in section 3.4? It seems like some of these claims are directly testable. Could you visualize the curriculum learned in your experiments with eva? It would be very nice to get an intuition for why performance improves and what the heuristic prioritizes over time.

Answer: We thank the reviewer for the constructive suggestions on empirically validating the intuition. We have revised the manuscript with additional visualization on potential curriculum learned in § E. In general, we observe the creator prioritizes learning in math and coding, which brings gradual improvement on benchmark performance on relevant categories over iterations. We have attached the bar plot and radar figure here for the reviewer's reference:

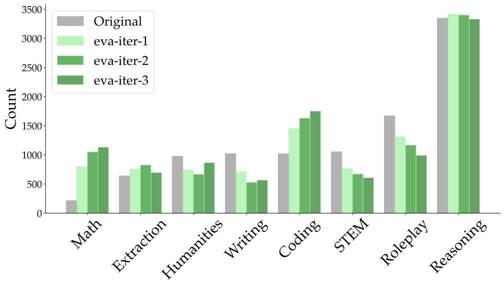

Figure 11: **Training distributions.** The prompt distribution of Table 11 for evolved prompts by zero-shot classification. **`eva`** creates a curriculum that prioritizes math / coding prompts.

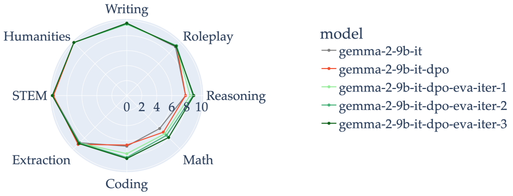

Figure 12: **Benchmark performance.** The radar figure for ratings on MT-Bench (Zheng et al., 2023), where each category contains ten problems. **`eva`** prioritizes and gradually improves on coding, math and reasoning over iterations, implicitly reflecting a learned curriculum.

We further evaluate the complexity and quality of the prompt distributions. As in Table 24, there is a gradual improvement of prompt complexity and quality over iterations with **`eva`**.

Prompt Set (↓) / **Metric** (→)	**Complexity** (1-5)	**Quality** (1-5)
UltraFeedback (seed)	2.90	3.18
UltraFeedback-**`eva`**-Iter-1	3.84	3.59
UltraFeedback-**`eva`**-Iter-2	3.92	3.63
UltraFeedback-**`eva`**-Iter-3	**3.98**	**3.73**

Table 24: **`eva`** improves prompt quality and complexity.

Implementation notes: We sample 10K prompts per iteration, and use the below prompts modified from Liu et al. (2023b) for the complexity and quality evaluation, with `gemini-1.5-flash` as the generative scorer:

```
Rank the following questions according to their quality. Your evaluation should consider the following
↪  factors: Helpfulness, Relevance, Accuracy, Depth, Creativity, and Level of detail.
Score each response from 1 to 5: 1: Poor quality, 2: Below average, 3: Average, 4: Good, 5: Excellent.
```

```
Ranking the following questions according to their difficulty and complexity. Use a fixed scoring system: 1:
↪  Very simple, 2: Simple, 3: Moderate,  4: Difficult, 5: Very difficult
```

> **Q4 (Scaling)**: When discussing future directions, the authors write further scaling up w/ million-level data. Can you clarify what this means? Seems like some important context is missing?

TL;DR: We consider (i) applying **`eva`** when the seed set contains million-level or more prompts; or (ii) using **`eva`** to robustly generate million-level or more prompts when the seed set is limited.

Answer: (i) The current paper uses the UltraFeedback (Cui et al., 2023) as the seed prompt set, which is a ten-thousands level dataset; in training practically useful large language models (Brown et al., 2020; Team et al., 2024a; Singh et al., 2023), the seed prompt set is usually much larger than such a level. We believe it is an interesting direction to explore the data scaling properties of **`eva`** on larger seed prompt sets, in combination with our on-policy variants. (ii) On the other hand, when the seed prompt set contains only limited data (this issue is particularly concerning in hard reasoning tasks like math (Yang et al., 2024)), can we still follow the data generating curriculum and synthesize million-level prompts/problems to help training, and how to robustly verify the generated prompts/problems? Beyond training-time scaling, there is also a recent trend in inference-time scaling (Snell et al., 2024), however these works only consider scaling in the $\mathcal{Y}$ space, not the $\mathcal{X}$ or the joint $(\mathcal{X}, \mathcal{Y})$ space. We believe synthetically scaling up the joint $(\mathcal{X}, \mathcal{Y})$ space to a much more larger magnitude under **`eva`**'s game-theoretic design presents a new direction worth investigating.

> **Final remarks.** We thank the reviewer once again for spending time providing constructive feedback that helps improve **`eva`**. Please let us know if there is any other concerns or questions, and we are more than grateful to have the opportunity to learn from and discuss with you.

REBUTTAL TO REVIEWER ZXTK

Summary. We sincerely thank the reviewer for all the constructive feedback helping improving the **`eva`** method. In response, we have provided:

- **experiments on more iterations** in § D.2;
- **extended discussions on the regret objective and the proxy** in § G;
- **evidence on distinction of advantage-based metrics and variance-based ones** in § F;
- **revised illustration of the method** in § 3, and **evidence on evolving prompt distributions** (to answer it cannot "cheat by selecting easier prompts") in § E.

We appreciate the chance to address the reviewer's comments and have made every effort to thoroughly address the concerns and revise our manuscript accordingly. We hope that these revisions meet the reviewer's expectations and would be grateful if the reviewer could kindly consider revise the score.

> **W1 (Running for More Iterations)**: The number of iterations in the main results is 2, with only one EVA step in each experiment, which is a little different from what the demonstration in Figure 3 shows. If the **`eva`** step is performed multiple times, would the results be better or worse? What is performance like when you access all data in UltraFeedback?

TL;DR: We added experimental result on running more iterations with more data, and **`eva`** remains to be effective. We have added § D.2 in the manuscript to incorporate the reviewer's suggestion.

Rebuttal: As an addition to § 4.2.4, we have experimented with the following settings:

- 10K prompts per iteration with 3 iterations.
- 20K prompts per iteration with 3 iterations (*i.e.*, all seed prompts are used).
- 60K prompts per iteration with 2 iterations (*i.e.*, all seed prompts are used).

Due to time constraints, we did not perform an extensive hyper-parameter search; however, we believe the results presented below sufficiently demonstrate the performance gains achieved by **`eva`**.

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO (10k)	51.6	651
$\boldsymbol{\theta}_{1\to2}$: DPO (10k)	59.8	718
$\boldsymbol{\theta}_{2\to3}$: DPO (10k)	61.2	802
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **`eva`** (10k)	60.1	733
$\boldsymbol{\theta}_{\tilde{1}\to\tilde{2}}$: + **`eva`** (10k)	62.0	787
$\boldsymbol{\theta}_{\tilde{2}\to\tilde{3}}$: + **`eva`** (10k)	62.2	774

Table 25: Results of using 10k prompts per iteration.

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO (20k)	53.2	625
$\boldsymbol{\theta}_{1\to2}$: DPO (20k)	47.0	601
$\boldsymbol{\theta}_{2\to3}$: DPO (20k)	46.8	564
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **`eva`** (20k)	59.5	826
$\boldsymbol{\theta}_{\tilde{1}\to\tilde{2}}$: + **`eva`** (20k)	60.0	817
$\boldsymbol{\theta}_{\tilde{2}\to\tilde{3}}$: + **`eva`** (20k)	61.4	791

Table 26: Results of using 20k prompts per iteration.

Model Family (→)	Gemma-2-9B-it	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to 1}$: DPO (60k)	58.9	717
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + `eva` (60k)	59.6	725
$\boldsymbol{\theta}_{\tilde{1}\to\tilde{1}'}$: + `eva` (60k)	61.9	792

Table 27: Results of using 60k prompts per iteration.

`eva` can bring robust gains with multiple iterations. As shown in Table 25, 26, and 27, our method presents persistent performance gain over iterations, and concretely surpasses the performance by default DPO training with true human prompts.

However, there exist diminishing marginal gains in iterative off-policy training. We ground `eva` in the iterative (off-policy) preference alignment paradigm due to its efficiency and ease of integration. However, such paradigms inherently face diminishing returns, where performance gains decrease with successive iterations, as previously observed in (Wu et al., 2024; Setlur et al., 2024; Yuan et al., 2024; Nikishin et al., 2022). While the generative data schedule in `eva` mitigates these challenges and extends beyond default training with human prompts (see also §4.2.4), the gains can weaken over iterations. We summarize potential reasons as: (i) the **off-policy signal decay** – as the number of examples increases, signals from the off-policy data become weaker due to distributional shift; (ii) the **loss of plasticity**, where the agent's ability to learn good policies decreases in continuing training with more iterations (Nikishin et al., 2022); (iii) the **ability of the solver** – as we evolve more harder prompts, it is harder for the solver to produce preferred response (thus more explicit reasoning techniques may be needed); (iv) the **ability of the reward model** to correctly provide reward signals to responses and thus informativeness signals to prompts, as there may exists distributional mismatch.

Thus, we envision future work to build on `eva` by: (i) exploring its integration with **on-policy RLHF** (*e.g.*, instead of evolving prompts in iterations, one may evolve in batches); (ii) **enhancing solver capabilities**, such as sampling more responses during inference or leveraging meta-instructions to guide deeper reasoning; (iii) online training of RM to co-evolve with the creator and the solver.

Bonus experiments on adding rewriter in the solver step. This is beyond the current paper, and we present the basic idea here for practitioners to build upon `eva`. The motivation comes from the hypotheses derived from § D.2: as the prompts gets harder by evolving, there may be greater demands on the solver's capabilities *compared to earlier iterations*. As such, the solver may not be naively treated the same. One may address this by either inference-time scaling on responses or introducing meta-instructions to explicitly enhance the solver's reasoning.

We hereby design a proof-of-concept experiment *w.r.t* the latter by adding **rewriter** in `eva`'s solver step. Previously, as in Algo. 1 and § 3.3.2, for each prompt $\mathbf{x}$, we generate multiple responses, and choose the best as $\mathbf{y}_+$ and the worst as $\mathbf{y}_-$ for preference optimization. Now, we add one more rewriting step that attempts to enhance $\mathbf{y}_+$ to be $\mathbf{y}'_+$, by applying a rewriting instruction (Liu et al., 2023b) that asks the solver to alter $\mathbf{y}_+$ with imporved helpfulness, relevance, reasoning depths, creativity and details while keeping the similar length. We then train with $(\mathbf{x}, \mathbf{y}'_+, \mathbf{y}_-)$ for preference optimization. Table 14 shows that adding the rewriter yields concrete performance gains over the default method, while keeping training budgets and only slightly increasing response generation cost.

Model Family (→)	Gemma-2-9B-it	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to 1}$: DPO	51.6	651
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + `eva`	**60.1**	733
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + `eva` with `rewriter`	**61.9**	741

Table 28: Results of adding `rewriter` in the `solver` step.

> **W2 (Connection in Minimax Regret and The Algorithm)**: The connection between the minimax regret objective and the algorithm is a somehow vague. The regret concerns the performance gap with the optimal policy. It's not reflected by the informativeness proxy.

TL;DR: We have added § G to address the reviewer's concern in detail. (i) In the current algorithm, the solver explicitly minimizes the regret by plug-in preference optimization algorithms (*e.g.*, DPO), while the creator implicitly maximizes the regret by first finding high-regret prompts and generate variations as new prompt distributions for training. (ii) The informativeness proxy is an advantage-based estimate of the regret; similar variants have been used in prior literature like Jiang et al. (2021b); Parker-Holder et al. (2022); as the policy optimizes, the proxy can approximate the true regret better.

Rebuttal: For the rebuttal to be self-contained, we extract contents from § G.3 here. We feel § G offers a better overview – it would be great if you could take some time to review § G when feasible.

KL-regularized regret. In the RLHF setting at fixed prompt distribution, the objective is:

$$\max_{\pi_{\boldsymbol{\theta}}} \mathbb{E}_{\mathbf{x}\sim\pi_\phi(\cdot),\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\Big[\beta\cdot\pi_\phi(\cdot)_{\text{KL}}\Big[\,\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}) \parallel \pi_{\text{SFT}}(\mathbf{y}\mid\mathbf{x})\,\Big]\Big].$$

The optimal policy of the above KL-constrained objective is:

$$\pi^{\star}_{\text{KL}}(\mathbf{y}\mid\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\pi_{\text{SFT}}(\mathbf{y}\mid\mathbf{x})\exp\left(\frac{1}{\beta}\cdot r(\mathbf{x},\mathbf{y})\right),$$

where the partition function is defined as $Z(\mathbf{x}) = \sum_{\mathbf{y}} \pi_{\text{ref}}(\mathbf{y}\mid\mathbf{x})\exp\left(\frac{1}{\beta}r(\mathbf{x},\mathbf{y})\right)$.

We can now formally define the *regret* with regard to $\pi^{\star}_{\text{KL}}(\cdot\mid\mathbf{x})$ as:

$$\text{Regret}_{\text{KL}}(\mathbf{x},\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \mathbb{E}_{\mathbf{y}\sim\pi^{\star}_{\text{KL}}(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big]. \tag{18}$$

Regret Minimization for the Solver. It is rather straightforward/trivial to understand the objective of the solver $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ as regret minimization, since the goal is to align the policy $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ with the KL-optimal solution $\pi^{\star}_{\text{KL}}(\cdot\mid\mathbf{x})$, which directly minimizes the KL-regularized regret by design. This formulation allows flexibility in the plug-in preference optimization algorithms for the solver's step in Algorithm 1, and ensures *the alignment problem is well-defined*. In practice, we use Direct Preference Optimization (DPO) and its variants, which approximate the KL-optimal solution by iteratively adjusting $\pi_{\boldsymbol{\theta}}$ to reflect preference differences.

Regret Maximization for the Creator. As discussed previously, while it is often trivial for the solver to minimize the regret through direct policy optimization, the optimal policy remains unknown during the optimization process, thus we cannot directly calculate the regret – we must approximate it when using it as the utility for the creator. Similarly to heuristics proposed by prior works (Jiang et al., 2021b;a; Parker-Holder et al., 2022), we use the advantage-based estimate:

$$|\widehat{\text{Regret}}(\mathbf{x},\pi_{\boldsymbol{\theta}})| \leftarrow \text{info}_{\boldsymbol{\theta}}(\mathbf{x}) := r(\mathbf{x},\mathbf{y}_+) - r(\mathbf{x},\mathbf{y}_{\text{baseline}}), \tag{19}$$

where

$$\mathbf{y}_+ := \arg\max_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y}), \tag{20}$$

$$\mathbf{y}_{\text{baseline}} := \arg\min_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y}) \text{ or } \mathbf{y}_{\text{baseline}} := \text{avg}_{\mathbf{y}_i}\, r(\mathbf{x},\mathbf{y}), \tag{21}$$

and $\{\mathbf{y}_i\}_{i=1}$ is a set of responses sampled from $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ and $r(\cdot,\cdot)$ is the reward oracle. We use $\arg\min_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y})$ by default due to its simplicity and efficiency (see also § 3.4 for additional interpretation) and consistent strong empirical gains we observed in vast experiments. As the policy optmizes, the proxy should approximate the regret better. We leave exploration of other informativeness proxy designs in **`eva`** to future work.

For new environment generation, as illustrated in § 3.3.1, we start from the seed prompt set, choose those high-regret prompts and generate variations upon them by `EvolInstruct`, then mixing with a buffer of the original set to form the new training distribution at each iteration.

> **Q1 (Advantage v.s. Variance)**: The informativeness proxy seems to be similar to the variance of the rewards because they all concern the diversity of the generated responses. However, in lines 393-395, the results show using variance leads to poor performance. How to interpret this?

Rebuttal: To explain, (i) variance does not directly capture the **learning potential** in preference optimization, while advantage-based informativeness proxy is better aligned to the learning objective; (iii) we **empirically show** that variance and advantage are only **weakly correlated** thus will likely result in different sampling. We have added § F to incorporate the reviewer's suggestion.

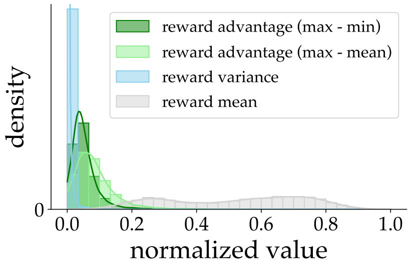

Figure 13: The probability density distributions of informativeness metrics in Table 3 – they show different patterns.

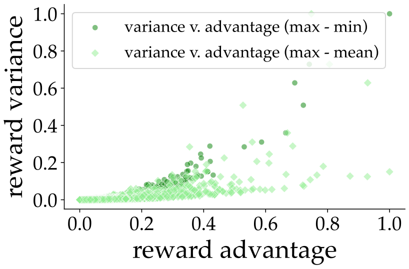

Figure 14: The correlation plot for reward advantage (ours) and reward variance – they are only *weakly* correlated.

In `eva`, we assign each prompt an informativeness value, which the creator will use as the weight to sample from the seed prompts for prompt synthesis. In § 4.2.1, we have shown that traditional methods like reward mean and reward variance are less effective as our advantage-based informativeness proxy. The intuition is simple: advantage/regret-based proxy aligns better with the preference optimization objective. We here further illustrate that they are statistically different from other choices:

- Figure 13: The distribution of informativeness values shows that reward variance is heavily concentrated at lower values, reward mean is more uniformly scattered, and reward advantage achieves a better balance, providing a broader yet also focused sampling range.
- Figure 14: The *weak correlation* between reward variance and reward advantage shows that variance *cannot* serve as a substitute for advantage as a proxy for informativeness.

We have discussed the contrastive curriculum hypothesis in § 3.4 to support using reward advantage in the sense that the induced samples tend to decrease the loss the most in the contrastive optimization. Furthermore, assuming the optimization algorithm can converge to the *more optimal* responses, neither reward mean nor variance directly capture the learning potential of such responses – one may easily construct cases with identical variance yet differ much in reward range – thus variance fails to distinguish such scenarios. By contrast, reward advantage estimate inherently captures the relative improvement towards better response, and is sensitive to differences in reward range; variants of advantage estimate are commonly used in literature, and we discuss underlying principles in § G.

> **Q2 (Open-Ended Objective)**: In Line 1399 (authors' note: should be around Line 880 - 886 in the main OpenReview PDF), why is this objective ideal? Optimizing the prompt distribution seems like cheating by selecting easier prompts for a higher reward.

TL;DR: No – the whole design (we impose the KL to the open-ended reference in principle, and its approximation by the creator through regret maximization) is to avoid cheating! We also present empirical evidence showing `eva` helps create more complex prompts.

Answer: To start with, we quote the conceptual objective below:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\pi_{\boldsymbol{\phi}}(\cdot),\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})} \Big[r(\mathbf{x},\mathbf{y}) \Big] - \beta \cdot \mathbb{D}_{\mathrm{KL}} \Big[\pi_{\phi}(\mathbf{x}) \cdot \pi_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{x}) \parallel p_{\mathrm{ref}}(\mathbf{x}) \cdot \pi_{\mathrm{SFT}}(\mathbf{y} \mid \mathbf{x}) \Big].$$

Another way to express the principled objective (with refined coefficients) is:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\pi_{\phi}(\cdot)} \Big[\underbrace{\mathbb{E}_{\mathbf{y}\sim\pi_{\theta}(\cdot\mid\mathbf{x})} \Big[r(\mathbf{x},\mathbf{y}) \Big] - \beta_1 \cdot \mathbb{D}_{\mathrm{KL}} \big[\pi_{\theta}(\mathbf{y} \mid \mathbf{x}) \parallel \pi_{\mathrm{SFT}}(\mathbf{y} \mid \mathbf{x}) \big]}_{\text{solver} \sim \text{“regret minimization”}} \Big] - \underbrace{\beta_2 \cdot \mathbb{D}_{\mathrm{KL}} \Big[\pi_{\phi}(\mathbf{x}) \parallel p_{\mathrm{ref}}(\mathbf{x}) \Big]}_{\text{creator} \sim \text{“regret maximization” (implicit)}} .$$

Conceptually, the cheating will happen when the reference distribution is narrow or wrongly defined. It is important that in our case $p_{\text{ref}}(\mathbf{x})$ represents an *underspecified*, potentially intractable probability distribution over possible tasks (instantiated *via* prompts) in the wild, as a realizable **open-ended reference** that covers the full diversity and complexity of tasks agents may encounter, *not* the initial static prompt set $\mathcal{D}$. The joint regularization towards $\pi_{\text{ref}}(\mathbf{x}, \mathbf{y})$ captures the objective for agents to generalize on alignment in $p_{\text{ref}}(\mathbf{x})$ with broader open-ended prompts, while being close to the SFT policy $\pi_{\text{SFT}}(\mathbf{y}|\mathbf{x})$. In brief, the definition of the conceptual $p_{\text{ref}}(\mathbf{x})$ and the **regularization** avoids collapsing to distributions with easier prompts.

Practically, we do not directly optimize this principle, rather we design a creator-solver game to implicitly and iteratively achieve this. It is important that we use *regret* as the objective and its approximation by the estimate of the *optimal reward advantage*, which avoids selecting easy prompts by design as well. See also § 3.4 on auto-curricula and learning potential for prompt selection, and § G for more connection between the objective and the algorithm.

For **empirical evidence**, as in Table 29, there is a gradual improvement of prompt complexity and quality over iterations with `eva`. We also observe the creator auto-prioritizes learning in problems like math and coding in Fig. 15, which are initially hard for it as in Fig. 16. Thus the creator also practically does not select easier prompts in the `eva` game. Details can be found in § E.

Prompt Set (↓) **/ Metric** (→)	**Complexity** (1-5)	**Quality** (1-5)
UltraFeedback (seed)	2.90	3.18
UltraFeedback-**eva**-Iter-1	3.84	3.59
UltraFeedback-**eva**-Iter-2	3.92	3.63
UltraFeedback-**eva**-Iter-3	**3.98**	**3.73**

Table 29: `eva` improves prompt quality and complexity.

In addition, the whole literature of curriculum RL, open-ended learning and so on are about designing the right metric for the agents to learn increasingly complex and general capabilities, and we summarize at § H for the reviewer's reference.

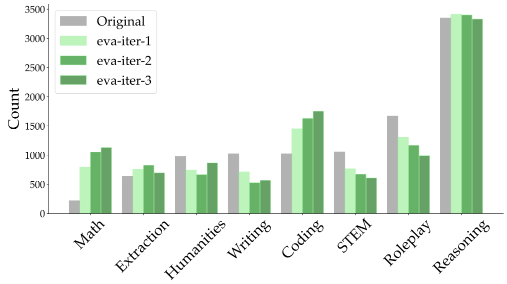

Figure 15: **Training distributions.** The prompt distribution of Table 11 for evolved prompts by zero-shot classification. `eva` creates a curriculum that prioritizes math / coding prompts.

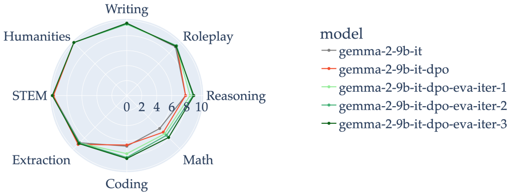

Figure 16: **Benchmark performance.** The radar figure for ratings on MT-Bench (Zheng et al., 2023), where each category contains ten problems. `eva` prioritizes and gradually improves on coding, math and reasoning over iterations, implicitly reflecting a learned curriculum.

> **Remarks.** We thank Reviewer `zxTK` for the constructive feedback. We have made careful efforts to address all the weaknesses and questions raised. We would be grateful if the reviewer may kindly consider again the rating for `eva`, also with regard to its strong performance gain, theeasy-to-implement method, as well as the new principle and the tractable algorithm. If there are additional concerns, we are more than happy to discuss and revise our manuscript further.

REBUTTAL TO REVIEWER I9KX

Summary. We thank the reviewer for the thoughtful and detailed feedback. In response, we have:

- provided a point-by-point rebuttal fully addressing each suggested weakness and question;
- revised definitions for the regret and the informativeness proxy in § 3;
- added a detailed review on Bi-Level RLHF and open-ended learning in § I and § H;
- added detailed illustration on the method, from the principle to the asymmetric game setting, then to the regret minimization by the solver and maximization by the creator in § G;
- added new experiments on empirical gains and prompt evaluation of `eva` in § D and § E.

To avoid repetition, we reorganize the questions and group related answers into a single response.

> **Q5 (Intuition on Open-Ended RLHF)**: Can you provide intuitions behind equation 7, on the KL divergence between the joint policy for both prompt and response? Is it even tractable to estimate or approximate this KL?
>
> **W2 (Regret and KL)**: The optimization is over π in Eq. 9 for solving the minimax regret. However, its not absolutely clear how the KL divergence plays a role here and how it is ensured that the response and prompt distributions are close to reference. Without that, the alignment problem is ill-defined. Please provide concrete justifications in theory and empirical results.
>
> **W1 (Proxy Tractability)**: How is Eq. 10 tractable and being solved? Any heuristic of sampling and approximating should result in sub-optimality which is not clear where its accounted.

TL;DR: We have added § G to fully address related concerns. Regarding specific concerns:

- **Q5 - Intuition.** The joint regularization towards $\pi_{\text{ref}}(\mathbf{x}, \mathbf{y})$ captures the objective for agents to generalize on alignment in $p_{\text{ref}}(\mathbf{x})$ with broader open-ended prompts, while being close to $\pi_{\text{SFT}}(\mathbf{y}|\mathbf{x})$. Note that $p_{\text{ref}}(\mathbf{x})$ is the *underspecified* open-ended reference, *not* the initial static prompt set $\mathcal{D}$. We can reformulate the principle to Eq. 8, while the KL on SFT response policy is tractable, we need to approximate the KL on the open-ended reference $p_{\text{ref}}(\mathbf{x})$. One way to achieve this heuristically is by iteratively creating a *sequence* of prompt distributions.
- **Q2 - KL.** We have revised Eq. 9 so that the regret is the difference in the reward of the current policy and the KL-optimal policy (thanks for catching this). For the solver, by design, preference optimization would be equivalent to regret minimization, thus the alignment problem remains to be correctly defined. For the creator, the distribution matching to the open-ended reference is implicitly achieved by prompt curriculum construction, and we present empirical evidence in § E to justify that prompts are evolving towards broader tasks with higher complexity.
- **W1 - Proxy.** We have revised Definition 2 for better readability. It is estimated by sampling multiple responses from the stochastic policy and calculating the reward range (or other advantage-based proxy). This approximation will result in sub-optimality for creator's regret maximization process, and we present more discussion in § G.2.

Rebuttal: For the rebuttal to be self-contained, we extract contents from § G here.

First, we re-present the open-ended RLHF principle, and discuss the intuition under the KL regularization. Next, we show heuristic approaches in open-ended learning to approximate this objective, with a focus on minimax game formulation. Finally, we formalize the regret objective in our RLHF setting, and discuss the regret minimization for the solver and the regret maximization for the creator.

J.1 THE CONCEPTUAL OPEN-ENDED RLHF FORMULATION

Classical RLHF optimizes over a static prompt set:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\Big[\beta\cdot\mathbb{D}_{\text{KL}}\Big[\,\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}) \parallel \pi_{\text{SFT}}(\mathbf{y}\mid\mathbf{x})\,\Big]\Big].$$

We propose to drop the static prompt set assumption, and jointly update the prompt distribution via a creator policy for Open-Ended RLHF, with the ideal objective below:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\pi_{\boldsymbol{\phi}}(\cdot),\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \beta\cdot\mathbb{D}_{\mathrm{KL}}\Big[\, \pi_{\phi}(\mathbf{x})\cdot\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}) \,\|\, p_{\mathrm{ref}}(\mathbf{x})\cdot\pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x}) \,\Big].$$

This generalizes RLHF (Eq. 1). We can rewrite Eq. 7 with modified coefficients for precision:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\pi_{\phi}(\cdot)}\Big[\underbrace{\mathbb{E}_{\mathbf{y}\sim\pi_{\theta}(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \beta_1\cdot\mathbb{D}_{\mathrm{KL}}\big[\pi_{\theta}(\mathbf{y}\mid\mathbf{x}) \,\|\, \pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\big]}_{\text{solver}} \Big] - \beta_2\cdot \underbrace{\mathbb{D}_{\mathrm{KL}}\Big[\, \pi_{\phi}(\mathbf{x}) \,\|\, p_{\mathrm{ref}}(\mathbf{x}) \,\Big]}_{\text{creator}}.$$

The newly proposed p_{ref} represents an *underspecified*, potentially intractable probability distribution over possible tasks in the **open-ended world** (instatiated *via* prompts) – it is *not* the initial static training prompt distribution (which is only the seed set for the creator to evolve upon); it can be seen serve as a conceptual guide to steer the prompt distribution.

To further clarify, there are two types of regularization in open-ended RLHF:

- $\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(\mathbf{y}\mid\mathbf{x}) \,\|\, \pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\right]$: this is the classical regularization on the response policy, ensuring that no matter how the training distribution over prompts evolves during optimization, the response policy remained anchored to the supervised fine-tuned (SFT) policy.
  - This KL (and preference optimization) is **explicitly achieved** in plug-in algorithms (*e.g.*, DPO) in Algo. 1. We later show how it relates to **solver's regret minimization**.
- $\mathbb{D}_{\mathrm{KL}}\Big[\, \pi_\phi(\mathbf{x}) \,\|\, p_{\mathrm{ref}}(\mathbf{x}) \,\Big]$: this probability matching term captures the intuition on optimizing $\pi_\phi(\mathbf{x})$ to approach the conceptualized $p_{\mathrm{ref}}(\mathbf{x})$, in the sense that a language model optimizes itself by adapting its training distributions with newly generated prompts for self-training to develop increasingly general capabilities, directing its learning towards informative, new tasks (Jiang, 2023), instead being constrained in a static, pre-defined set of tasks.
  - This conceptual KL is **implicitly achieved** by the creator step in the current **`eva`** setting by training on a *sequence of informative prompt sets*. We later show how it relates to **creator's regret maximization**. As illustrated in § 3.3.1, we start from the seed prompt set, choose those high-regret prompts and generate variations upon them by `EvolInstruct`, then mixing with a buffer of the original set to form the new training distribution at each iteration. This approach resembles classical open-ended learning in § G.2, and we hope it can serve as a small step for future works to build upon.
  - A common misunderstanding among readers may be to confuse the open-ended reference $p_{\mathrm{ref}}(\mathbf{x})$ with the initial seed prompt distribution $\mathcal{D}$, which is static. In contrast, $p_{\mathrm{ref}}(\mathbf{x})$ represents a broader space of tasks (*e.g.*, user prompts in the real wild world), as a conceptual target derived from the *underspecified distribution* (Dennis et al., 2020), *i.e.*, an environment with free parameters that control. Let's use an illustrative example with Fig. 6: the prompt distribution may be defined along several dimensions (*e.g.*, the number or complexity of coding problems); a potential creator can be designed to modify these dimensions, steering the initial $\mathcal{D}$ to new training distributions, by certain decision rules (*e.g.*, minimax regret, which offers worst-case guarantees) that forms *a sequence of informative prompts* for training.

This joint optimization objective only serves as a general principle. In the next, we discuss how existing works **implicitly achieve** the open-ended learning objective through **two-player games**.

J.2 APPROACHING OPEN-ENDED LEARNING BY UNSUPERVISED ENVIRONMENT DESIGN

J.2.1 THE ASYMMETRIC GAME FORMULATION FOR UNSUPERVISED ENVIRONMENT DESIGN

While we cannot directly train the agent with the intractable $p_{\mathrm{ref}}(\mathbf{x})$ of the open-ended world, it is possible to curate a **curriculum of prompt distributions** to improve over the static distribution and support the *continual training* of the policy $\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})$, for it to keep improving and succeed over the full task space, thus conceptually approaching $p_{\mathrm{ref}}(\mathbf{x})$. This is often framed as an **asymmetric two-player game**.

Dennis et al. (2020) first formally define this problem as Unsupervised Environment Design (UED). The idea is that while the real-world environments are inexhaustible and hard to tract, there may exist some free parameters (*e.g.*, height and roughness in a maze) which one may control to generate new environments; UED then concerns about designing a distribution of those free parameters (*i.e.*, settings) to create new fully specified environments, that can be used to train the agents.

In this setup, one player, the **creator**, generates new environments based on some specific decision rules (see the following), while the other player, the **solver**, optimizes its policy within these training environments, and the process continues iteratively. Common **heuristic strategies** include:

- **Randomization**: environments are generated uniformly and independently of the solver's current policy. This method is simple but less effective (Tobin et al., 2017).
- **Maximin**: the creator generates environments that minimize the solver's maximum possible reward, which can often lead to unsolvable scenarios (Khirodkar and Kitani, 2018).
- **Minimax regret**: The creator targets environments that maximize the solver's *regret*, defined as the difference between the optimal return achievable and that of the solver's current policy (Beukman et al., 2024b). The regret is often conceived as the **creator's utility**.

Among them[5], the minimax regret approach presents a sweet spot where the creator can create hard yet solvable environments, and is often empirically better. The minimax regret strategy also implies that the agent's policy is trained to perform well under all levels/settings, thus enjoys a worst-case guarantee. However, while it is often straightforward for the solver to minimize the regret (*e.g.*, through direct policy optimization, as we discuss in § G.3), the optimal policy remains *unknown* during the optimization process, thus regret as the decision signal is often intractable to the creator – which requires *approximation* (as an amusing side note, this is described as the Achilles' heel of those curriculum RL methods by Parker-Holder et al. (2022)).

J.2.2 APPROXIMATING THE REGRET AND GENERATING NEW ENVIRONMENTS

In general, the **creator** design in this line of research contains two steps:

1. **identifying high-regret levels** using different (often heuristic) regret approximation;
2. **generating new environments** by making variations or retrieving from buffers on those high-regret levels.

We hereby review major works on regret approximation and environment generation as follows:

Dennis et al. (2020) propose joint training for the creator and two competing solvers.

- **Regret approximation**: here, two solver policies are trained, with the regret approximated as the **difference in their returns**. During each optimization step, one solver *maximizes* this regret, the other *minimizes* it, and the creator maximizes it.
- **Environment generation**: the system directly sample the parameter from the creator policy and use that to specify the environment.

Jiang et al. (2021b) propose to random sampling on high-regret levels.

- **Regret approximation**: as a heuristic, the authors use *positive value loss*, which is a function of Generalized Advantage Estimate (Schulman et al., 2015) (which itself is a function of the TD error – the difference between the expected and the actual returns) as the creator's utility.
- **Environment generation**: the creator have a rolloing buffer of highest-regret levels by random searching on relevant configurations.

Jiang et al. (2021a) further propose a double-creator setting based on (Jiang et al., 2021b), where one creator is actively generating new environments, and the other is retrieving from the buffer.

Parker-Holder et al. (2022) propose to sample high-regret levels and generate new environments by making *edits* on existing ones. The regret approximation is the same as (Jiang et al., 2021b) – the

---

[5] We have implemented variants of these in § 4.2.1, and show minimax regret is empirically better.

positive value loss. For the environment generation, the authors suggest a general editing/mutation mechanism, where the creator chooses from high-regret levels and make small variations within an edit distance, which by heuristics will lead to the discovery of more high-regret environments. There is an additional filtering step: they do not directly train on the newly generated levels, but evaluate on those levels first, then add only the high-regret ones to the training buffer.

Note the solvers are often directly trained with PPO (Schulman et al., 2017) under the environments.

J.3 REGRET FORMULATION FOR OPEN-ENDED RLHF

Next, we discuss the regret minimization and maximization in our setting for alignment. Specifically,

- **Regret minimization for the solver**: we avoid calculating regret and use direct policy optimization (*e.g.*, DPO) to equivalently achieve regret minimization.
- **Regret maximization for the creator**: similarly to (Jiang et al., 2021b; Parker-Holder et al., 2022), we first find an approximation of regret, then curate new environments for the solver to train on by (i) sampling from a replay buffer of existing prompts, and (ii) making variations (through `EvolInstruct` (Xu et al., 2023a)) on those high-regret prompts. Specifically, we use **advantage-based estimates of the current policy**, as summarized in Table 2.

This asymmetric two-player game serves as one potential modeling choice to implicitly achieve the open-ended RLHF principle that we proposed in Definition 1. We look forward to exploring more principled solutions in the future.

KL-regularized regret. In the RLHF setting at fixed prompt distribution, the objective is:

$$\max_{\pi_{\boldsymbol{\theta}}} \mathbb{E}_{\mathbf{x}\sim\pi_\phi(\cdot),\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[r(\mathbf{x},\mathbf{y}) \Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\Big[\beta\cdot\pi_\phi(\cdot)_{\mathrm{KL}}\Big[\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}) \,\|\, \pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x}) \Big]\Big].$$

The optimal policy of the above KL-constrained objective is:

$$\pi^\star_{\mathrm{KL}}(\mathbf{y}\mid\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\exp\left(\frac{1}{\beta}\cdot r(\mathbf{x},\mathbf{y})\right).$$

In our current setting, we assume there is an oracle preference model for the preference pair labeling.

We can now formally define the *regret* with regard to $\pi^\star_{\mathrm{KL}}(\cdot\mid\mathbf{x})$ as:

$$\mathrm{Regret}_{\mathrm{KL}}(\mathbf{x},\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[r(\mathbf{x},\mathbf{y}) \Big] - \mathbb{E}_{\mathbf{y}\sim\pi^\star_{\mathrm{KL}}(\cdot|\mathbf{x})}\Big[r(\mathbf{x},\mathbf{y}) \Big]. \tag{22}$$

Regret Minimization for the Solver. It is rather straightforward/trivial to understand the objective of the solver $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ as regret minimization, since the goal is to align the policy $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ with the KL-optimal solution $\pi^\star_{\mathrm{KL}}(\cdot\mid\mathbf{x})$, which directly minimizes the KL-regularized regret by design. This formulation allows flexibility in the plug-in preference optimization algorithms for the solver's step in Algorithm 1, and ensures *the alignment problem is well-defined*. In practice, we use Direct Preference Optimization (DPO) and its variants, which approximate the KL-optimal solution by iteratively adjusting $\pi_{\boldsymbol{\theta}}$ to reflect preference differences.

Regret Maximization for the Creator. As discussed previously, while it is often trivial for the solver to minimize the regret through direct policy optimization, the optimal policy remains unknown during the optimization process, thus we cannot directly calculate the regret – we must approximate it when using it as the utility for the creator. Similarly to heuristics proposed by prior works (Jiang et al., 2021b;a; Parker-Holder et al., 2022), we use the advantage-based estimate:

$$|\widehat{\mathrm{Regret}}(\mathbf{x},\pi_{\boldsymbol{\theta}})| \leftarrow \mathrm{info}_{\boldsymbol{\theta}}(\mathbf{x}) := r(\mathbf{x},\mathbf{y}_+) - r(\mathbf{x},\mathbf{y}_{\text{baseline}}), \tag{23}$$

where

$$\mathbf{y}_+ := \arg\max\nolimits_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y}), \tag{24}$$

$$\mathbf{y}_{\text{baseline}} := \arg\min\nolimits_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y}) \text{ or } \mathbf{y}_{\text{baseline}} := \mathrm{avg}_{\mathbf{y}_i}\ r(\mathbf{x},\mathbf{y}), \tag{25}$$

and $\{\mathbf{y}_i\}_{i=1}$ is a set of responses sampled from $\pi_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x})$ and $r(\cdot, \cdot)$ is the reward oracle. We use $\arg\min_{\mathbf{y}_i} r(\mathbf{x}, \mathbf{y})$ by default due to its simplicity and efficiency (see also § 3.4 for additional interpretation) and consistent strong empirical gains we observed in vast experiments. As the policy optimizes, the proxy should approximate the true regret better. We leave exploration of other informativeness proxy designs in **eva** to future work.

For new environment generation, as illustrated in § 3.3.1, we start from the seed prompt set, choose those high-regret prompts and generate variations upon them by `EvolInstruct`, then mixing with a buffer of the original set to form the new training distribution at each iteration.

> **W3 (Understanding the Iterative Algorithm)**: As described in Algorithm 1, informativeness is evaluated and a prompt subset is created based on current policy estimate and then the policy is updated based on the prompt subset. However, this causes an inter-dependence between the two which leads to nested structure, which is not clearly explained. Specifically, while computing the informativeness score for the prompts, it depends on $\theta^\star(x_{t-1})$, *i.e.*, optimal parameter for the previous distribution. Provide clear explaination on the same.

TL;DR: (i) We revised Algo. 1 with updated subscripts to reflect the training process – please take a look in our main paper. Given a current model checkpoint, we evaluate the prompt informativeness based on it, and evolve a new prompt set more informative to the current checkpoint, and use the new prompt set for continual training. (ii) We intend to use an iterative best-response framework to approximate equilibrium in expectation, balancing computational efficiency and practicality.

Rebuttal: The iterative updates in **eva**, as described in Algo. 1, are based on a best-response-to-best-response framework. Specifically, the creator updates the prompt distribution based on the solver's current policy, and the solver then optimizes its policy over the updated prompts, and the process repeats. This sequential structure approximates a Nash equilibrium in expectation over iterations, inspired by works such as Freund and Schapire (1999); Wu et al. (2024), which establish convergence to optimal policies on average through iterative optimization.

We intentionally avoid simultaneous joint optimization as it would significantly increase computational and memory overhead, making it less practical for integration into current RLHF pipelines.

> **W4 (Understanding Reward Models)**: While iterating, every new prompt distribution will require generating new response, how is the evaluation coming from which reward model? Is the ground reward available, if not please explain how the preference is obtained and how does it affect suboptimality? Also: **Q3 (RM Availability)**: What's the reward model availability? Is the true reward model available?

TL;DR: We assume a preference oracle provided by an **external, pre-trained reward model**, which is practically used in many real-world LLM training scenarios (Team et al., 2023).

Rebuttal: As discussed in the beginning of the experimental setting in § 4, we assume the availability of a pre-trained, fixed reward model. This approach is practically adopted in industry (Team et al., 2023; 2024a;b) and is also commonly used in academia works (Xu et al., 2023b; Meng et al., 2024; Wu et al., 2024). The reason is more on efficiency concerns. For example, in GEMMA-2 training, the reward model is *an order of magnitude larger* than the policy (Team et al., 2024b); it would thus be impractical or the gain may only be marginal if we update the reward model on-the-fly.

Nevertheless, it is possible to incorporate the online RM training within **eva** – we have shown in § 4.2.3 (ablation studies) that **eva** scales with quality of reward models, thus integrating online RM training may further enhance performance and address the potential distribution mismatch problem. We believe this is an interesting direction to pursue, and have listed it in § 6 (future works) on adding more players including rewarders in the self-play loop.

> **W5 (Improvement of Sub-Optimality)**: Overall, which expression/Theorem guides us in understanding the improvement of prior suboptimality is not clear? Can you please point out/highlight how the current method improves upon the prior suboptimality due to static prompt distribution?

TL;DR: The improvement of sub-optimality is guided by the minimax regret objective (Remark 1) through its iterative implementation. While this work does not explicitly derive suboptimality bounds, our approach has demonstrated **strong empirical gains** over the training by static distributions, as shown in § 4 (main experiments), § E (benchmark performance), and § D.2 (alignment gains over iterations).

Rebuttal: In general, the improvement of prior suboptimality due to static prompt distributions is guided by the minimax game outlined in Remark 1. This expression forms the basic foundation for our iterative algorithm, where the creator updates prompts to maximize informativeness (proxy for regret), and the solver minimizes regret (through direct preference optimization). This iterative process ensures the solver and creator adapt to each other, implicitly forming a curriculum and addressing sub-optimality in static prompts. We also added § G to help illustrate the intuition behind.

In general, the empirical results in § 4 (main results), § E (curriculum effect and benchmark improvement), and § D.2 (alignment gains over iterations) demonstrate that the dynamic prompt distribution improves solver performance and alignment metrics, thereby mitigating suboptimality. While the current package does not explicitly derive sub-optimality bounds (as would be typical in formal RL/bandit theory papers) and emphasizes practicality and usability as a methodology paper, we would love to learn if the reviewer has any suggestions for this as the future work.

> **W6 (Prompt Distribution)**: It is extremely crucial to show the prompt distribution and demonstrate its perplexity to ensure its not generating some meaningless or irrelevant prompts, since its not very evident on the KL divergence in the prompt space and its relation with the informative measure. Please provide detailed explanation to clarify that.

TL;DR: We have (i) added experimental results in § E (prompt distribution visualization) and § J (prompt examples) to verify that `eva` evolves meaningful and relevant prompts with improved complexity and quality; (ii) added explanation in § G (detailed illustration of method) on the KL regularization in the prompt space and the implicit approximation by the creator.

Rebuttal: (This rebuttal also addresses Reviewer `ac2p`'s concerns on curriculum.)

We have revised the manuscript with additional visualization on potential curriculum learned in § E. In general, we observe the creator prioritizes learning in math and coding for the generated prompt distribution, which brings gradual improvement on benchmark performance on relevant categories over iterations. In other words, `eva` effectively shifts focus towards harder yet learnable categories. We have attached the bar plot and radar figure here for the reviewer's reference:

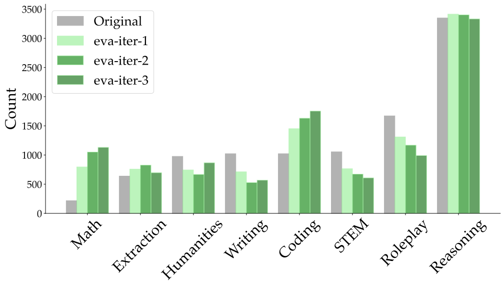

Figure 17: **Training distributions.** The prompt distribution of Table 11 for evolved prompts by zero-shot classification. `eva` creates a curriculum that prioritizes math / coding prompts.

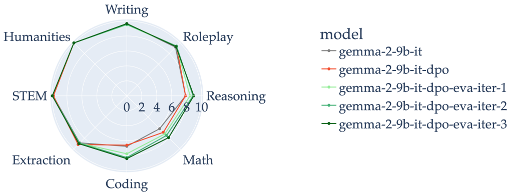

Figure 18: **Benchmark performance.** The radar figure for ratings on MT-Bench (Zheng et al., 2023), where each category contains ten problems. `eva` prioritizes and gradually improves on coding, math and reasoning over iterations, implicitly reflecting a learned curriculum.

We have added Table 16 providing qualitative examples for evolved prompts. Also, as noted in § G, the solver maintains KL regularization during optimization, ensuring that the response distribution remain close to the reference policy; in the this work, we do not explicitly add KL regularization in the prompt distribution since we do not directly conduct parameter update for the creator (which we empirically find to bring training instability); rather, we use **meta instructions** and **buffer sampling** to constrain the prompt generations (as described in § 3.3, § A and § D.1), which is empirically very effective, and introduces only minimal changes to existing pipeline thus can be easily applied. As noted in § 6, we look forward to future works on making the creator policy differentiable.

Regarding the relation with the informativeness measure, our current proxy is an efficient proxy among many possibilities. We have provided detailed discussions in § G to help interpret it. There could be other proxies – one interesting direction is to completely remove the dependence on the reward model and directly use model likelihoods to make the prompt selection.

We note that perplexity is not commonly used or not the most preferred measure for data quality in practical training of large language models (Team et al., 2023; Fang et al., 2024), and can be computationally heavy to measure. We have added experiments in § E.1 which we follow Liu et al. (2023b) to generatively measure the complexity and quality of prompt distributions. As in Table 30, there is a gradual improvement of prompt complexity and quality over iterations with **`eva`**. We hope this would address the reviewer's concerns.

Prompt Set (↓) **/ Metric** (→)	**Complexity** (1-5)	**Quality** (1-5)
UltraFeedback (seed)	2.90	3.18
UltraFeedback-**`eva`**-Iter-1	3.84	3.59
UltraFeedback-**`eva`**-Iter-2	3.92	3.63
UltraFeedback-**`eva`**-Iter-3	**3.98**	**3.73**

Table 30: **`eva`** improves prompt quality and complexity.

> **Q1 and Q2 (KL in the Solver Loop)**: Since equation 7, can't be directly solved, and is solved in an asymmetric fashion, then in the solver loop the KL should be over the response distribution and not joint right? How is the KL divergence w.r.t reference policy for the algorithm? Please provide detailed ablation.

Answer: (i) Yes, in the solver loop, the KL regularization is applied over the response distribution, not the joint distribution, as shown in Line 5 of Algo. 1. (ii) The KL divergence *w.r.t.* reference policy is determined by the plug-in solver (*e.g.*, DPO, SimPO, ...), which is orthogonal to our framework. We have added detailed explanation in § G to illustrate the whole process.

> **Q4 (Literature)**: There is a recent line of works on Stacklberg and Bilevel RLHF which deals with the entanglement in a leader-follower setting. Although not specific to updating prompt dist, but can be trivially applied. Provide a detailed comparison with the literature around that [1,2,3].

TL;DR: We thank the reviewer for this nice suggestion. Please see below for a detailed review on the relevant literature, highlighting the unique contribution of **`eva`**. We have added § I in the manuscript.

Rebuttal: Bi-level optimization refers to optimization problems where the cost function is defined *w.r.t.* the optimal solution to another optimization problem (Grosse, 2022). There is a recent line of works applying bi-level optimization to RLHF. While they all rely on a fixed dataset of prompts, **`eva`** propose to dynamically update the prompt set, as in § 1. We present a detailed comparison of **`eva`** with Ding et al. (2024); Shen et al. (2024); Makar-Limanov et al. (2024). We thank the anonymous reviewer for the kind references, and welcome suggestions for any other works we may have missed.

Ding et al. (2024) formulate iterative online RLHF as a bi-level optimization problem, where the upper-level represents the reward learning, and the lower-level represents the policy optimization. Leveraging reward re-parameterization tricks in Rafailov et al. (2023), Ding et al. (2024) reduces the problem to a single-level objective with regard to the policy. The differences of this work and our work lie in the prompt distribution and preference oracle: (i) **`eva`** features by **dynamic prompt set generation for Open-Ended RLHF**, whereas (Ding et al., 2024) remains using a static prompt set; (ii) we assume the existence of the preference oracle (as discussed in § 4), while Ding et al. (2024) consider online training of reward models and ablate on self-rewarding by the current LLM policy. Our usage of a pre-trained reward model follows from industrial practices (Team et al., 2023; 2024b), which is also commonly used by prior works in academia (Meng et al., 2024; Wu et al., 2024).

Makar-Limanov et al. (2024) provide an interesting exploration on formulating RLHF as a leader-follower game, where the language model (LM) policy is the leader and the reward model (RM) policy is the follower, and the solution is **Stackelberg equilibrium** (von Stackelberg, 1934; Rajeswaran et al., 2020), where the *leader does not likewise best respond to the follower's strategy*. Here, following the curriculum RL literature (Dennis et al., 2020; Parker-Holder et al., 2022), we seek the **Nash equilibrium** (Nash et al., 1950) between the creator for prompt generation and the solver for response generation. In the current setting of **`eva`**, the goal is to search for an optimal solver policy with a best supporting prompt distribution, *and* an optimal prompt distribution with a best supporting solver policy. Nevertheless, the LM-RM iterative optimization may be added on top of **`eva`**'s framework, and we look forward to future works exploring the leader-follower re-formulation of **`eva`**.

Shen et al. (2024) present a rigorous theoretical work (though it does not directly involve practical post-training of large language models). The authors propose to reduce the bi-level problem to a single-level problem with a penalty-based reformulation, and apply it in the setting of LM-RM optimization within a *fixed* environment, whereas **`eva`** focuces on dynamic prompt generation and practically train large language models with extensive empirical experiments conducted. We believe it would be interesting to adapt similar first-order optimization techniques to solve Open-Ended RLHF.

In summary, existing bi-level RLHF works focus on online optimization of both the RM and the LM (as the response policy), all with **fixed** prompt/state distribution. **`eva`** presents an orthogonal direction on **dynamic** prompt generation for Open-Ended RLHF, with an empirical algorithm which attains state-of-the-art performance with large language models on a variety of benchmarks. It is possible to incorporate the online RM training within **`eva`** – we have shown in § 4.2.3 that **`eva`** scales with quality of reward models, thus integrating online RM training may further enhance performance and mitigate potential distributional mismatch problems as we evolves for more prompts. This direction may have not been widely adopted in real-world training of language models, due to concerns about practicality (Team et al., 2023; 2024a;b; Adler et al., 2024). We look forward to future works exploring *efficient* variations unifying **`eva`** and existing bi-level RM-LM frameworks.

> **Final Remarks.** We thank the reviewer for the precious time and efforts on the **`eva`** method. We value all those opinions, and have made every effort to thoroughly address the concerns raised and revise our manuscript accordingly. Regarding the rejection, we hope the reviewer may kindly consider the points that we have summarized at the beginning of this rebuttal, on the **strong empirical alignment gain** brought by the **simple design** of **`eva`**, also on judging the merit of a work (*cf.,* (Castro, 2021)) *w.r.t.* the practicality and how the community may easily build on top of the principle and the method we proposed (*cf.,* (Hamming, 1986)), which we are confident are valuable to the broader alignment community. We look forward to any future discussion, and would be grateful if the reviewer may consider revising the score if the revision is satisfactory.

APPENDIX

The appendix is organized as follows:

- § A - **Details On Reproducibility**
- § B - **Plug-In Loss Functions Used In Main Results**
- § C - **Extended Results for Experiments in the Main Paper**
- § D - **Additional Experiments**
- § G - **Illustration on Methodology**
- § E and § J - **Illustrations on Prompts, Responses and Relevant Distributions**
- § H and § I - **Additional Literature Review**

A DETAILS ON REPRODUCIBILITY

Our code is built based on many open-source packages, and we sincerely thank every developer and contributor of these projects for their efforts and contributions to the community.

Code release. We hope to open-source all codes, generated data and trained models, *upon approval* – before then, we are more than happy to provide any clarification to help re-implement **`eva`** and replicate our results. In general, our code base is made to be simple to use for practitioners, requiring **only a creator module addition** within the commonly adopted Alignment Handbook pipeline.

Hyperparameter settings. We follow the original hyperparameter settings as in (Hong et al., 2024; Meng et al., 2024; Wu et al., 2024), default to be:

Hyperparameter (↓) / Loss (→)	**DPO**	**ORPO**	**SimPO**	**SPPO**
learning rate	5e-7	5e-7	8e-7	5e-7
learning rate scheduler	cosine	cosine	cosine	linear
β	0.05	/	10	0.001
γ	/	/	5	/
λ	/	0.5	/	/
no. epochs per iter	2	1	1	6
warmup ratio per iter	0.1	0.1	0.1	0.1
effective batch size	8	8	32	8
max length	2048	2048	2048	1024
max prompt length	1024	1024	1024	512
optimizer	adamw	adamw	adamw	rmsprop

Iterative Training Settings. By default (Tran et al., 2023; Yuan et al., 2024), we train with equal-size prompt subset in each iteration. Unless otherwise specified, we use 10K prompts from the UltraFeedback dataset (Cui et al., 2023) per iteration. The incremental training proceeds as follows:

- $\boldsymbol{\theta}_0$: Base SFT model.
- $\boldsymbol{\theta}_{0\to1}$: initialize with $\boldsymbol{\theta}_0$; then train with the prompt split $\mathcal{X}_1$ by self-generated responses from the initial model $\boldsymbol{\theta}_0$.
- $\boldsymbol{\theta}_{1\to2}$: initialize with $\boldsymbol{\theta}_{0\to1}$; trained with the prompt split $\mathcal{X}_2$ via by self-generated responses from the initial model $\boldsymbol{\theta}_{0\to1}$.

For evolving prompts (*e.g.*, evolving $\mathcal{X}_1$ to $\mathcal{X}_{\tilde{1}}$), with the calculated informativeness metric for each prompt, we normalize them as the weight to do weighted sampling for a 25% informative subset to get $\mathcal{X}_1^{\texttt{info}}$. We then iterate over in $\mathcal{X}_1^{\texttt{info}}$ and call `EvolInstrut` (Xu et al., 2023a) as the plug-in evolving method (with the number of evolutions as 4) using the default mutation templates for (i) in-depth evolving (constraints, deepening, concretizing, increased reasoning steps) and (ii) in-breadth evolving (extrapolation) as implemented in `tasks/evol_instruct/utils.py` of `distilabel==1.3.2`. Next we uniformly select 80% prompts from this evolved dataset and 20% from the original dataset (*i.e.*, the buffer) to form $\mathcal{X}_{\tilde{1}}$. We do not seek extensive parameter search (*e.g.*, the number of evolutions, the evolving ratio) in this stage and encourage future works on exploring this and other plug-in evolving methods. For solver we generate 6 responses per prompt.

Software environments. All experiments are conducted on 8xNVIDIA H100 SXM GPUs. Our codebase primarily relies on `transformers==4.40.0`. For the response generation of GEMMA models at the training stage, we use `vllm==0.5.4` with `flashinfer` backend for `CUDA 12.4` and `torch 2.4`. For evolving prompts, we use `distilabel==1.3.2`, and use `LiteLLM` to serve Gemini (default to be `gemini-1.5-pro`) and transformers models (default to be `gemma-2-9b-it`). For evaluation on all benchmarks, we use `sglang==0.2.10` and `openai==1.35.14`, with `gpt-4-1106-preview` as the judge model and `gpt-4-0314-preview` as the baseline model. Specifically for AlpacaEval 2.0, we use `alpaca_eval_gpt4_turbo_fn` as the annotator config. We use 42 as the random seed.

B PLUG-IN LOSS FUNCTIONS USED IN MAIN RESULTS

With Reference Model	
DPO (Rafailov et al., 2023)	$\ell_\beta(\pi_{\boldsymbol{\theta}}) = -\log\left[\sigma\left(\beta \cdot \Delta^{\mathbf{x}}_{\pi_{\boldsymbol{\theta}};\pi_{\text{ref}}}\right)\right] := -\log\left[\sigma\left(\beta \cdot \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{y}_+\|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_+\|\mathbf{x})} - \beta \cdot \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{y}_-\|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_-\|\mathbf{x})}\right)\right]$
SPPO (Wu et al., 2024)	$\ell_\beta(\pi_{\boldsymbol{\theta}}) = -\log\left[\sigma\left(\left(\beta \cdot \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{y}_+\|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_+\|\mathbf{x})} - \frac{1}{2}\right)^2 + \left(\beta \cdot \log \frac{\pi_{\boldsymbol{\theta}}(\mathbf{y}_-\|\mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_-\|\mathbf{x})} + \frac{1}{2}\right)^2\right)\right]$
Without Reference Model	
SimPO (Meng et al., 2024)	$\ell_{\beta,\alpha}(\pi_{\boldsymbol{\theta}}) = -\log\left[\sigma\left(\beta \cdot \Delta^{\mathbf{x}}_{\pi_{\boldsymbol{\theta}}^{1/\|\mathbf{y}\|};1} - \alpha\right)\right] := -\log\left[\sigma\left(\frac{\beta}{\|\mathbf{y}_+\|} \log \pi_{\boldsymbol{\theta}}(\mathbf{y}_+\|\mathbf{x}) - \frac{\beta}{\|\mathbf{y}_-\|} \log \pi_{\boldsymbol{\theta}}(\mathbf{y}_-\|\mathbf{x}) - \alpha\right)\right]$
ORPO (Hong et al., 2024)	$\ell_\lambda(\pi_{\boldsymbol{\theta}}) = -\log\left[\sigma\left(\lambda \cdot \Delta^{\mathbf{x}}_{\text{odds}_{\boldsymbol{\theta}};1}\right)\right] := -\log\left[\sigma\left(\lambda \cdot \log \frac{\text{odds}_{\boldsymbol{\theta}}(\mathbf{y}_+\|\mathbf{x})}{\text{odds}_{\boldsymbol{\theta}}(\mathbf{y}_-\|\mathbf{x})}\right)\right]$, where $\text{odds}_{\boldsymbol{\theta}} = \frac{\pi_{\boldsymbol{\theta}}}{1-\pi_{\boldsymbol{\theta}}}$

Table 6: Direct preference alignment algorithms used in the main experiments. In parameter tuning, we include an additional negative log-likelihood loss for chosen responses (*i.e.*, $\frac{\gamma}{|\mathbf{y}_+|} \log \pi_{\boldsymbol{\theta}}(\mathbf{y}_+|\mathbf{x})$).

C ADDITIONAL EXPERIMENTAL RESULTS FOR THE MAIN PAPER

In general, **eva** maintains the accuracy on downstream tasks and is robust on those reasoning-heavy tasks, and the scaling with reward models is more prominent on AlpacaEval, possibly due to the training sources for such reward models.

Method (↓) / Dataset (→)	**MUSR-TA**	**TruthfulQA-Gen**	**WMDP**	**GSM8K**	**GSM-Plus**	**MMLU-Pro**
$\boldsymbol{\theta}_0$: SFT	38.80	34.76	58.62	24.64	18.62	52.08
$\boldsymbol{\theta}_{0\to 1}$: DPO	38.40	34.76	58.45	24.56	18.50	52.63
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva**	38.40	34.15	58.40	24.26	17.96	**53.03**
$\boldsymbol{\theta}_{0\to 1}$: SPPO	40.80	34.15	58.72	24.79	18.42	52.70
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva**	**41.20**	34.64	**58.94**	**25.40**	**18.88**	52.47

Table 7: **Performance on Downstream tasks.**

Model Family (→)	GEMMA-2-9B-IT					
Benchmark (→)	**MT-Bench**			**Arena-Hard**	**AlpacaEval 2.0**	
Method (↓) / Metric (→)	**avg. score**	1st turn	2nd turn	**WR** (%)	**LC** (%)	**WR** (%)
$\boldsymbol{\theta}_{0\to 1}$: DPO	8.66	9.01	8.32	51.6	55.01	51.68
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva-i** (ARMO-8B)	**8.90**	9.04	8.75	60.1	55.35	55.53
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva-i** (SKYWORKRM-27B)	8.75	9.07	8.43	**60.3**	**56.12**	**56.40**

Table 8: **Effect of (pointwise) reward models.**

Model Family (→)	GEMMA-2-9B-IT					
Benchmark (→)	**MT-Bench**			**Arena-Hard**	**AlpacaEval 2.0**	
Method (↓) / Metric (→)	**avg. score**	1st turn	2nd turn	**WR** (%)	**LC** (%)	**WR** (%)
$\boldsymbol{\theta}_{0\to 1}$: SPPO	8.62	9.03	8.21	55.7	51.58	42.17
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva-i** (PAIRRM-0.4B)	8.78	**9.11**	8.45	58.9	51.86	43.04
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva-i** (PAIRRM-8B)	**8.89**	9.08	**8.70**	**60.2**	**52.71**	**44.52**

Table 9: **Effect of (pairwise) reward models.**

D ADDITIONAL EXPERIMENTAL RESULTS (AS EXTENSIONS)

D.1 EXPERIMENTS ON DIFFERENT `evolve(·)` METHODS

As an addition to Table 1, we have experimented with three different `evolve`(·) methods, including:

- **SelfInstruct** (Wang et al., 2022): Given seed prompts, variations are created based on criteria such as verb diversity and style blending (mixing interrogative and imperative styles). Unlike EvolInstruct (Xu et al., 2023a), which generates prompt variations sequentially, this approach generates independently. We follow the one-shot implementation in `self_instruct.py` of `distilabel==1.4.1` and modified the instruction on conciseness so that newly generated prompts have similar lengths compared to the seed prompts.
- **EvolQuality** and **EvolComplexity** (Liu et al., 2023b): The two methods use the same evolutionary approach (*i.e.*, sequentially generating), but with slightly different meta-instructions for prompt generation, where EvolQuality asks to improve the quality (*i.e.*, helpfulness, relevance, etc) of the seed prompt and EvolComplexity asks to improve the complexity (*i.e.*, increased reasoning steps, etc) of the seed prompt. We follow the implementation in `evol_quality/utils.py` and `evol_complexity/utils.py` of `distilabel==1.4.1`.

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) / **Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO	51.6	651
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `EvolInstruct`)	60.1	733
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `EvolQuality`)	58.7	721
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `EvolComplexity`)	**60.6**	749
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (`evolve`(·) = `SelfInstruct`)	57.2	725

Table 10: Results of using different evolving methods.

eva is effective under different evolving methods. As shown in Table 10, our method brings strong performance gain without training with additional human prompts. Among the experimented methods, we find EvolComplexity shows better results.

We believe the main strength of such method is its **simplicity**. Viewing the evolving process as $\mathbf{x}' \leftarrow p_{\boldsymbol{\theta}}(\cdot \mid \mathbf{x}, \texttt{meta\_prompt})$, one can easily tune the meta prompt in natural language for improved performance. However, such simplicity comes at a price: (i) the main weakness is that the default method does not take **environmental feedback** into account (*e.g.*, rewards received, verbal critique on responses, etc) and relies on the pre-defined meta prompt, thus the evolving may be less directional; we encourage practitioners to consider incorporating more richer feedback during evolving (one way to formulate this is by generative optimization (Yuksekgonul et al., 2024; Cheng et al., 2024; Nie et al., 2024)); (ii) another weakness is that existing method is single-shot (*i.e.*, we evolve based on a single $\mathbf{x}$ each time), thus the **diversity** of the generation may be limited – we anticipate future works improving this with multi-shot evolving by graph-based sampling. In this regard, the evolving process can be viewed as $\{\mathbf{x}'\}_{i=1}^{N} \leftarrow p_{\boldsymbol{\theta}}(\cdot \mid \{\mathbf{x}\}_{i=1}^{M}, \texttt{meta\_prompt}, \texttt{env\_feedback})$.

D.2 EXPERIMENTS ON NUMBER OF ITERATIONS

As an addition to § 4.2.4, we have experimented with the following settings:

- 10K prompts per iteration with 3 iterations.
- 20K prompts per iteration with 3 iterations (*i.e.*, all seed prompts are used).
- 60K prompts per iteration with 2 iterations (*i.e.*, all seed prompts are used).

Due to time constraints, we did not perform an extensive hyper-parameter search; however, we believe the results presented below sufficiently demonstrate the performance gains achieved by **eva**.

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO (10k)	51.6	651
$\boldsymbol{\theta}_{1\to2}$: DPO (10k)	59.8	718
$\boldsymbol{\theta}_{2\to3}$: DPO (10k)	61.2	802
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (10k)	60.1	733
$\boldsymbol{\theta}_{\tilde{1}\to\tilde{2}}$: + **eva** (10k)	62.0	787
$\boldsymbol{\theta}_{\tilde{2}\to\tilde{3}}$: + **eva** (10k)	62.2	774

Table 11: Results of using 10k prompts per iteration (DPO + length-penalized NLL loss).

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO (20k)	53.2	625
$\boldsymbol{\theta}_{1\to2}$: DPO (20k)	47.0	601
$\boldsymbol{\theta}_{2\to3}$: DPO (20k)	46.8	564
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (20k)	59.5	826
$\boldsymbol{\theta}_{\tilde{1}\to\tilde{2}}$: + **eva** (20k)	60.0	817
$\boldsymbol{\theta}_{\tilde{2}\to\tilde{3}}$: + **eva** (20k)	61.4	791

Table 12: Results of using 20k prompts per iteration (DPO + length-penalized NLL loss).

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) **/ Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to1}$: DPO (60k)	58.9	717
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + **eva** (60k)	59.6	725
$\boldsymbol{\theta}_{\tilde{1}\to\tilde{1}'}$: + **eva** (60k)	61.9	792

Table 13: Results of using 60k prompts per iteration (DPO + length-penalized NLL loss).

eva can bring robust gains with multiple iterations. As shown in Table 11, 12, and 13 below, our method presents persistent performance gain over iterations, and concretely surpasses the performance by default DPO training with true human prompts.

However, there exist diminishing marginal gains in iterative off-policy training. We ground **eva** in the iterative (off-policy) preference alignment paradigm due to its efficiency and ease of integration. However, such paradigms inherently face diminishing returns, where performance gains decrease with successive iterations, as previously observed in (Wu et al., 2024; Setlur et al., 2024; Yuan et al., 2024; Nikishin et al., 2022). While the generative data schedule in **eva** mitigates these challenges and extends beyond default training with human prompts (see also §4.2.4), the gains can weaken over iterations. We summarize potential reasons as: (i) the **off-policy signal decay** – as the number of examples increases, signals from the off-policy data become weaker due to distributional shift; (ii) the **loss of plasticity**, where the agent's ability to learn good policies decreases in continuing training with more iterations (Nikishin et al., 2022); (iii) the **ability of the solver** – as we evolve more harder prompts, it is harder for the solver to produce preferred response (thus more explicit reasoning techniques may be needed); (iv) the **ability of the reward model** to correctly provide reward signals to responses and thus informativeness signals to prompts, as there may exists distributional mismatch.

Thus, we envision future work to build on **eva** by: (i) exploring its integration with **on-policy RLHF** (*e.g.*, instead of evolving prompts in iterations, one may evolve in batches); (ii) **enhancing solver capabilities**, such as sampling more responses during inference or leveraging meta-instructions to guide deeper reasoning; (iii) online training of RM to co-evolve with the creator and the solver.

D.2.1 BONUS EXPERIMENTS ON `rewriter(·)` IN THE LOOP

Though beyond the current package, we present the basic idea here for practitioners to build upon. The motivation comes from the hypotheses derived from § D.2: as the prompts gets harder by evolving, there may be greater demands on the solver's capabilities *compared to earlier iterations*. As such, the solver may not be naively treated the same. One may address this by either inference-time scaling on responses or introducing meta-instructions to explicitly enhance the solver's reasoning.

We design a proof-of-concept experiment *w.r.t* the latter by adding **rewriter** in **`eva`**'s solver step. Previously, as in Algo. 1 and § 3.3.2, for each prompt $\mathbf{x}$, we generate multiple responses, and choose the best as $\mathbf{y}_+$ and the worst as $\mathbf{y}_-$ for preference optimization. Now, we add one more rewriting step that attempts to enhance $\mathbf{y}_+$ to be $\mathbf{y}'_+$, by applying a rewriting instruction (Liu et al., 2023b) that asks the solver to alter $\mathbf{y}_+$ with imporved helpfulness, relevance, reasoning depths, creativity and details while keeping the similar length. We then train with $(\mathbf{x}, \mathbf{y}'_+, \mathbf{y}_-)$ for preference optimization. Table 14 shows that adding the rewriter yields concrete performance gains over the default training method, while keeping the training budget and slightly increasing cost for offline data generation.

Model Family (→)	GEMMA-2-9B-IT	
Benchmark (→)	**Arena-Hard**	
Method (↓) / **Metric** (→)	**WR** (%)	**avg. len**
$\boldsymbol{\theta}_0$: SFT	41.3	544
$\boldsymbol{\theta}_{0\to 1}$: DPO	51.6	651
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + `eva`	**60.1**	733
$\boldsymbol{\theta}_{1\to\tilde{1}}$: + `eva` with `rewriter`	**61.9**	741

Table 14: Results of adding `rewriter` in the `solver` step.

E CURRICULUM VISUALIZATION OVER ITERATIONS

We now present initial observations supporting the intuition in § 3.4, where **`eva`** brings auto-curricula and the creator is incentivized to create new prompts that are both learnable and worth-learning.

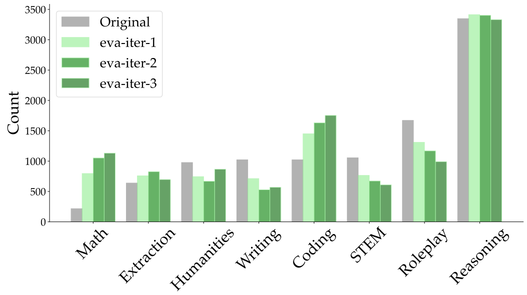

Figure 6: **Training distributions.** The prompt distribution of Table 11 for evolved prompts by zero-shot classification. **`eva`** creates a curriculum that prioritizes math / coding prompts over iterations.

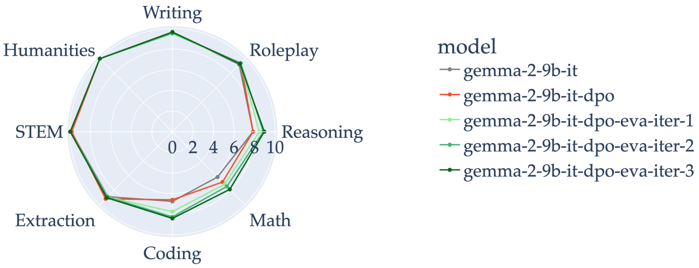

Figure 7: **Benchmark performance.** The radar figure for ratings on MT-Bench (Zheng et al., 2023), where each category contains ten problems. **`eva`** prioritizes and gradually improves on coding, math and reasoning over iterations, implicitly reflecting a learned curriculum.

E.1 COMPLEXITY AND QUALITY OF PROMPTS OVER ITERATIONS

Prompt Set (↓) **/ Metric** (→)	**Complexity** (1-5)	**Quality** (1-5)
UltraFeedback (seed)	2.90	3.18
UltraFeedback-**eva**-Iter-1	3.84	3.59
UltraFeedback-**eva**-Iter-2	3.92	3.63
UltraFeedback-**eva**-Iter-3	**3.98**	**3.73**

Table 15: **eva** improves prompt quality and complexity.

As in Table 15, there is a gradual improvement of prompt complexity and quality over iterations with **eva**. We sample 10K prompts per iteration, and use the below prompts modified from Liu et al. (2023b) for the complexity and quality evaluation, with `gemini-1.5-flash` as the scorer:

```
Rank the following questions according to their quality. Your evaluation should consider the following
↪  factors: Helpfulness, Relevance, Accuracy, Depth, Creativity, and Level of detail.
Score each response from 1 to 5: 1: Poor quality, 2: Below average, 3: Average, 4: Good, 5: Excellent.
```

```
Ranking the following questions according to their difficulty and complexity. Use a fixed scoring system: 1:
↪  Very simple, 2: Simple, 3: Moderate,  4: Difficult, 5: Very difficult
```

F VISUALIZATION ON PROMPT SELECTION METRIC

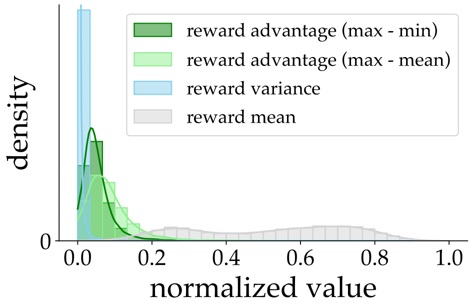

Figure 8: The probability density distributions of informativeness metrics compared in Table 3 – they show different patterns.

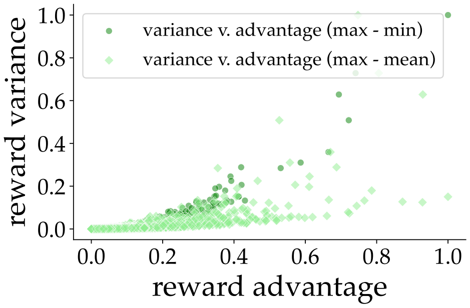

Figure 9: The correlation plot for reward advantage (ours) and reward variance – they are only *weakly* correlated.

In **eva**, we assign each prompt an informativeness value, which the creator will use as the weight to sample from the seed prompts for prompt synthesis. In § 4.2.1, we have shown that traditional methods like reward mean and reward variance are less effective as our advantage-based informativeness proxy. The intuition is simple: advantage/regret-based proxy aligns better with the preference optimization objective. We here further illustrate that they are statistically different from other choices:

- Figure 8: The distribution of informativeness values shows that reward variance is heavily concentrated at lower values, reward mean is more uniformly scattered, and reward advantage achieves a better balance, providing a broader yet also focused sampling range.
- Figure 9: The *weak correlation* between reward variance and reward advantage shows that variance *cannot* serve as a substitute for advantage as a proxy for informativeness.

We have discussed the contrastive curriculum hypothesis in § 3.4 to support using reward advantage in the sense that the induced samples tend to decrease the loss the most in the contrastive optimization. Furthermore, assuming the optimization algorithm can converge to the *more optimal* responses, neither reward mean nor variance directly capture the learning potential of such responses – one may easily construct cases with identical variance yet differ much in reward range – thus variance fails to distinguish such scenarios. By contrast, reward advantage estimate inherently captures the relative improvement towards better response, and is sensitive to differences in reward range; variants of advantage estimate are commonly used in literature, and we discuss underlying principles in § G.

G EXTENDED ILLUSTRATION ON THE METHODOLOGY

This is an extended version of § 3. In § G.1, we re-present the open-ended RLHF principle in Definition 1, and discuss the intuition under the KL regularization. In § G.2, we show heuristic approaches in open-ended learning to approximate this objective, with a focus on minimax game formulation. In § G.3, we formalize the regret objective in our RLHF setting, and discuss the regret minimization for the solver and the regret maximization for the creator.

G.1 THE CONCEPTUAL OPEN-ENDED RLHF FORMULATION

Classical RLHF optimizes over a static prompt set:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[\,r(\mathbf{x},\mathbf{y})\,\Big]-\mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\Big[\beta\cdot\mathbb{D}_{\mathrm{KL}}\Big[\,\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x})\;\|\;\pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\,\Big]\Big].$$

We propose to drop the static prompt set assumption, and jointly update the prompt distribution via a creator policy for Open-Ended RLHF, with the ideal objective below:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}} \mathbb{E}_{\mathbf{x}\sim\pi_{\boldsymbol{\phi}}(\cdot),\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})}\Big[\,r(\mathbf{x},\mathbf{y})\,\Big]-\beta\cdot\mathbb{D}_{\mathrm{KL}}\Big[\,\pi_{\phi}(\mathbf{x})\cdot\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x})\;\|\;p_{\mathrm{ref}}(\mathbf{x})\cdot\pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\,\Big].$$

This generalizes RLHF (Eq. 1). We can rewrite Eq. 7 with modified coefficients for precision:

$$\max_{\boldsymbol{\phi},\boldsymbol{\theta}}\ \mathbb{E}_{\mathbf{x}\sim\pi_{\phi}(\cdot)}\Big[\underbrace{\mathbb{E}_{\mathbf{y}\sim\pi_{\theta}(\cdot|\mathbf{x})}\Big[\,r(\mathbf{x},\mathbf{y})\,\Big]-\beta_1\cdot\mathbb{D}_{\mathrm{KL}}\big[\pi_{\theta}(\mathbf{y}\mid\mathbf{x})\;\|\;\pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\big]}_{\text{solver}}\Big]-\beta_2\cdot\underbrace{\mathbb{D}_{\mathrm{KL}}\Big[\,\pi_{\phi}(\mathbf{x})\;\|\;p_{\mathrm{ref}}(\mathbf{x})\,\Big]}_{\text{creator}}.$$

The newly proposed p_{ref} represents an *underspecified*, potentially intractable probability distribution over possible tasks in the **open-ended world** (instatiated *via* prompts) – it is *not* the initial static training prompt distribution (which is only the seed set for the creator to evolve upon); it can be seen serve as a conceptual guide to steer the prompt distribution.

To further clarify, there are two types of regularization in open-ended RLHF:

- $\mathbb{D}_{\mathrm{KL}}\left[\pi_{\theta}(\mathbf{y}\mid\mathbf{x})\;\|\;\pi_{\mathrm{SFT}}(\mathbf{y}\mid\mathbf{x})\right]$: this is the classical regularization on the response policy, ensuring that no matter how the training distribution over prompts evolves during optimization, the response policy remained anchored to the supervised fine-tuned (SFT) policy.
    - This KL (and preference optimization) is **explicitly achieved** in plug-in algorithms (*e.g.*, DPO) in Algo. 1. We later show how it relates to **solver's regret minimization**.
- $\mathbb{D}_{\mathrm{KL}}\Big[\,\pi_{\phi}(\mathbf{x})\;\|\;p_{\mathrm{ref}}(\mathbf{x})\,\Big]$: this probability matching term captures the intuition on optimizing $\pi_{\phi}(\mathbf{x})$ to approach the conceptualized $p_{\mathrm{ref}}(\mathbf{x})$, in the sense that a language model optimizes itself by adapting its training distributions with newly generated prompts for self-training to develop increasingly general capabilities, directing its learning towards informative, new tasks (Jiang, 2023), instead being constrained in a static, pre-defined set of tasks.
    - This conceptual KL is **implicitly achieved** by the creator step in the current **eva** setting by training on a *sequence of informative prompt sets*. We later show how it relates to **creator's regret maximization**. As illustrated in § 3.3.1, we start from the seed prompt set, choose those high-regret prompts and generate variations upon them by `EvolInstruct`, then mixing with a buffer of the original set to form the new training distribution at each iteration. This approach resembles classical open-ended learning in § G.2, and we hope it can serve as a small step for future works to build upon.
    - A common misunderstanding among readers may be to confuse the open-ended reference $p_{\mathrm{ref}}(\mathbf{x})$ with the initial seed prompt distribution $\mathcal{D}$, which is static. In contrast, $p_{\mathrm{ref}}(\mathbf{x})$ represents a broader space of tasks (*e.g.*, user prompts in the real wild world), as a conceptual target derived from the *underspecified distribution* (Dennis et al., 2020), *i.e.*, an environment with free parameters that control. Let's use an illustrative example with Fig. 6: the prompt distribution may be defined along several dimensions (*e.g.*, the number or complexity of coding problems); a potential creator can be designed to modify these dimensions, steering the initial $\mathcal{D}$ to new training distributions, by certain decision rules (*e.g.*, minimax regret, which offers worst-case guarantees) that forms *a sequence of informative prompts* for training.

This joint optimization objective only serves as a general principle. In the next, we discuss how existing works **implicitly achieve** the open-ended learning objective through **two-player games**.

G.2 APPROACHING OPEN-ENDED LEARNING BY UNSUPERVISED ENVIRONMENT DESIGN

G.2.1 THE ASYMMETRIC GAME FORMULATION FOR UNSUPERVISED ENVIRONMENT DESIGN

While we cannot directly train the agent with the intractable $p_{\text{ref}}(\mathbf{x})$ of the open-ended world, it is possible to curate a **curriculum of prompt distributions** to improve over the static distribution and support the *continual training* of the policy $\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{x})$, for it to keep improving and succeed over the full task space, thus conceptually approaching $p_{\text{ref}}(\mathbf{x})$. This is often framed as an **asymmetric two-player game**.

Dennis et al. (2020) first formally define this problem as Unsupervised Environment Design (UED). The idea is that while the real-world environments are inexhaustible and hard to tract, there may exist some free parameters (*e.g.*, height and roughness in a maze) which one may control to generate new environments; UED then concerns about designing a distribution of those free parameters (*i.e.*, settings) to create new fully specified environments, that can be used to train the agents.

In this setup, one player, the **creator**, generates new environments based on some specific decision rules (see the following), while the other player, the **solver**, optimizes its policy within these training environments, and the process continues iteratively. Common **heuristic strategies** include:

- **Randomization**: environments are generated uniformly and independently of the solver's current policy. This method is simple but less effective (Tobin et al., 2017).
- **Maximin**: the creator generates environments that minimize the solver's maximum possible reward, which can often lead to unsolvable scenarios (Khirodkar and Kitani, 2018).
- **Minimax regret**: The creator targets environments that maximize the solver's *regret*, defined as the difference between the optimal return achievable and that of the solver's current policy (Beukman et al., 2024b). The regret is often conceived as the **creator's utility**.

Among them[4], the minimax regret approach presents a sweet spot where the creator can create hard yet solvable environments, and is often empirically better. The minimax regret strategy also implies that the agent's policy is trained to perform well under all levels/settings, thus enjoys a worst-case guarantee. However, while it is often straightforward for the solver to minimize the regret (*e.g.*, through direct policy optimization, as we discuss in § G.3), the optimal policy remains *unknown* during the optimization process, thus regret as the decision signal is often intractable to the creator – which requires *approximation* (as an amusing side note, this is described as the Achilles' heel of those curriculum RL methods by Parker-Holder et al. (2022)).

G.2.2 APPROXIMATING THE REGRET AND GENERATING NEW ENVIRONMENTS

In general, the **creator** design in this line of research contains two steps:

1. **identifying high-regret levels** using different (often heuristic) regret approximation;
2. **generating new environments** by making variations or retrieving from buffers on those high-regret levels.

We hereby review major works on regret approximation and environment generation as follows:

Dennis et al. (2020) propose joint training for the creator and two competing solvers.

- **Regret approximation**: here, two solver policies are trained, with the regret approximated as the **difference in their returns**. During each optimization step, one solver *maximizes* this regret, the other *minimizes* it, and the creator maximizes it.
- **Environment generation**: the system directly sample the parameter from the creator policy and use that to specify the environment.

---

[4] We have implemented variants of these in § 4.2.1, and show minimax regret is empirically better.

Jiang et al. (2021b) propose to random sampling on high-regret levels.

- **Regret approximation**: as a heuristic, the authors use *positive value loss*, which is a function of Generalized Advantage Estimate (Schulman et al., 2015) (which itself is a function of the TD error – the difference between the expected and the actual returns) as the creator's utility.
- **Environment generation**: the creator have a rolloing buffer of highest-regret levels by random searching on relevant configurations.

Jiang et al. (2021a) further propose a double-creator setting based on (Jiang et al., 2021b), where one creator is actively generating new environments, and the other is retrieving from the buffer.

Parker-Holder et al. (2022) propose to sample high-regret levels and generate new environments by making *edits* on existing ones. The regret approximation is the same as (Jiang et al., 2021b) – the positive value loss. For the environment generation, the authors suggest a general editing/mutation mechanism, where the creator chooses from high-regret levels and make small variations within an edit distance, which by heuristics will lead to the discovery of more high-regret environments. There is an additional filtering step: they do not directly train on the newly generated levels, but evaluate on those levels first, then add only the high-regret ones to the training buffer.

Note the solvers are often directly trained with PPO (Schulman et al., 2017) under the environments.

G.3 REGRET FORMULATION FOR OPEN-ENDED RLHF

Next, we discuss the regret minimization and maximization in our setting for alignment. Specifically,

- **Regret minimization for the solver**: we avoid calculating regret and use direct policy optimization (*e.g.*, DPO) to equivalently achieve regret minimization.
- **Regret maximization for the creator**: similarly to (Jiang et al., 2021b; Parker-Holder et al., 2022), we first find an approximation of regret, then curate new environments for the solver to train on by (i) sampling from a replay buffer of existing prompts, and (ii) making variations (through `EvolInstruct` (Xu et al., 2023a)) on those high-regret prompts. Specifically, we use **advantage-based estimates of the current policy**, as summarized in Table 2.

This asymmetric two-player game serves as one potential modeling choice to implicitly achieve the open-ended RLHF principle that we proposed in Definition 1. We look forward to exploring more principled solutions in the future.

Preliminaries. Let $r(\cdot,\cdot)$ be an oracle reward model. The (unregularized) optimal policy is:

$$\pi^\star = \arg\max_{\pi} \mathbb{E}_{\mathbf{x}\sim\mathcal{D},\mathbf{y}\sim\pi(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big].$$

We have the *optimal advantage* / the *negated regret* as:

$$\begin{aligned} A^\star(\mathbf{x},\mathbf{y}) &= r(\mathbf{x},\mathbf{y}) - \mathbb{E}_{\mathbf{y}'\sim\pi^\star(\cdot|\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}') \,\Big] \\ &= r(\mathbf{x},\mathbf{y}) - V^\star(\mathbf{x},\mathbf{y}). \end{aligned}$$

Classical preference-based RL assumes a *reward*-based preference model, that is:

$$P\left(\mathbf{y}^+ \succeq \mathbf{y}^-\right) = \frac{\exp\big(r\left(\mathbf{x},\mathbf{y}^+\right)\big)}{\exp\big(r\left(\mathbf{x},\mathbf{y}^+\right)\big) + \exp\big(r\left(\mathbf{x},\mathbf{y}^-\right)\big)}.$$

As a side note (Hejna et al., 2023), this is equivalent to the *advantage/regret*-based preference model, due to the bandit setup in RLHF:

$$\begin{aligned} P\left(\mathbf{y}^+ \succeq \mathbf{y}^-\right) &= \frac{\exp\big(r\left(\mathbf{x},\mathbf{y}^+\right) - V^\star(\mathbf{x},\mathbf{y})\big)}{\exp\big(r\left(\mathbf{x},\mathbf{y}^+\right) - V^\star(\mathbf{x},\mathbf{y})\big) + \exp\big(r\left(\mathbf{x},\mathbf{y}^-\right) - V^\star(\mathbf{x},\mathbf{y})\big)} \\ &= \frac{\exp\big(A^\star\left(\mathbf{x},\mathbf{y}^+\right)\big)}{\exp\big(A^\star\left(\mathbf{x},\mathbf{y}^+\right)\big) + \exp\big(A^\star\left(\mathbf{x},\mathbf{y}^-\right)\big)}. \end{aligned}$$

In our current setting, we assume there is an oracle preference model for the preference pair labeling.

KL-regularized regret. In the RLHF setting at fixed prompt distribution, the objective is:

$$\max_{\pi_{\boldsymbol{\theta}}} \mathbb{E}_{\mathbf{x}\sim\pi_{\phi}(\cdot),\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\Big[\beta\cdot\pi_{\phi}(\cdot)_{\mathrm{KL}}\Big[\,\pi_{\boldsymbol{\theta}}(\mathbf{y}\mid\mathbf{x}) \parallel \pi_{\text{SFT}}(\mathbf{y}\mid\mathbf{x})\,\Big]\Big].$$

The optimal policy of the above KL-constrained objective is:

$$\pi^{\star}_{\mathrm{KL}}(\mathbf{y}\mid\mathbf{x}) = \frac{1}{Z(\mathbf{x})}\pi_{\text{SFT}}(\mathbf{y}\mid\mathbf{x})\exp\left(\frac{1}{\beta}\cdot r(\mathbf{x},\mathbf{y})\right),$$

where the partition function is defined as $Z(\mathbf{x}) = \sum_{\mathbf{y}} \pi_{\text{ref}}(\mathbf{y}\mid\mathbf{x})\exp\left(\frac{1}{\beta}r(\mathbf{x},\mathbf{y})\right)$.

We can now formally define the *regret* with regard to $\pi^{\star}_{\mathrm{KL}}(\cdot\mid\mathbf{x})$ as:

$$\text{Regret}_{\mathrm{KL}}(\mathbf{x},\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathbf{y}\sim\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big] - \mathbb{E}_{\mathbf{y}\sim\pi^{\star}_{\mathrm{KL}}(\cdot\mid\mathbf{x})}\Big[\, r(\mathbf{x},\mathbf{y}) \,\Big]. \tag{14}$$

Regret Minimization for the Solver. It is rather straightforward/trivial to understand the objective of the solver $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ as regret minimization, since the goal is to align the policy $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ with the KL-optimal solution $\pi^{\star}_{\mathrm{KL}}(\cdot\mid\mathbf{x})$, which directly minimizes the KL-regularized regret by design. This formulation allows flexibility in the plug-in preference optimization algorithms for the solver's step in Algorithm 1, and ensures *the alignment problem is well-defined*. In practice, we use Direct Preference Optimization (DPO) and its variants, which approximate the KL-optimal solution by iteratively adjusting $\pi_{\boldsymbol{\theta}}$ to reflect preference differences.

Regret Maximization for the Creator. As discussed previously, while it is often trivial for the solver to minimize the regret through direct policy optimization, the optimal policy remains unknown during the optimization process, thus we cannot directly calculate the regret – we must approximate it when using it as the utility for the creator. Similarly to heuristics proposed by prior works (Jiang et al., 2021b;a; Parker-Holder et al., 2022), we use the advantage-based estimate:

$$|\hat{\text{Regret}}(\mathbf{x},\pi_{\boldsymbol{\theta}})| \leftarrow \text{info}_{\boldsymbol{\theta}}(\mathbf{x}) := r(\mathbf{x},\mathbf{y}_{+}) - r(\mathbf{x},\mathbf{y}_{\text{baseline}}), \tag{15}$$

where

$$\mathbf{y}_{+} := \arg\max\nolimits_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y}), \tag{16}$$

$$\mathbf{y}_{\text{baseline}} := \arg\min\nolimits_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y}) \text{ or } \mathbf{y}_{\text{baseline}} := \text{avg}_{\mathbf{y}_i}\ r(\mathbf{x},\mathbf{y}), \tag{17}$$

and $\{\mathbf{y}_i\}_{i=1}$ is a set of responses sampled from $\pi_{\boldsymbol{\theta}}(\cdot\mid\mathbf{x})$ and $r(\cdot,\cdot)$ is the reward oracle. We use $\arg\min_{\mathbf{y}_i} r(\mathbf{x},\mathbf{y})$ by default due to its simplicity and efficiency (see also § 3.4 for additional interpretation) and consistent strong empirical gains we observed in vast experiments. As the policy optimizes, the proxy should approximate the true regret better. We leave exploration of other informativeness proxy designs in `eva` to future work.

For new environment generation, as illustrated in § 3.3.1, we start from the seed prompt set, choose those high-regret prompts and generate variations upon them by `EvolInstruct`, then mixing with a buffer of the original set to form the new training distribution at each iteration.

H EXTENDED LITERATURE REVIEW FOR OPEN-ENDED LEARNING

The design of our game-theoretic framework for language model post-training is inspired from many prior works in open-ended learning. The central idea of open-ended learning is *not* to optimize for a *specific, static* distribution, but to develop an agent that can *generalize* well across *unseen, novel* environments, which are the environments that the agent has not been explicitly trained on. To achieve this, unsupervised environment design proposes to generate environments that present a curriculum of *increasing complexity* for the agent to evolve, which ensures that the agent's learning is not *narrow*, but broad enough to handle the diversity of complexity of future environments. In such curriculum, as the agent solves simpler environments, it moves on to more difficult ones, thus progressively builds more sophisticated strategies. Furthermore, by adopting a *minimax regret* framework, this approach adds a layer of robustness by minimizing the agent's performance gap in worst-case (*i.e.*, most adversarial) environments. In addition to distinctions discussed in § 5, we here list several foundational works in this line, and encourage the LLM community to explore with more rigor and depth: Schmidhuber (1991) presents an initial investigation into open-ended learning via self-supervised curiosity-driven exploration; Wang et al. (2019) emphasize co-evolution of environments and agent policies by training a population of agents that adapt to and solve progressively complex challenges; Dennis et al. (2020) formally introduce the notion of Unsupervised Environment Design (UED), where a protagonist and antagonist agent pair simulates regret by competing in shared environments, driving the protagonist (the main learner) to adapt to increasingly challenging scenarios; Jiang et al. (2021b) introduce Prioritized Level Replay (PLR), which uses a rolling buffer of high-regret levels to dynamically adjust the training curriculum, and selects levels with the higher learning potential; Parker-Holder et al. (2022) further propose improvements by editing previously high-regret levels; Hughes et al. (2024b) present a formal definition for open-ended system with respect to *novelty* and *learnability*, that generalizes various systems, *e.g.*, AlphaGo (Silver et al., 2016), AdA (Team et al., 2021), etc.

I EXTENDED LITERATURE REVIEW IN BI-LEVEL RLHF

Bi-level optimization refers to optimization problems where the cost function is defined *w.r.t.* the optimal solution to another optimization problem (Grosse, 2022). There is a recent line of works applying bi-level optimization to RLHF. While they all rely on a fixed dataset of prompts, **`eva`** propose to dynamically update the prompt set, as in § 1. We present a detailed comparison of **`eva`** with Ding et al. (2024); Shen et al. (2024); Makar-Limanov et al. (2024). We sincerely thank the anonymous reviewer for the kind references, and welcome suggestions for any other works we may have missed.

Ding et al. (2024) formulate iterative online RLHF as a bi-level optimization problem, where the upper-level represents the reward learning, and the lower-level represents the policy optimization. Leveraging reward re-parameterization tricks in Rafailov et al. (2023), Ding et al. (2024) reduces the problem to a single-level objective with regard to the policy. The differences of this work and our work lie in the prompt distribution and preference oracle: (i) **`eva`** features by **dynamic prompt set generation for Open-Ended RLHF**, whereas (Ding et al., 2024) remains using a static prompt set; (ii) we assume the existence of the preference oracle (as discussed in § 4), while Ding et al. (2024) consider online training of reward models and ablate on self-rewarding by the current LLM policy. Our usage of a pre-trained reward model follows from industrial practices (Team et al., 2023; 2024b), which is also commonly used by prior works in academia (Meng et al., 2024; Wu et al., 2024).

Makar-Limanov et al. (2024) provide an interesting exploration on formulating RLHF as a leader-follower game, where the language model (LM) policy is the leader and the reward model (RM) policy is the follower, and the solution is **Stackelberg equilibrium** (von Stackelberg, 1934; Rajeswaran et al., 2020), where the *leader does not likewise best respond to the follower's strategy*. Here, following the curriculum RL literature (Dennis et al., 2020; Parker-Holder et al., 2022), we seek the **Nash equilibrium** (Nash et al., 1950) between the creator for prompt generation and the solver for response generation. In the current setting of **`eva`**, the goal is to search for an optimal solver policy with a best supporting prompt distribution, *and* an optimal prompt distribution with a best supporting solver policy. Nevertheless, the LM-RM iterative optimization may be added on top of **`eva`**'s framework, and we look forward to future works exploring the leader-follower re-formulation of **`eva`**.

Shen et al. (2024) present a rigorous theoretical work (though it does not directly involve practical post-training of large language models). The authors propose to reduce the bi-level problem to a single-level problem with a penalty-based reformulation, and apply it in the setting of LM-RM optimization within a *fixed* environment, whereas **`eva`** focuces on dynamic prompt generation and practically train large language models with extensive empirical experiments conducted. We believe it would be interesting to adapt similar first-order optimization techniques to solve Open-Ended RLHF.

In summary, existing bi-level RLHF works focus on online optimization of both the RM and the LM (as the response policy), all with **fixed** prompt/state distribution. **`eva`** presents an orthogonal direction on **dynamic** prompt generation for Open-Ended RLHF, with an empirical algorithm which attains state-of-the-art performance with large language models on a variety of benchmarks. It is possible to incorporate the online RM training within **`eva`** – we have shown in § 4.2.3 that **`eva`** scales with quality of reward models, thus integrating online RM training may further enhance performance and mitigate potential distributional mismatch problems as we evolves for more prompts. This direction may have not been widely adopted in real-world training of language models, due to concerns about practicality (Team et al., 2023; 2024a;b; Adler et al., 2024). We look forward to future works exploring *efficient* variations unifying **`eva`** and existing bi-level RM-LM frameworks.

J EXAMPLES ON PROMPTS AND MODEL GENERATIONS

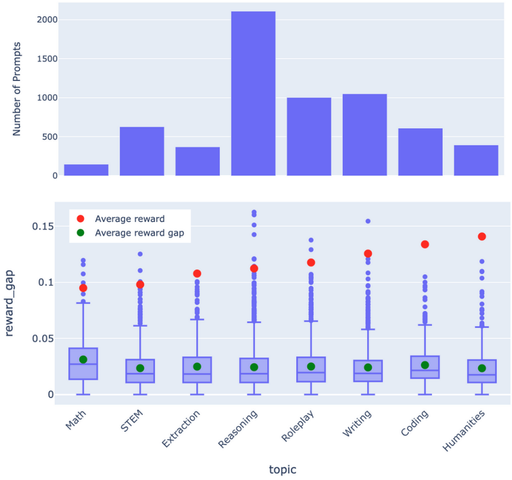

Figure 10: The initial prompt distribution of AlpacaEval by `bart-large-mnli` with 0-shot classification, which is imbalanced. For the reward distribution, the category with lowest average reward has the highest reward gap (*i.e.*, the default informativeness proxy), implying the potential to improve.

initial prompt →	Write me the code for a distributed transaction manager.\nThink step by step and use pseudo code first.\nThen, define interfaces for all involved actors and entities.\nUse Rational Unified approach for this part.\n\nOnly then move on to the actual implementation, class-by-class, and method-by-method.\nMake the code be implemented in C# and follow SOLID principles.
evolved #1 →	Craft a suite of syntax for a distributed transaction coordinator. Start with a sequential breakdown in pseudocode format. Following that, establish the protocols for communication and interaction amongst the various participants and components, incorporating the Rational Unified Process methodology.\n\nProceed thereafter to the concrete creation, detailing each class and function. Ensure that the final C# code adheres to the precepts of SOLID and is annotated for clarification and maintainability purposes.
evolved #2 →	Devise a comprehensive set of directives and structures for a distributed transaction coordinator architecture. Initiate by formulating a step-by-step algorithmic decomposition in pseudocode. Subsequently, delineate the frameworks for dialog and cooperation between the numerous entities and elements, utilizing the tenets of the Rational Unified Process methodology.\n\nContinue to the actual generation of the code, meticulously detailing every class and corresponding method. Guarantee that the culminating C# code is in strict compliance with SOLID principles and is supplemented with descriptive commentary to enhance future clarity and upkeep, while also validating the code against a set of unit tests to ensure robust functionality.
evolved #3 →	Commence by constructing an algorithm in pseudocode that meticulously breaks down the sequential stages for a distributed transaction coordinator system. Progress by outlining the communication protocols and collaborative mechanisms among the myriad components, adhering to the Rational Unified Process approach.\n\nAdvance to crafting the code in C#, ensuring each class and method is elaborated with precision, aligning with SOLID design principles. Integrate thorough comments for heightened maintainability and conduct comprehensive unit testing for assured operational integrity.
evolved #4 →	Design a detailed flowchart that represents the intricate steps of an orchestration engine for managing microservices in a cloud-native environment. Begin by detailing the service discovery, load balancing, and fault tolerance strategies, while incorporating the DevOps philosophy.\n\nProceed by developing the implementation in Java, focusing on clean, modular code following object-oriented programming best practices. Accompany your code with clear documentation for easy understanding and perform rigorous integration testing to guarantee robustness.

Table 16: Examples of evolved prompts from AlpacaEval. We suggest practitioners to include generated responses with oracle rewards and self-critique in the context for better prompt evolving, which is helpful for mitigating the effect of potential self-consuming loops in synthetic data training.

1st-turn prompt: When a number is divided by 10, the remainder is 4. What is the remainder when twice the number is divided by 4?	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
Here's how to solve this problem: 1. Understand the first statement: * "When a number is divided by 10, the remainder is 4" means the number can be represented as 10*k + 4, where k is an integer. 2. Double the number: * Twice the number is 2*(10*k + 4) = 20*k + 8 3. Divide by 4: * (20*k + 8) divided by 4 is (20*k)/4 + 8/4 = 5*k + 2 4. Find the remainder: * Since 5*k is a multiple of 4, the remainder when twice the number is divided by 4 is 2. Let me know if you'd like to try another example!	Here's how to solve this problem: 1. Understand the first statement: - "When a number is divided by 10, the remainder is 4" means the number can be represented as 10*k + 4, where k is an integer. 2. Double the number: - Twice the number is 2*(10*k + 4) = 20*k + 8 3. Divide by 4: - When (20*k + 8) is divided by 4, the 20*k term is perfectly divisible by 4. - The remainder will be the same as the remainder when 8 is divided by 4, which is 0. Therefore, the remainder when twice the number is divided by 4 is 0.
2nd-turn prompt: What about when twice the number is divided by 5?	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
Here's how to solve that: 1. We know: Twice the number is 20*k + 8 (from the previous calculation). 2. Divide by 5: (20*k + 8) divided by 5 is (20*k)/5 + 8/5 = 4*k + 1.6 3. Find the remainder: Since 4*k is a multiple of 5, the remainder when twice the number is divided by 5 is 1. Let me know if you have any more number theory puzzles!	Here's how to solve that: 1. Twice the number: We know twice the number is 20*k + 8 (from the previous steps). 2. Divide by 5: * 20*k is divisible by 5. * 8 divided by 5 leaves a remainder of 3. Therefore, the remainder when twice the number is divided by 5 is 3.
Judgment (by gpt-4)	
Judgment for gemma-2-9b-it-dpo	**Judgment for gemma-2-9b-it-dpo-eva**
The assistant's response is incorrect. The assistant incorrectly calculated the remainder when twice the number is divided by 5. The assistant calculated the decimal part of the division operation, not the remainder. The correct calculation should be (20n + 8) % 5 = (20n % 5) + (8 % 5) = 0 + 3 = 3. Therefore, the remainder when twice the number is divided by 5 is 3, not 1 as the assistant stated. The corrected response would be: 1. We know: Twice the number is 20*k + 8 (from the previous calculation). 2. Divide by 5: (20*k + 8) divided by 5 gives us a remainder of (20*k % 5) + (8 % 5) = 0 + 3 = 3. 3. Find the remainder: The remainder when twice the number is divided by 5 is 3. Rating: [[3]], "score": 3, "turn": 2, "tstamp": 1722737011.7768948	The assistant's answer is correct and matches the reference answer. The assistant correctly explains that twice the number is 20*k + 8 and that when this is divided by 5, the remainder is 3. The assistant's explanation is clear and easy to understand, making it helpful for the user. There are no mistakes to correct. Rating: [[10]], "score": 10, "turn": 2, "tstamp": 1722846518.5543818

Table 17: Comparison of model generations on examples from MT-Bench.

1st-turn prompt: Imagine you are writing a blog post comparing two popular smartphone models. Develop an outline for the blog post, including key points and subheadings to effectively compare and contrast the features, performance, and user experience of the two models. Please answer in fewer than 200 words.	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
## Smartphone Showdown: [Model A] vs. [Model B]\n\n**I. Introduction**\n* Briefly introduce the two smartphone models being compared, highlighting their popularity and target audience.\n* State the purpose of the comparison: to help readers decide which phone best suits their needs.\n\n**II. Design and Display**\n* Physical dimensions, weight, and build quality comparison.\n* Screen size, resolution, and display technology (e.g., OLED, LCD).\n* Design aesthetics and user preferences.\n\n**III. Performance and Specs**\n* Processor, RAM, and storage comparison.\n* Benchmark scores and real-world performance analysis.\n* Battery life and charging capabilities.\n\n**IV. Camera Capabilities**\n* Rear and front camera specs (megapixels, aperture, features).\n* Image quality comparison in various lighting conditions.\n* Video recording capabilities.\n\n**V. Software and User Experience**\n* Operating system and user interface comparison.\n* App ecosystem and availability.\n* Software features and customization options.\n\n**VI. Price and Value**\n* Compare the price points of both models.\n* Analyze the value proposition based on features and performance.\n\n**VII. Conclusion**\n* Summarize the key differences and similarities.\n* Recommend the best phone for specific user profiles (e.g., budget-conscious, photography enthusiasts, gamers).	## iPhone 14 Pro vs. Samsung Galaxy S23: Which Flagship Reigns Supreme?\n\n**I. Introduction**\n* Briefly introduce both smartphones and their position in the market.\n* State the purpose of the comparison: to help readers decide which phone best suits their needs.\n\n**II. Design and Display**\n* Physical dimensions, weight, and build quality comparison.\n* Screen size, resolution, refresh rate, and display technology (OLED vs. Super AMOLED).\n* Design aesthetics and unique features (e.g., notch vs. punch-hole, color options).\n\n**III. Performance and Camera**\n* Processor comparison (Apple A16 Bionic vs. Snapdragon 8 Gen 2).\n* RAM and storage options.\n* Camera specs: megapixels, lens types, aperture, and special features (e.g., telephoto, macro).\n* Image quality comparison based on sample photos and user reviews.\n\n**IV. Software and User Experience**\n* Operating systems: iOS 16 vs. Android 13.\n* User interface comparison: intuitiveness, customization options, and app ecosystems.\n* Special features: Face ID vs. in-display fingerprint sensor, voice assistants.\n\n**V. Battery Life and Charging**\n* Battery capacity and real-world usage comparison.\n* Fast charging speeds and wireless charging capabilities.\n\n**VI. Price and Value**\n* Compare starting prices and different storage configurations.\n* Assess the overall value for money based on features and performance.\n\n**VII. Conclusion**\n* Summarize the key differences and strengths of each phone.\n* Recommend the best phone for specific user types (e.g., photography enthusiasts, gamers, budget-conscious consumers).
2nd-turn prompt: Take your previous response and rephrase it as a limerick.	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
Two phones, a showdown so grand,\n[Model A] and [Model B], hand in hand.\nDesign, specs, and price,\nWe’ll analyze nice,\nTo help you choose, understand?\n\nLet me know if you’d like me to try filling in the bracketed parts with specific model names!	\nTwo phones, a Pro and a Galaxy,\nFor flagships, they truly are spacy.\nWith cameras and screens,\nAnd processors keen,\nWhich one wins? Read on, and be savvy! \nLet me know if you’d like another limerick!
Judgment (by gpt-4)	
Judgment for gemma-2-9b-it-dpo	**Judgment for gemma-2-9b-it-dpo-eva**
The assistant’s response is creative and relevant to the user’s request. The user asked the assistant to rephrase the previous response, which was a detailed outline for a blog post, into a limerick. The assistant successfully did so, maintaining the essence of the original response in a limerick format. The assistant also offered to further customize the limerick by filling in the bracketed parts with specific model names, showing a willingness to provide additional detail if needed. However, the limerick does not fully capture the depth and detail of the original response, which is understandable given the constraints of the limerick format. Rating: [[8]], "score": 8, "turn": 2, "tstamp": 1722737024.0327556	The assistant’s response is creative and relevant to the user’s request. The assistant successfully rephrased the previous response, which was a detailed outline for a blog post comparing two smartphone models, into a limerick. The limerick is playful and engaging, and it maintains the essence of the original response, which is to compare the two phones and help the reader make an informed decision. The assistant also offers to create another limerick if the user wants, showing a willingness to continue the interaction. Rating: [[9]], "score": 9, "turn": 2, "tstamp": 1722846542.9437995

Table 18: (Continued) comparison of model generations on examples from MT-Bench.

1st-turn prompt: Each problem consists of three statements. Based on the first two statements, the third statement may be true, false, or uncertain.\n1. Oranges cost more than apples.\n2. Oranges cost less than bananas.\n3. Bananas cost more than apples and bananas cost more than orange.\nIf the first two statements are true, then the third statement is?	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
Here’s how to break down the logic:\n\n* **Statement 1:** Oranges > Apples (Oranges cost more than apples)\n* **Statement 2:** Oranges < Bananas (Oranges cost less than bananas)\n\nThis means the price order is: Bananas > Oranges > Apples\n\n* **Statement 3:** Bananas > Apples AND Bananas > Oranges\n\nStatement 3 is **partially true**.\n\n* The part "Bananas > Apples" is true based on our price order.\n* The part "Bananas > Oranges" is also true based on our price order.\n\n\nLet me know if you’d like to try another one!	Here’s how to break down the logic:\n\n* **Statement 1:** Oranges > Apples (Oranges cost more than apples)\n* **Statement 2:** Oranges < Bananas (Oranges cost less than bananas)\n\nThese statements tell us that bananas are the most expensive, then oranges, and apples are the least expensive.\n\n* **Statement 3:** Bananas > Apples AND Bananas > Oranges\n\nSince bananas are indeed more expensive than both apples and oranges, statement 3 is **true**.\n\n\nLet me know if you’d like to try another one!
2nd-turn prompt: If the third statement is true. Is the first statement true, false, or uncertain? Please explain.	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
If the third statement is true, the first statement is **false**.\n\nHere’s why:\n\n* **Statement 3:** States that bananas cost more than both apples AND oranges.\n\nThis means the price order is: Bananas > Oranges > Apples.\n\n* **Statement 1:** States that oranges cost more than apples.\n\nIf bananas are the most expensive, and oranges are second, then oranges cannot be more expensive than apples.\n\n\nLet me know if you have any other logic puzzles!	If the third statement is true, then the first statement is **uncertain**.\n\nHere’s why:\n\n* **Statement 3:** Bananas > Apples AND Bananas > Oranges\n\nThis tells us bananas are the most expensive, but it doesn’t explicitly state the relative prices of apples and oranges.\n\n* **Statement 1:** Oranges > Apples\n\nThis statement claims oranges are more expensive than apples.\n\nStatement 3 doesn’t contradict statement 1, but it also doesn’t confirm it. It’s possible that oranges *are* more expensive than apples, but statement 3 alone doesn’t give us enough information to be sure.
Judgment (by gpt-4)	
Judgment for gemma-2-9b-it-dpo	**Judgment for gemma-2-9b-it-dpo-eva**
The assistant’s response is incorrect. The assistant states that if the third statement is true, the first statement is false. However, the third statement only establishes that bananas cost more than both apples and oranges. It does not provide information about the relative costs of apples and oranges. Therefore, the first statement could be true (if oranges cost more than apples), false (if oranges cost less than apples), or uncertain (if oranges and apples cost the same). The assistant’s explanation is flawed because it incorrectly assumes that the order of costs must be Bananas > Oranges > Apples, when in fact the order could also be Bananas > Apples = Oranges or Bananas > Apples > Oranges.\n\nCorrected Assistant’s Answer: If the third statement is true, then bananas cost more than apples and bananas cost more than oranges. This means that oranges could still cost more than apples, making the first statement true. However, it’s also possible that oranges and apples have the same cost, which would make the first statement false. Therefore, based on the third statement alone, the first statement is uncertain. Rating: [[2]], "score": 2, "turn": 2, "tstamp": 1722737017.2808955	The assistant’s response is correct and helpful. The assistant explains that the first statement is uncertain if the third statement is true, which aligns with the reference answer. The assistant provides a clear explanation of why this is the case, stating that the third statement doesn’t contradict the first statement, but it also doesn’t confirm it. Therefore, the assistant’s response is accurate and informative.\n\nThe assistant’s response: "If the third statement is true, then the first statement is **uncertain**. Here’s why: Statement 3: Bananas > Apples AND Bananas > Oranges. This tells us bananas are the most expensive, but it doesn’t explicitly state the relative prices of apples and oranges. Statement 1: Oranges > Apples. This statement claims oranges are more expensive than apples. Statement 3 doesn’t contradict statement 1, but it also doesn’t confirm it. It’s possible that oranges *are* more expensive than apples, but statement 3 alone doesn’t give us enough information to be sure. Rating: [[10]], "score": 10, "turn": 2, "tstamp": 1722846527.2339694

Table 19: (Continued) comparison of model generations on examples from MT-Bench.

1st-turn prompt: Implement a function to find the median of two sorted arrays of different sizes with O(1) space complexity and O(n) time complexity.	
Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
python\n\def find_median_sorted_arrays(nums1, nums2):\n """\n Finds the median of two sorted arrays of different sizes.\n\n Args:\n nums1: The first sorted array.\n nums2: The second sorted array.\n\n Returns:\n The median of the two sorted arrays.\n """\n\n m, n = len(nums1), len(nums2)\n if m > n:\n nums1, nums2, m, n = nums2, nums1, n, m # Ensure m <= n\n\n imin, imax, half_len = 0, m, (m + n + 1) // 2\n\n while imin <= imax:\n i = (imin + imax) // 2\n j = half_len - i\n if i < m and nums2[j - 1] > nums1[i]:\n imin = i + 1 # i is too small, increase it\n elif i > 0 and nums1[i - 1] > nums2[j]:\n imax = i - 1 # i is too big, decrease it\n else:\n # i is perfect\n\n if i == 0:\n max_left = nums2[j - 1]\n elif j == 0:\n max_left = nums1[i - 1]\n else:\n max_left = max(nums1[i - 1], nums2[j - 1])\n\n if (m + n) % 2 == 1:\n return max_left\n\n if i == m:\n min_right = nums2[j]\n elif j == n:\n min_right = nums1[i]\n else:\n min_right = min(nums1[i], nums2[j])\n\n return (max_left + min_right) / 2.0\n\n# Example usage\nnums1 = [1, 3]\nnums2 = [2]\nmedian = find_median_sorted_arrays(nums1, nums2)\nprint(f"The median of {nums1} and {nums2} is: {median}")\n\n\n**Explanation:**\n\n1. **Handle Unequal Lengths:**\n - If ‘nums1‘ is longer than ‘nums2‘, swap them to ensure ‘nums1‘ is always the shorter array. This simplifies the logic.\n\n2. **Binary Search:**\n - The core idea is to use binary search on the shorter array (‘nums1‘).\n - ‘imin‘ and ‘imax‘ define the search range for the partition point ‘i‘ in ‘nums1‘.\n - ‘half_len‘ calculates the desired total number of elements on the left side of the partition (to find the median).\n\n3. **Partitioning:**\n - In each iteration, ‘i‘ divides ‘nums1‘ into two parts: ‘nums1[0:i]‘ and ‘nums1[i:m]‘.\n - ‘j‘ is calculated to divide ‘nums2‘ so that the total number of elements on the left side is ‘half_len‘.\n\n4. **Adjusting the Partition:**\n - We compare elements at the boundaries of the partitions to see if ‘i‘ is in the correct position:\n - If ‘nums2[j - 1] > nums1[i]‘, ‘i‘ is too small, so we move ‘imin‘ to ‘i + 1‘.\n - If ‘nums1[i - 1] > nums2[j]‘, ‘i‘ is too large, so we move ‘imax‘ to ‘i - 1‘.\n\n5. **Finding the Median:**\n - Once ‘i‘ is in the correct position, we find:\n - ‘max_left‘: The maximum element in the left partitions of both arrays.\n - ‘min_right‘: The minimum element in the right partitions of both arrays.\n - If the total length is odd, ‘max_left‘ is the median.\n - If the total length is even, the median is the average of ‘max_left‘ and ‘min_right‘.\n\n**Time Complexity:** O(log(min(m, n))) because we perform binary search on the shorter array.\n\n**Space Complexity:** O(1) as we only use a constant amount of extra space for variables.	python\n\def find_median_sorted_arrays(nums1, nums2):\n """\n Finds the median of two sorted arrays of different sizes.\n\n Args:\n nums1: The first sorted array.\n nums2: The second sorted array.\n\n Returns:\n The median of the two sorted arrays.\n """\n\n m, n = len(nums1), len(nums2)\n if m > n:\n nums1, nums2, m, n = nums2, nums1, n, m # Ensure m <= n\n\n imin, imax, half_len = 0, m, (m + n + 1) // 2\n\n while imin <= imax:\n i = (imin + imax) // 2\n j = half_len - i\n if i < m and nums2[j - 1] > nums1[i]:\n imin = i + 1 # i is too small, increase it\n elif i > 0 and nums1[i - 1] > nums2[j]:\n imax = i - 1 # i is too big, decrease it\n else:\n # i is perfect\n\n if i == 0:\n max_left = nums2[j - 1]\n elif j == 0:\n max_left = nums1[i - 1]\n else:\n max_left = max(nums1[i - 1], nums2[j - 1])\n\n if (m + n) % 2 == 1:\n return max_left\n\n if i == m:\n min_right = nums2[j]\n elif j == n:\n min_right = nums1[i]\n else:\n min_right = min(nums1[i], nums2[j])\n\n return (max_left + min_right) / 2.0\n\n# Example usage\nnums1 = [1, 3]\nnums2 = [2]\nmedian = find_median_sorted_arrays(nums1, nums2)\nprint(f"The median of {nums1} and {nums2} is: {median}")\n\n\n**Explanation:**\n\n1. **Handle Unequal Lengths:**\n - If ‘nums1‘ is longer than ‘nums2‘, swap them to ensure ‘nums1‘ is always the shorter array. This simplifies the logic.\n\n2. **Binary Search:**\n - The core idea is to use binary search on the shorter array (‘nums1‘).\n - ‘imin‘ and ‘imax‘ define the search range for the partition point ‘i‘ in ‘nums1‘.\n - ‘half_len‘ calculates the desired total number of elements on the left side of the partition (to find the median).\n\n3. **Partitioning:**\n - In each iteration, ‘i‘ divides ‘nums1‘, and ‘j‘ divides ‘nums2‘ such that the total number of elements on the left side is ‘half_len‘.\n\n4. **Adjusting the Search Range:**\n - If ‘nums2[j - 1] > nums1[i]‘, it means ‘i‘ is too small, so we need to move the partition point in ‘nums1‘ to the right (‘imin = i + 1‘).\n - If ‘nums1[i - 1] > nums2[j]‘, ‘i‘ is too large, so we move it to the left (‘imax = i - 1‘).\n\n5. **Finding the Median:**\n - When the correct partition is found (‘i‘ is in the right position), we find:\n - ‘max_left‘: The maximum element on the left side of the partition.\n - ‘min_right‘: The minimum element on the right side of the partition.\n - If the total number of elements is odd, ‘max_left‘ is the median.\n - If even, the median is the average of ‘max_left‘ and ‘min_right‘.\n\n**Time Complexity:** O(log(min(m, n))) because we are performing binary search on the shorter array.\n\n**Space Complexity:** O(1) as we are only using a constant amount of extra space for variables.
Judgment (by gpt-4)	
Judgment for gemma-2-9b-it-dpo	**Judgment for gemma-2-9b-it-dpo-eva**
The assistant’s answer is correct and provides a detailed explanation of the code. The assistant correctly states that the time complexity is O(log(min(m, n))) and the space complexity is O(1), which aligns with the reference answer. The assistant also provides an example usage of the function, which is helpful for understanding how to use it. The assistant’s code is a different implementation than the reference answer, but it achieves the same result and meets the same complexity requirements. However, the assistant’s answer does not address the user’s original question about achieving O(n) time complexity. The assistant should have clarified that it’s not possible to find the median of two sorted arrays of different sizes with O(1) space complexity and O(n) time complexity, as stated in the reference answer. Rating: [[8]], "score": 8, "turn": 1, "tstamp": 1722737514.2011044	The assistant’s answer is correct and provides a detailed explanation of the code. The assistant correctly states that the time complexity is O(log(min(m, n))) and the space complexity is O(1), which aligns with the reference answer. The assistant also provides a step-by-step explanation of the code, which is very helpful for understanding the logic behind the solution. The assistant’s code is also correct and will correctly find the median of two sorted arrays of different sizes. Therefore, there are no corrections needed for the assistant’s response. Rating: [[10]], "score": 10, "turn": 1, "tstamp": 1722846533.772981

Table 20: (Continued) comparison of model generations on examples from MT-Bench.

1st-turn prompt: You are given two sorted lists of size m and n. Implement a function to find the kth smallest element in the union of the two lists with linear complexity.

Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
‘‘‘python\n\def find_kth_smallest(list1, list2, k):\n """\n Finds the kth smallest element in the union of two sorted lists.\n\n Args:\n list1: The first sorted list.\n list2: The second sorted list.\n k: The desired kth smallest element.\n\n Returns:\n The kth smallest element in the union of the two lists.\n """\n\n m, n = len(list1), len(list2)\n i, j = 0, 0\n min_heap = [] \n\n # Build a min-heap with the first elements of both lists\n for x in range(min(m, n)):\n heapq.heappush(min_heap, (list1[x], x, 0)) # (value, list_index, position_in_list)\n heapq.heappush(min_heap, (list2[x], x, 1))\n\n # Extract k smallest elements from the heap\n for _ in range(k - 1):\n _, _, _ = heapq.heappop(min_heap)\n\n # The top element of the heap is the kth smallest\n return heapq.heappop(min_heap)[0]\n\n# Example usage:\nlist1 = [2, 5, 8, 12]\nlist2 = [1, 3, 7, 9, 10]\nk = 5\nkth_smallest = find_kth_smallest(list1, list2, k)\nprint(f"The {k}th smallest element is: {kth_smallest}")\n\n\n**Explanation:**\n\n1. **Initialization:**\n - ‘m‘ and ‘n‘ store the lengths of the input lists.\n - ‘i‘ and ‘j‘ are pointers to track the current position in each list.\n - ‘min_heap‘ is a min-heap used to efficiently store and retrieve the smallest elements encountered so far.\n\n2. **Building the Min-Heap:**\n - The code iterates through the first ‘min(m, n)‘ elements of both lists.\n - For each element, a tuple ‘(value, list_index, position_in_list)‘ is pushed onto the ‘min_heap‘. This tuple helps us keep track of the element’s value, the list it belongs to, and its original position.\n\n3. **Extracting k-1 Smallest Elements:**\n - The ‘for‘ loop runs ‘k - 1‘ times, effectively extracting the ‘k-1‘ smallest elements from the ‘min_heap‘.\n\n4. **Finding the kth Smallest:**\n - After the loop, the top element of the ‘min_heap‘ contains the ‘k‘th smallest element.\n - ‘heapq.heappop(min_heap)‘ removes and returns this element.\n\n**Time Complexity:**\n\n- Building the min-heap: O(m + n)\n- Extracting k-1 elements: O(k * log(k))\n- Overall: **O(m + n + k * log(k))**, which is approximately **O(m + n)** since ‘k‘ is usually much smaller than ‘m + n‘.\n\n\n\nLet me know if you have any other questions!	‘‘‘python\n\def find_kth_smallest(list1, list2, k):\n """\n Finds the kth smallest element in the union of two sorted lists.\n\n Args:\n list1: The first sorted list.\n list2: The second sorted list.\n k: The desired rank of the smallest element.\n\n Returns:\n The kth smallest element in the union of the lists.\n """\n\n i = 0 # Index for list1\n j = 0 # Index for list2\n merged = [] # Store the merged elements\n\n while i < len(list1) and j < len(list2):\n if list1[i] <= list2[j]:\n merged.append(list1[i])\n i += 1\n else:\n merged.append(list2[j])\n j += 1\n\n # Add any remaining elements from list1\n while i < len(list1):\n merged.append(list1[i])\n i += 1\n\n # Add any remaining elements from list2\n while j < len(list2):\n merged.append(list2[j])\n j += 1\n\n return merged[k - 1] # Return the kth element (index k-1)\n\n# Example usage:\nlist1 = [2, 5, 8, 12]\nlist2 = [1, 3, 7, 9, 10]\nk = 5\nkth_smallest = find_kth_smallest(list1, list2, k)\nprint(f"The {k}th smallest element is: {kth_smallest}")\n\n\n**Explanation:**\n\n1. **Initialization:**\n - ‘i‘ and ‘j‘ are index pointers for ‘list1‘ and ‘list2‘ respectively, starting at 0.\n - ‘merged‘ is an empty list to store the merged sorted elements.\n\n2. **Merging:**\n - The ‘while‘ loop continues as long as both ‘i‘ and ‘j‘ are within the bounds of their respective lists.\n - In each iteration, it compares the elements at ‘list1[i]‘ and ‘list2[j]‘.\n - The smaller element is appended to ‘merged‘, and the corresponding index (‘i‘ or ‘j‘) is incremented.\n\n3. **Adding Remaining Elements:**\n - After one list is exhausted, the remaining elements from the other list are directly appended to ‘merged‘.\n\n4. **Returning the kth Element:**\n - Finally, ‘merged[k - 1]‘ is returned, which represents the kth smallest element in the union (remember that list indices start from 0).\n\n**Linear Complexity:**\n\n- The algorithm iterates through each element in both lists at most once during the merging process.\n- The time complexity is therefore O(m + n), which is linear in the total number of elements (m + n), making it efficient for large lists.

Judgment (by gpt-4)

Judgment for gemma-2-9b-it-dpo	**Judgment for gemma-2-9b-it-dpo-eva**
The assistant’s answer is incorrect. The user asked for a solution with linear complexity, but the assistant provided a solution with a time complexity of O(m + n + k * log(k)), which is not linear. The assistant’s solution uses a min-heap, which is not necessary and adds unnecessary complexity to the solution. Rating: [[2]], "score": 2, "turn": 1, "tstamp": 1722737030.6411152	The assistant’s answer is correct and helpful. It provides a Python function that implements the two-pointer approach to find the kth smallest element in the union of two sorted lists. The assistant’s function is slightly different from the reference, but it achieves the same goal. The assistant’s function merges the two lists into a new list and then returns the kth element from the merged list. The assistant also provides a detailed explanation of how the function works and confirms that the time complexity is linear, which is what the user asked for. However, the assistant’s function uses extra space to store the merged list, which is not necessary and not as efficient as the reference. Rating: [[8]], "score": 8, "turn": 1, "tstamp": 1722846543.867601

Table 21: (Continued) comparison of model generations on examples from MT-Bench.

2nd-turn prompt: Does there exist an algorithm with better time complexity? If so, implement it.

Answer from gemma-2-9b-it-dpo	**Answer from gemma-2-9b-it-dpo-eva**
You are absolutely right! There’s a more efficient algorithm with a time complexity of **O(m + n)** without using a heap.\n\nHere’s the implementation:\n\npython\n\def find_kth_smallest_optimized(list1, list2, k):\n """\n Finds the kth smallest element in the union of two sorted lists\n with O(m + n) time complexity.\n\n Args:\n list1: The first sorted list.\n list2: The second sorted list.\n k: The desired kth smallest element.\n\n Returns:\n The kth smallest element in the union of the two lists.\n """\n m, n = len(list1), len(list2)\n i, j = 0, 0\n result = []\n\n # Merge the lists while keeping track of the kth smallest\n while len(result) < k:\n if i < m and j < n:\n if list1[i] <= list2[j]:\n result.append(list1[i])\n i += 1\n else:\n result.append(list2[j])\n j += 1\n elif i < m:\n result.append(list1[i])\n i += 1\n else:\n result.append(list2[j])\n j += 1\n\n return result[k - 1]\n\n# Example usage:\nlist1 = [2, 5, 8, 12]\nlist2 = [1, 3, 7, 9, 10]\nk = 5\nkth_smallest = find_kth_smallest_optimized(list1, list2, k)\nprint(f"The {k}th smallest element is: {kth_smallest}")\n\n\n**Explanation:**\n\n1. **Initialization:**\n - ‘m‘ and ‘n‘ store the lengths of the lists.\n - ‘i‘ and ‘j‘ are pointers for ‘list1‘ and ‘list2‘, respectively.\n - ‘result‘ is a list to store the merged and sorted elements.\n\n2. **Merging and Tracking:**\n - The ‘while‘ loop continues until ‘k‘ elements are in the ‘result‘ list.\n - In each iteration, it compares the current elements from ‘list1‘ and ‘list2‘.\n - The smaller element is appended to ‘result‘, and the corresponding pointer (‘i‘ or ‘j‘) is incremented.\n - If one list is exhausted, the remaining elements from the other list are appended to ‘result‘.\n\n3. **Returning the kth Smallest:**\n - Finally, ‘result[k - 1]‘ contains the ‘k‘th smallest element.\n\n**Time Complexity:**\n\n- The algorithm iterates through both lists once, resulting in a time complexity of **O(m + n)**.\n\n\n\nLet me know if you have any more questions!	You are right to ask! While the previous algorithm has linear time complexity, we can achieve an even better solution with **O(log(min(m, n)))** time complexity using a slightly more sophisticated approach.\n\nThe key idea is to leverage the sorted nature of the lists and use a **binary search** strategy.\n\nHere’s the implementation:\n\n\##\npython\n\def find_kth_smallest_optimized(list1, list2, k):\n """\n Finds the kth smallest element in the union of two sorted lists with O(log(min(m, n))) time complexity.\n\n Args:\n list1: The first sorted list.\n list2: The second sorted list.\n k: The desired rank of the smallest element.\n\n Returns:\n The kth smallest element in the union of the lists.\n """\n m, n = len(list1), len(list2)\n if m > n:\n list1, list2, m, n = list2, list1, n, m # Ensure list1 is the shorter one\n\n left, right = 0, m\n while left <= right:\n partitionX = (left + right) // 2\n partitionY = k - partitionX\n\n maxLeftX = list1[partitionX - 1] if partitionX > 0 else float(’-inf’)\n minRightX = list1[partitionX] if partitionX < m else float(’inf’)\n\n maxLeftY = list2[partitionY - 1] if partitionY > 0 else float(’-inf’)\n minRightY = list2[partitionY] if partitionY < n else float(’inf’)\n\n if maxLeftX <= minRightY and maxLeftY <= minRightX:\n if partitionX == 0:\n return list2[partitionY - 1]\n elif partitionY == 0:\n return list1[partitionX - 1]\n else:\n return max(list1[partitionX - 1], list2[partitionY - 1])\n elif maxLeftX > minRightY:\n right = partitionX - 1 # Too many elements on the right side of list1\n else:\n left = partitionX + 1 # Too many elements on the right side of list2\n\n\n**Explanation:**\n\n1. **Shorten Lists:** If ‘list1‘ is longer than ‘list2‘, swap them to ensure ‘list1‘ is always the shorter one. This simplifies the logic.\n\n2. **Binary Search:**\n - ‘left‘ and ‘right‘ pointers define the search space within ‘list1‘.\n - ‘partitionX‘ is the middle point in ‘list1‘, and ‘partitionY‘ is calculated to divide the remaining elements in ‘list2‘ such that ‘partitionX + partitionY = k‘.\n - ‘maxLeftX‘, ‘minRightX‘, ‘maxLeftY‘, and ‘minRightY‘ represent the largest element to the left and smallest element to the right of the partitions in both lists.\n\n3. **Comparison and Adjustment:**\n - If the ‘maxLeftX‘ is less than or equal to ‘minRightY‘ and ‘maxLeftY‘ is less than or equal to ‘minRightX‘, it means the kth smallest element lies within the current partitions. We return the larger of ‘maxLeftX‘ and ‘maxLeftY‘.\n - If ‘maxLeftX‘ is greater than ‘minRightY‘, it indicates that we need to search in the left subarray of ‘list1‘ (i.e., decrease ‘right‘).\n - Otherwise, we search in the right subarray of ‘list1‘ (i.e., increase ‘left‘).\n\n4. **Termination:** The binary search continues until ‘left‘ crosses ‘right‘, indicating the correct partition has been found.\n\n**Time Complexity:**\n\n- The binary search divides the search space in half at each step.\n- Therefore, the time complexity is O(log(min(m, n))), which is significantly faster than linear time for large lists where one list is much smaller than the other.

Judgment (by gpt-4)

Judgment for gemma-2-9b-it-dpo	**Judgment for gemma-2-9b-it-dpo-eva**
The assistant’s first response to the user’s question is incorrect. The user asked for a function to find the kth smallest element in the union of two sorted lists with linear complexity. The assistant provided a function that uses a min-heap, which has a time complexity of O(m + n + k * log(k)). This is not linear complexity, as the user requested. The assistant’s second response to the user’s question is also incorrect. The user asked if there exists an algorithm with better time complexity than linear. The assistant provided a function that has a time complexity of O(m + n), which is linear, not better than linear. The assistant should have provided a function that uses a binary search approach, which has a time complexity of O(log(m) + log(n)), as the reference answer correctly does. Rating: [[2]], "score": 2, "turn": 2, "tstamp": 1722737031.5033472	The assistant’s response is correct and helpful. The assistant provided a Python function that uses a binary search approach to find the kth smallest element in the union of two sorted lists. The assistant also provided a detailed explanation of how the function works, including the time complexity of the function. The assistant’s answer matches the reference answer in terms of the algorithm used and the explanation provided. The assistant also correctly identified that the binary search approach has a better time complexity than the linear approach. However, the assistant’s code has a minor issue. The assistant’s code does not handle the case when ‘k‘ is greater than the sum of the lengths of the two lists. In such a case, the code will raise an ‘IndexError‘. This can be fixed by adding a check at the beginning of the function to return ‘None‘ if ‘k‘ is greater than the sum of the lengths of the two lists. Rating: [[9]], "score": 9, "turn": 2, "tstamp": 1722846556.6828268

Table 22: (Continued) comparison of model generations on examples from MT-Bench.

REFERENCES

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C Machado. Loss of plasticity in continual deep reinforcement learning. In *Conference on Lifelong Learning Agents*, pages 620–636. PMLR, 2023.

Bo Adler, Niket Agarwal, Ashwath Aithal, Dong H Anh, Pallab Bhattacharya, Annika Brundyn, Jared Casper, Bryan Catanzaro, Sharon Clay, Jonathan Cohen, et al. Nemotron-4 340B Technical Report. *arXiv preprint arXiv:2406.11704*, 2024.

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting reinforce style optimization for learning from human feedback in llms. *arXiv preprint arXiv:2402.14740*, 2024.

Jordan T Ash and Ryan P Adams. On the difficulty of warm-starting neural network training. *arXiv preprint arXiv:1910.08475*, 2019.

Alfredo Banos. On pseudo-games. *The Annals of Mathematical Statistics*, 39(6):1932–1945, 1968.

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

Michael Beukman, Samuel Coward, Michael Matthews, Mattie Fellows, Minqi Jiang, Michael Dennis, and Jakob Foerster. Refining Minimax Regret for Unsupervised Environment Design. *arXiv preprint arXiv:2402.12284*, 2024a.

Michael Beukman, Samuel Coward, Michael Matthews, Mattie Fellows, Minqi Jiang, Michael Dennis, and Jakob Foerster. Refining Minimax Regret for Unsupervised Environment Design. *arXiv preprint arXiv:2402.12284*, 2024b.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc., 2020. URL `https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf`.

Seth Chaiklin et al. The zone of proximal development in Vygotsky's analysis of learning and instruction. *Vygotsky's educational theory in cultural context*, 1(2):39–64, 2003.

Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the Next AutoDiff: Generative Optimization with Rich Feedback, Execution Traces, and LLMs. *arXiv preprint arXiv:2406.16218*, 2024.

Eugene Choi, Arash Ahmadian, Matthieu Geist, Oilvier Pietquin, and Mohammad Gheshlaghi Azar. Self-Improving Robust Preference Optimization. *arXiv preprint arXiv:2406.01660*, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

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 preprint arXiv:2310.01377*, 2023.

Nirjhar Das, Souradip Chakraborty, Aldo Pacchiano, and Sayak Ray Chowdhury. Provably sample efficient rlhf via active preference optimization. *arXiv preprint arXiv:2402.10500*, 2024.

Peter Dayan and Geoffrey E Hinton. Using expectation-maximization for reinforcement learning. *Neural Computation*, 9(2):271–278, 1997.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

Mucong Ding, Souradip Chakraborty, Vibhu Agrawal, Zora Che, Alec Koppel, Mengdi Wang, Amrit Bedi, and Furong Huang. Sail: Self-improving efficient online alignment of large language models. *arXiv preprint arXiv:2406.15567*, 2024.

Shibhansh Dohare, J Fernando Hernandez-Garcia, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Maintaining plasticity in deep continual learning. *arXiv preprint arXiv:2306.13812*, 2023.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Yihan Zhang, Winnie Chow, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and Tong Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *arXiv preprint arXiv:2304.06767*, 2023.

Hanze Dong, Wei Xiong, Bo Pang, Haoxiang Wang, Han Zhao, Yingbo Zhou, Nan Jiang, Doyen Sahoo, Caiming Xiong, and Tong Zhang. RLHF Workflow: From Reward Modeling to Online RLHF, 2024.

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

Yann Dubois, Balázs Galambosi, Percy Liang, and Tatsunori B Hashimoto. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.

Ky Fan. Minimax theorems. *Proceedings of the National Academy of Sciences*, 39(1):42–47, 1953.

Lizhe Fang, Yifei Wang, Zhaoyang Liu, Chenheng Zhang, Stefanie Jegelka, Jinyang Gao, Bolin Ding, and Yisen Wang. What is Wrong with Perplexity for Long-context Language Modeling? *arXiv preprint arXiv:2410.23771*, 2024.

Yoav Freund and Robert E. Schapire. (Adaptive Game Playing Using Multiplicative Weights). *Games and Economic Behavior*, 29:79–103, 1999.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, et al. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data. *arXiv preprint arXiv:2404.01413*, 2024.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

Roger Grosse. Bilevel Optimization and Generalization, 2022.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Shangmin Guo, Biao Zhang, Tianlin Liu, Tianqi Liu, Misha Khalman, Felipe Llinares, Alexandre Rame, Thomas Mesnard, Yao Zhao, Bilal Piot, et al. Direct language model alignment from online ai feedback. *arXiv preprint arXiv:2402.04792*, 2024.

Joey Hejna, Rafael Rafailov, Harshit Sikchi, Chelsea Finn, Scott Niekum, W Bradley Knox, and Dorsa Sadigh. Contrastive prefence learning: Learning from human feedback without rl. *arXiv preprint arXiv:2310.13639*, 2023.

Jiwoo Hong, Noah Lee, and James Thorne. Orpo: Monolithic preference optimization without reference model. *arXiv preprint arXiv:2403.07691*, 2(4):5, 2024.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

Edward Hughes, Michael Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktaschel. Open-Endedness is Essential for Artificial Superhuman Intelligence. *arXiv preprint arXiv:2406.04268*, 2024a.

Edward Hughes, Michael Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktaschel. Open-Endedness is Essential for Artificial Superhuman Intelligence. *arXiv preprint arXiv:2406.04268*, 2024b.

Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminksy, Michael Kozuch, Zachary C Lipton, et al. Accelerating deep learning by focusing on the biggest losers. *arXiv preprint arXiv:1910.00762*, 2019.

Dongfu Jiang, Xiang Ren, and Bill Yuchen Lin. Llm-blender: Ensembling large language models with pairwise ranking and generative fusion. *arXiv preprint arXiv:2306.02561*, 2023.

Minqi Jiang. Learning Curricula in Open-Ended Worlds. *arXiv preprint arXiv:2312.03126*, 2023.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021a.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pages 4940–4950. PMLR, 2021b.

Kenji Kawaguchi and Haihao Lu. Ordered sgd: A new stochastic optimization framework for empirical risk minimization. In *International Conference on Artificial Intelligence and Statistics*, pages 669–679. PMLR, 2020.

John Maynard Keynes. *A treatise on probability*. Courier Corporation, 1921.

Rawal Khirodkar and Kris M Kitani. Adversarial domain randomization. *arXiv preprint arXiv:1812.00491*, 2018.

Haoran Li, Qingxiu Dong, Zhengyang Tang, Chaojun Wang, Xingxing Zhang, Haoyang Huang, Shaohan Huang, Xiaolong Huang, Zeqiang Huang, Dongdong Zhang, et al. Synthetic data (almost) from scratch: Generalized instruction tuning for language models. *arXiv preprint arXiv:2402.13064*, 2024a.

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. *arXiv preprint arXiv:2406.11939*, 2024b.

Chris Yuhao Liu and Liang Zeng. Skywork Reward Model Series. `https://huggingface.co/Skywork`, September 2024. URL `https://huggingface.co/Skywork`.

Tianqi Liu, Yao Zhao, Rishabh Joshi, Misha Khalman, Mohammad Saleh, Peter J Liu, and Jialu Liu. Statistical rejection sampling improves preference optimization. *arXiv preprint arXiv:2309.06657*, 2023a.

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

Lin Long, Rui Wang, Ruixuan Xiao, Junbo Zhao, Xiao Ding, Gang Chen, and Haobo Wang. On llms-driven synthetic data generation, curation, and evaluation: A survey. *arXiv preprint arXiv:2406.15126*, 2024.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*, 2023.

Jacob Makar-Limanov, Arjun Prakash, Denizalp Goktas, Amy Greenwald, and Nora Ayanian. STA-RLHF: Stackelberg Aligned Reinforcement Learning with Human Feedback. In *Coordination and Cooperation for Multi-Agent Reinforcement Learning Methods Workshop*, 2024. URL `https://openreview.net/forum?id=Jcn8pxwRa9`.

Yu Meng, Mengzhou Xia, and Danqi Chen. SimPO: Simple Preference Optimization with a Reference-Free Reward. *arXiv preprint arXiv:2405.14734*, 2024.

Sören Mindermann, Jan M Brauner, Muhammed T Razzak, Mrinank Sharma, Andreas Kirsch, Winnie Xu, Benedikt Höltgen, Aidan N Gomez, Adrien Morisot, Sebastian Farquhar, et al. Prioritized training on points that are learnable, worth learning, and not yet learnt. In *International Conference on Machine Learning*, pages 15630–15649. PMLR, 2022.

William Muldrew, Peter Hayes, Mingtian Zhang, and David Barber. Active Preference Learning for Large Language Models. *arXiv preprint arXiv:2402.08114*, 2024.

Rémi Munos, Michal Valko, Daniele Calandriello, Mohammad Gheshlaghi Azar, Mark Rowland, Zhaohan Daniel Guo, Yunhao Tang, Matthieu Geist, Thomas Mesnard, Andrea Michi, et al. Nash learning from human feedback. *arXiv preprint arXiv:2312.00886*, 2023.

John F Nash et al. Non-cooperative games. *Princeton University*, 1950.

Allen Nie, Ching-An Cheng, Andrey Kolobov, and Adith Swaminathan. The Importance of Directional Feedback for LLM-based Optimizers. *arXiv preprint arXiv:2405.16434*, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pages 16828–16847. PMLR, 2022.

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

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*, 2024.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pages 17473–17498. PMLR, 2022.

Gabriel Poesia, David Broman, Nick Haber, and Noah D Goodman. Learning Formal Mathematics From Intrinsic Motivation. *arXiv preprint arXiv:2407.00695*, 2024.

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 preprint arXiv:2305.18290*, 2023.

Aravind Rajeswaran, Igor Mordatch, and Vikash Kumar. A game theoretic framework for model based reinforcement learning. In *International conference on machine learning*, pages 7953–7963. PMLR, 2020.

Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacroce, Ahmed Awadallah, and Tengyang Xie. Direct Nash Optimization: Teaching Language Models to Self-Improve with General Preferences. *arXiv preprint arXiv:2404.03715*, 2024.

Daniel J Russo, Benjamin Van Roy, Abbas Kazerouni, Ian Osband, Zheng Wen, et al. A tutorial on thompson sampling. *Foundations and Trends® in Machine Learning*, 11(1):1–96, 2018.

Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

Mikayel Samvelyan, Akbir Khan, Michael Dennis, Minqi Jiang, Jack Parker-Holder, Jakob Foerster, Roberta Raileanu, and Tim Rocktäschel. MAESTRO: Open-ended environment design for multi-agent reinforcement learning. *arXiv preprint arXiv:2303.03376*, 2023.

Leonard J Savage. The theory of statistical decision. *Journal of the American Statistical association*, 46(253):55–67, 1951.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. *In Proc. of the international conference on simulation of adaptive behavior: From animals to animats*, 1991.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Hans-Paul Schwefel. *Evolutionsstrategien für die numerische Optimierung*. Springer, 1977.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. RL on Incorrect Synthetic Data Scales the Efficiency of LLM Math Reasoning by Eight-Fold. *arXiv preprint arXiv:2406.14532*, 2024.

Han Shen, Zhuoran Yang, and Tianyi Chen. Principled penalty-based methods for bilevel reinforcement learning and rlhf. *arXiv preprint arXiv:2402.06886*, 2024.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of Go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, 2021.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*, 2024.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. Principle-driven self-alignment of language models from scratch with minimal human supervision. *Advances in Neural Information Processing Systems*, 36, 2024.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 761–768, 2011.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. Preference fine-tuning of llms should leverage suboptimal, on-policy data. *arXiv preprint arXiv:2404.14367*, 2024.

Yunhao Tang, Zhaohan Daniel Guo, Zeyu Zheng, Daniele Calandriello, Rémi Munos, Mark Rowland, Pierre Harvey Richemond, Michal Valko, Bernardo Ávila Pires, and Bilal Piot. Generalized preference optimization: A unified approach to offline alignment. *arXiv preprint arXiv:2402.05749*, 2024.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Gemini Team, M Reid, N Savinov, D Teplyashin, Lepikhin Dmitry, T Lillicrap, JB Alayrac, R Soricut, A Lazaridou, O Firat, et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024a.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024b.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

Hoang Tran, Chris Glaze, and Braden Hancock. Iterative DPO Alignment. Technical report, Snorkel AI, 2023.

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. Will we run out of data? Limits of LLM scaling based on human-generated data. *arXiv preprint arXiv:2211.04325*, 2024.

Heinrich von Stackelberg. *Marktform und Gleichgewicht*. Die Handelsblatt-Bibliothek "Klassiker der Nationalökonomie". J. Springer, 1934. URL `https://books.google.com/books?id=wihBAAAAIAAJ`.

Haoxiang Wang, Wei Xiong, Tengyang Xie, Han Zhao, and Tong Zhang. Interpretable Preferences via Multi-Objective Reward Modeling and Mixture-of-Experts. *arXiv preprint arXiv:2406.12845*, 2024.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

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. *arXiv preprint arXiv:2212.10560*, 2022.

Huasen Wu and Xin Liu. Double thompson sampling for dueling bandits. *Advances in neural information processing systems*, 29, 2016.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*, 2024.

Wei Xiong, Hanze Dong, Chenlu Ye, Ziqi Wang, Han Zhong, Heng Ji, Nan Jiang, and Tong Zhang. Iterative preference learning from human feedback: Bridging theory and practice for rlhf under kl-constraint. In *Forty-first International Conference on Machine Learning*, 2024.

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 preprint arXiv:2304.12244*, 2023a.

Jing Xu, Andrew Lee, Sainbayar Sukhbaatar, and Jason Weston. Some things are more cringe than others: Preference optimization with the pairwise cringe loss. *arXiv preprint arXiv:2312.16682*, 2023b.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. Magpie: Alignment Data Synthesis from Scratch by Prompting Aligned LLMs with Nothing. *arXiv preprint arXiv:2406.08464*, 2024.

Fuzhao Xue, Yao Fu, Wangchunshu Zhou, Zangwei Zheng, and Yang You. To repeat or not to repeat: Insights from scaling llm under token-crisis. *Advances in Neural Information Processing Systems*, 36, 2024.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36, 2024.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*, 2023.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. TextGrad: Automatic" Differentiation" via Text. *arXiv preprint arXiv:2406.07496*, 2024.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Yao Zhao, Rishabh Joshi, Tianqi Liu, Misha Khalman, Mohammad Saleh, and Peter J Liu. Slic-hf: Sequence likelihood calibration with human feedback. *arXiv preprint arXiv:2305.10425*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Rui Zheng, Hongyi Guo, Zhihan Liu, Xiaoying Zhang, Yuanshun Yao, Xiaojun Xu, Zhaoran Wang, Zhiheng Xi, Tao Gui, Qi Zhang, et al. Toward Optimal LLM Alignments Using Two-Player Games. *arXiv preprint arXiv:2406.10977*, 2024.