

# Evolving Alignment *via* Asymmetric Self-Play

*Scalable Preference Fine-Tuning Beyond Static Human Prompts*

**Ziyu Ye \***  
University of Chicago

**Rishabh Agarwal**  
Google DeepMind

**Tianqi Liu**  
Google DeepMind

**Rishabh Joshi**  
Google DeepMind

**Sarmishta Velury**  
Google DeepMind

**Quoc V. Le**  
Google DeepMind

**Qijun Tan**  
Google DeepMind

**Yuan Liu**  
Google DeepMind

## Abstract

Current RLHF approaches for aligning large language models (LLMs) typically assume a fixed prompt distribution, which is sub-optimal and limits the generalization capabilities for language models. To address this issue, we introduce a general framework that casts alignment as an asymmetric game between two players: (i) **a creator** that generates increasingly informative prompt distributions using the reward model, and (ii) **a solver** that learns to produce more preferred responses on prompts produced by the creator. This framework of **Evolving Alignment *via* Asymmetric Self-Play (eva)**, results in a simple and efficient approach that can utilize any existing RLHF algorithm. **eva** outperforms state-of-the-art methods on widely-used benchmarks, without the need of any additional human crafted prompts. Specifically, **eva** improves the win rate of GEMMA2-9B-IT on Arena-Hard from 51.6% to 60.1% with DPO, from 55.7% to 58.9% with SPPO, from 52.3% to 60.7% with SimPO, and from 54.8% to 60.3% with ORPO, surpassing its 27B version and matching `claude-3-opus`. This improvement is persistent even when new human crafted prompts are introduced. Finally, we show **eva** is effective and robust under various ablation settings.

*What I cannot create, I do not understand.*

*Richard P. Feynman*

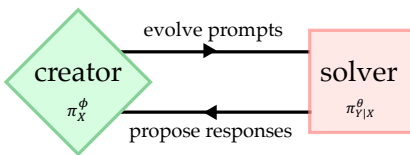


Figure 1: **eva**: Open-Ended RLHF *via* Asymmetric Self-Play. The creator is the prompt generation policy  $\pi_X$  and the solver is the response policy  $\pi_{Y|X}$ .

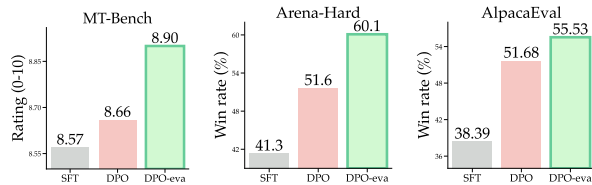


Figure 2: **Results**: Our method **eva** achieves concrete performance gain especially on *hard* alignment benchmarks, without relying on any additional human prompts. Here, we report results for DPO-**eva**; see more in §3.

## 1 Introduction

Long-lived artificial intelligence must deal with an ever-evolving, open-ended world, yet currently face constraints in both the *scale* and *quality* of available data, and the *growth rate* at which new, useful information is created. High quality human data, crucial for scaling large language model (LLM) based intelligence, is projected to run out in the next few years [Villalobos et al., 2024]; the quality of such data is also expected to stagnate: as LLMs become more capable, they need to solve increasingly complex or new challenges, requiring training data beyond abilities of humans to create. This necessitates a new fundamental mechanism for self-improving, where models can continuously self-generate and self-solve harder problems. We thereby investigate the research question below:

*Can language models self-create new, learnable tasks to work on,  
to self-improve to generalize better for human preferences alignment?*

\*Work done during an internship at Google.

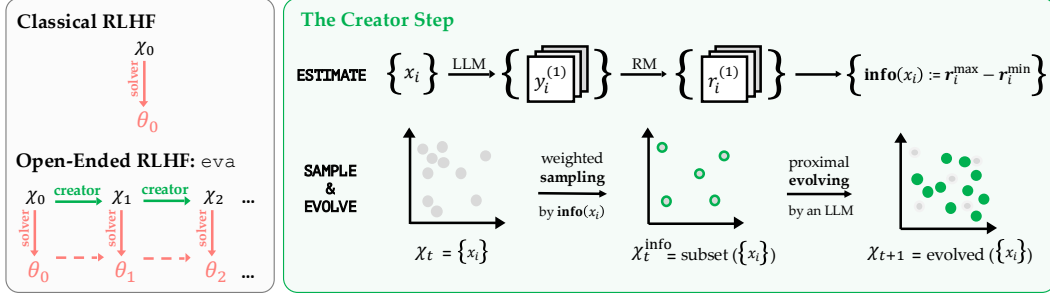


Figure 3: **Pipeline: eva** generalizes classical RLHF with open-ended RLHF, optimized *via* a creator-solver game for self-improving language models. See more on the objective and principle in § 2.

Many preference optimization algorithms [Christiano et al., 2017, Zhao et al., 2023] have been proposed to improve the alignment of language models, however, they all default to fixed prompt training distributions. Such fixed training paradigm inevitably leads to: (i) *generalization issues* (models may underperform or hack on insufficiently represented instructions) and (ii) *efficiency issues* (data annotation and model training are costly, however not all prompt provide the same utility, and identifying informative prompts by human efforts is nontrivial) [Team et al., 2023, 2024].

The objective thereby should not only be to optimize over a specific, static distribution of prompts, rather, to design an agent that can autonomously evolve the training distribution for self-improvement, to align well across unseen, novel environments or tasks (instantiated by prompts).

We develop **eva** (Evolving Alignment *via* Asymmetric Self-Play), as in Figure 1, 3. Central to **eva** is a game with the minimax-regret objective, achieved by alternating optimization of creating prompts and solving them. The interplay encourages evolving curricula [Parker-Holder et al., 2022], potentially benefits both generalization and efficiency (see § C). Orthogonal to many recent self-play studies in LLM alignment [Munos et al., 2023, Wu et al., 2024], **eva** is *asymmetric* [Sukhbaatar et al., 2017], with two policies of different goals; and in contrast to many self-training works [Gulcehre et al., 2023] focusing on improving in  $\mathcal{Y} \mid \mathcal{X}$ , we jointly optimize in  $(\mathcal{X}, \mathcal{Y})$  by generative exploration.

Our main contributions are: (i) **a new principle** of open-ended RLHF that generalize classical RLHF; (ii) **a new algorithm** with asymmetric self-play that can be plugged into any alignment pipeline; and (iii) **the new state-of-the-art** especially on *hard* alignment benchmarks across various algorithms.

**eva** is easy to deploy. We hope it can serve as a scalable method for the AI community to build robust, open-ended, sample-efficient and self-improving intelligence, that aligns with human values.

## 2 Method

**Algorithm overview.** On a high level, **eva** extends RLHF to open-ended RLHF *via* a creator that adapts prompt distributions with an easy-to-implement *estimate, sample then evolve* procedure, mimicking the *minimax-regret* policy of asymmetric self-play games, as detailed in § B.

---

### Algorithm 1 **eva**: Evolving Alignment *via* Asymmetric Self-Play

---

**Input:** initial policy  $\pi_{\theta_0}$ , initial prompt set  $\mathcal{X}_0$

- 1: **for** iteration  $t = 1, 2, \dots$  **do**
  - ▽ /\* **creator step** \*/
  - 2: *estimate informativeness:*  $\mathcal{X}_t \leftarrow \mathcal{X}_t \cup \{\mathbf{info}(\mathbf{x}_i)\}$
  - sample subset:*  $\mathcal{X}_t^{\text{info}} \leftarrow \mathbf{sample}(\mathcal{X}_t)$
  - self-evolve prompts:*  $\mathcal{X}'_t \leftarrow \mathbf{evolve}(\mathcal{X}_t^{\text{info}})$
  - ▽ /\* **solver step** \*/
  - 3: *self-generate responses:*  $\forall \mathbf{x}_i \in \mathcal{X}'_t, \text{generate } \{\mathbf{y}_i^{(j)}\} \sim \pi_{\theta_{t-1}}(\cdot \mid \mathbf{x}_i)$
  - 4: *annotate rewards:*  $\mathcal{X}'_t \leftarrow \mathcal{X}'_t \cup \{(\mathbf{y}_i^{(j)}, r_i^{(j)})\}$
  - 5: *preference optimization:*  $\theta_t \leftarrow \theta_{t-1} - \eta \nabla_{\theta} \mathcal{L}_{\mathcal{X}'_t}(\theta)$
  - 6: **end for**

---

## 2.1 The Principle: Open-ended RLHF for Joint Self-Improvement

**Motivation.** Classical RLHF optimizes over a *static* prompt set, thus brittle in ever-changing worlds. To have an agent that *generalizes* well across *unseen, novel* tasks, we must design a new objective.

**Formalization.** We thus formally introduce *optimizable* prompt generation policy  $\pi_\phi(\mathbf{x})$ , which is *jointly* optimized with the response policy  $\pi_\theta(\mathbf{y} \mid \mathbf{x})$  with regard to the *joint reference policy*<sup>2</sup>:

**Definition 1 (Open-ended RLHF)** Let  $\pi_{\phi, \theta}(\mathbf{x}, \mathbf{y}) := \pi_\phi(\mathbf{x}) \cdot \pi_\theta(\mathbf{y} \mid \mathbf{x})$  and  $\pi_{\text{ref}}(\mathbf{x}, \mathbf{y}) := p_{\text{ref}}(\mathbf{x}) \cdot \pi_{\text{ref}}(\mathbf{y} \mid \mathbf{x})$ . We define evolving alignment<sup>a</sup> as the open-ended joint optimization on the prompt and response policy for alignment w.r.t the joint reference policy:

$$\max_{\phi, \theta} \mathbb{E}_{\mathbf{x} \sim \pi_\phi(\mathbf{x}), \mathbf{y} \sim \pi_\theta(\mathbf{y} \mid \mathbf{x})} \left[ r(\mathbf{x}, \mathbf{y}) \right] - \beta \cdot \mathbb{D}_{\text{KL}} \left[ \pi_{\phi, \theta}(\mathbf{x}, \mathbf{y}) \parallel \pi_{\text{ref}}(\mathbf{x}, \mathbf{y}) \right], \quad (1)$$

<sup>a</sup>This generalizes RLHF (Eq. 4), which is a special case if  $\pi_\phi$  is static as  $p_{\text{ref}}$ , by expanding Eq. 1:

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

## 2.2 The Mechanism: Asymmetric Self-Play via the Creator v.s. Solver Game

**Intuition.** It is hard to directly optimize Eq. 1, due to (i) the **intractability** of the unspecified reference [Dennis et al., 2020]; (ii) the **instability** of joint differentiation [Goodfellow et al., 2014]. We present an *alternating optimization* objective by casting it as an asymmetric creator-solver game.

- Intuitively, the creator can guide the solver by prompt curricula with *increasing complexity*, encouraging efficient and general learning to handle the diversity in the wild.
- Mathematically, this resembles RL optimization *via* expectation-maximization [Dayan and Hinton, 1997, Singh et al., 2023], where  $\phi$  for the prompt distribution is fixed at each step.

**Formalization.** We formalize the alternating optimization as an asymmetric game as follows:

- **Creator:** the prompt player  $\pi_{\mathcal{X}}$  that strategically generate prompts for the solver.
- **Solver:** the response player  $\pi_{\mathcal{Y} \mid \mathcal{X}}$  (or  $\pi$ ) that learn to generate preferred responses.

We use *minimax regret* strategy [Savage, 1951, Dennis et al., 2020], where the solver minimizes and the creator maximizes regret (see § A). At the equilibrium [Nash et al., 1950], there is:

**Remark 1 (Minimax Regret)** If the solver-creator game reaches an equilibrium, the solver follows a minimax regret strategy, i.e., it optimizes to perform well under all cases:

$$\pi^* \in \arg \min_{\pi \in \Pi_{\mathcal{Y} \mid \mathcal{X}}} \max_{\pi_{\mathcal{X}} \in \Pi_{\mathcal{X}}} \mathbb{E}_{\mathbf{x} \sim \pi_{\mathcal{X}}} \left[ \text{Regret}(\mathbf{x}, \pi) \right]. \quad (2)$$

However, without access to the true optimal policy, we must approximate the regret. Leveraging the *stochastic policy* and the *reward model*, we design the advantage-based proxy:

**Definition 2 (Informativeness Proxy)** We measure the informativeness of a prompt by the (absolute) empirical worst-case optimal advantage, approximating the minimax regret:

$$\text{info}(\mathbf{x}) \leftarrow \hat{A}_{\min}^* := \left| \min_{\mathbf{y}} r(\mathbf{x}, \mathbf{y}) - \max_{\mathbf{y}'} r(\mathbf{x}, \mathbf{y}') \right|, \quad (3)$$

which is estimated by sampling multiple responses for  $\mathbf{x}$  from the solver and calculating gap between the maximal reward (from the best  $\mathbf{y}$ ) and the minimal reward (from the worst  $\mathbf{y}$ ).

In sum, our open-ended RLHF allow for an evolving joint distribution for better generalization; the minimax regret objective adds *robustness* on such evolving curricula by *incentivizing agents to align well in all cases*. We use the informativeness proxy (see § C for more; an attentive reader may want to distinguish it from Shannon’s [Shannon, 1948]) to guide learning. Together, it is a mechanism that the creator keeps challenging the solver, and the solver learns to improve.

<sup>2</sup>Here,  $p_{\text{ref}}(\mathbf{x})$  represents an idealized, potentially intractable probability measure over all possible tasks (instantiated *via* prompts) in the wild, as a conceptual reference or a guiding target for alignment.

### 3 Main Experimental Results

**eva** brings notable gains in alignment without relying on any human-crafted data, thus offering data efficiency as well. In the base setup, from the one-iteration finetuned model ( $\theta_{0 \rightarrow 1}$ ), **eva** adds a creator to self-evolve prompts of the initial iteration and optimizes for an additional open-ended RLHF iteration, resulting in  $\theta_{1 \rightarrow \bar{1}}$ . See detailed settings in § D.

**eva achieves self-improvement.** As shown in red rows in Table 1, **eva** yields notable performance improvement over  $\theta_{0 \rightarrow 1}$  across different optimization algorithms, especially on the harder Arena-Hard benchmark, which is recognized to be more challenging and distinguishable among others due to the complexity of its prompts and its fairer scoring system [Li et al., 2024b]. For example, **eva** brings 8.5% gain with DPO as the solver, surpassing its 27B version and matching `claude-3-opus` as in the AH leaderboard, while using fully self-automated joint generation.

**eva can surpass human-crafted prompts.** We further show that **eva**-prompt-trained models ( $\theta_{1 \rightarrow \bar{1}}$ ) can match and even outperform those trained on additional new prompts from UltraFeedback ( $\theta_{1 \rightarrow 2}$ ) (which we denoted as human prompts), while being much cheaper and more efficient. Additionally, on MT-Bench, training with new human prompts typically show decreased performance in the first turn and only modest gains in the second turn. In contrast, **eva** notably enhances second-turn performance. We hypothesize that **eva** evolves novel, learnable prompts that include characteristics of second-turn questions, reflecting emergent skills like handling follow-up interactions.

Model Family ( $\rightarrow$ )	GEMMA-2-9B-IT					
Benchmark ( $\rightarrow$ )	Arena-Hard	MT-Bench			AlpacaEval 2.0	
Method ( $\downarrow$ ) / Metric ( $\rightarrow$ )	WR (%)	avg. score	1 <sup>st</sup> turn	2 <sup>nd</sup> turn	LC-WR (%)	WR (%)
$\theta_0$ : SFT	41.3	8.57	8.81	8.32	47.11	38.39
$\theta_{0 \rightarrow 1}$ : DPO	51.6	8.66	9.01	8.32	55.01	51.68
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b>	<b>60.1</b> (+8.5)	<b>8.90</b>	<b>9.04</b>	<b>8.75</b> (+0.43)	55.35	55.53
$\theta_{1 \rightarrow 2}$ : + new human prompts	59.8	8.64	8.88	8.39	55.74	56.15
$\theta_{0 \rightarrow 1}$ : SPPO	55.7	8.62	9.03	8.21	51.58	42.17
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b>	<b>58.9</b> (+3.2)	<b>8.78</b>	<b>9.11</b>	<b>8.45</b> (+0.24)	<b>51.86</b>	<b>43.04</b>
$\theta_{1 \rightarrow 2}$ : + new human prompts	57.7	8.64	8.90	8.39	51.78	42.98
$\theta_{0 \rightarrow 1}$ : SimPO	52.3	8.69	9.03	8.35	54.29	52.05
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b>	<b>60.7</b> (+8.4)	<b>8.92</b>	<b>9.08</b>	<b>8.77</b> (+0.42)	<b>55.85</b>	<b>55.92</b>
$\theta_{1 \rightarrow 2}$ : + new human prompts	54.6	8.76	9.00	8.52	54.40	55.72
$\theta_{0 \rightarrow 1}$ : ORPO	54.8	8.67	9.04	8.30	52.17	49.50
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b>	<b>60.3</b> (+5.5)	<b>8.89</b>	<b>9.07</b>	<b>8.71</b> (+0.41)	<b>54.39</b>	<b>50.88</b>
$\theta_{1 \rightarrow 2}$ : + new human prompts	57.2	8.74	9.01	8.47	54.00	51.21

Table 1: **Main results.** Our **eva** achieves notable alignment gains and can surpass human prompts on major benchmarks across a variety of representative direct preference optimization algorithms.

We then conduct in-depth ablation studies on **eva**, with findings below in the Appendix:

- § E.1 - **informativeness metric**: our *regret*-based metric outperforms other alternatives.
- § E.2 - **sample-then-evolve procedure**: our method outperforms greedy selection.
- § E.3 - **scaling w/ reward models**: the alignment gain of **eva** scales with reward models.
- § E.4 - **continual training**: our method has monotonic gain with incremental training; the *evolved data and schedule* by **eva** serves as an *implicit regularizer* for better local minima.

### 4 Concluding Remarks

**eva** is a new, simple framework for aligning language models, and can be plugged into any existing alignment pipeline. The primary takeaway may be that RLHF can be made open-ended: (i) self-evolving joint data distributions can bring significant gain (as shown across various preference optimization algorithms), and (ii) reward advantage acts as an effective metric informing the collection and creation of *future* prompts for alignment. **eva** presents a new view of alignment by framing it as an asymmetric game between a creator generating *new* and *learnable* prompts and a solver producing preferred responses. **eva** also *incentivizes agents to create problems* rather than to simply *solve problems*, which is a key feature of intelligence, yet model trainers often neglect.

# Appendix

## A Preliminaries

We hereby review major concepts, which we later in § 2 use *regret* and the proxy by *advantage* to identify informative prompts, leading to learning curricular implicitly maximizing *contrastive ratio*.

**Alignment by RLHF.** Classical RLHF [Ouyang et al., 2022] optimizes on a fixed distribution  $\mathcal{D}$ :

$$\max_{\pi_{\theta}} \mathbb{E}_{\mathbf{x} \sim \mathcal{D}, \mathbf{y} \sim \pi_{\theta}(\mathbf{y}|\mathbf{x})} \left[ r(\mathbf{x}, \mathbf{y}) \right] - \beta \cdot \mathbb{D}_{\text{KL}} \left[ \pi_{\theta}(\mathbf{y} | \mathbf{x}) \parallel \pi_{\text{ref}}(\mathbf{y} | \mathbf{x}) \right], \quad (4)$$

where  $\mathbf{x}$  and  $\mathbf{y}$  denote the prompts and responses, and  $r(\cdot, \cdot)$  is the reward function.

**Reward.** Let the *optimal policy* of Eq. 4 be  $\pi^*(\cdot)$  and  $Z(\cdot)$  be the partition function, we have:

$$r(\mathbf{x}, \mathbf{y}) = \beta \cdot \log \frac{\pi^*(\mathbf{y} | \mathbf{x})}{\pi_{\text{ref}}(\mathbf{y} | \mathbf{x})} + \beta \cdot \log Z(\mathbf{x}). \quad (5)$$

**Regret.** Let  $r^*(\mathbf{x}) = \max_{\mathbf{y}'} r(\mathbf{x}, \mathbf{y}')$  be the optimal reward achievable at  $\mathbf{x}$ , the *regret* to take  $\mathbf{y}$  is:

$$\text{Regret}(\mathbf{x}, \mathbf{y}) = r^*(\mathbf{x}) - r(\mathbf{x}, \mathbf{y}). \quad (6)$$

The regret with regard to policy is defined as:

$$\text{Regret}(\mathbf{x}, \pi) = \mathbb{E}_{\mathbf{y}' \sim \pi(\mathbf{y}'|\mathbf{x})} \left[ r(\mathbf{x}, \mathbf{y}') \right] - \mathbb{E}_{\mathbf{y}' \sim \pi^*(\mathbf{y}'|\mathbf{x})} \left[ r(\mathbf{x}, \mathbf{y}') \right]. \quad (7)$$

**Advantage.** The *advantage* function quantifies how much better a response  $\mathbf{y}$  is w.r.t. a baseline:

$$A(\mathbf{x}, \mathbf{y}) = r(\mathbf{x}, \mathbf{y}) - \mathbb{E}_{\mathbf{y}' \sim \pi(\mathbf{y}'|\mathbf{x})} \left[ r(\mathbf{x}, \mathbf{y}') \right]. \quad (8)$$

Variants of advantage (e.g., the worst-case advantage  $A_{\min}^*$ ) are related to regret, as shown in Table 2.

**Direct preference optimization.** The DPO [Rafailov et al., 2023] objective for RLHF is:

$$\mathcal{L}_{\beta}^{\text{DPO}}(\pi_{\theta}) = \sum_{(\mathbf{y}_+, \mathbf{y}_-, \mathbf{x}) \in \mathcal{D}} -\log \left[ \sigma \left( \beta \cdot \Delta_{\theta; \text{ref}}^{\mathbf{x}} \right) \right], \quad (9)$$

where we use  $+$ ,  $-$  to denote chosen and rejected responses, and denote the **contrastive ratio** as:

$$\Delta_{\theta; \text{ref}}^{\mathbf{x}} := \log \frac{\pi_{\theta}(\mathbf{y}_+ | \mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_+ | \mathbf{x})} - \log \frac{\pi_{\theta}(\mathbf{y}_- | \mathbf{x})}{\pi_{\text{ref}}(\mathbf{y}_- | \mathbf{x})}. \quad (10)$$

By reward reparameterization with Eq. 5, advantage also relates to contrastive ratio, as in § C.

## B The Practical Algorithm

We now illustrate **eva** of Algorithm 1 (cf, Fig 3), with specifications in Appendix I.

### B.1 The Creator Step: Estimate, Sample then Evolve

Plainly, the creator finds most useful prompts and generate variants of them for preference optimization. One may relate this to **evolution strategies** [Schwefel, 1977] which find the most promising species, then mutate and crossover, or to **curriculum RL** [Parker-Holder et al., 2022] which finds environments with high-regret levels, then edits within some distance. As in Section 2.2, we do not seek a differentiable creator in this work. The creator is implemented in three steps as in Figure 3.

**Step 1: `info`(·) – estimate the informativeness.** For each  $\mathbf{x}$  in the prompt set  $\mathcal{X}_t$ , we generate responses, annotate rewards and estimate a informativeness metric to  $\mathbf{x}$  by Eq. 3 (see also Table 2).

**Step 2: `sample`(·) – weighted sampling for an informative subset.** Using the informativeness metric as the weight, we sample a informative prompt subset  $\mathcal{X}_t^{\text{info}}$  to be evolved later.

**Step 3: `evolve(·)` – evolving for a proximal region of high-advantage prompts.** Our algorithm is agnostic to and does not rely on any specific evolving method. We take EvolInstruct [Xu et al., 2023a] as an off-the-shelf method, which conducts in-depth (*i.e.*, adding constraints, deepening, concretising, complicating) and in-breadth evolving (*i.e.*, mutation) for prompts. Specifically, we iterate over each prompt in the  $\mathcal{X}_t^{\text{info}}$ , where each one is evolved to multiple variations, then optionally mix the newly generated prompts with a uniformly sampled buffer from  $\mathcal{X}_t$  to create  $\mathcal{X}'_t$ .

## B.2 The Solver Step: Solve then Optimize

This step is the classical preference optimization [Rafailov et al., 2023], where responses are generated and the gradient descent is performed. Take the pointwise reward model setting as an example, for every prompt, we sample  $n$  responses with reward annotated for each; we take the responses with the maximal and the minimal reward to construct the preference pairs, then optimize upon.

Put together, **eva** can unify existing iterative optimization pipeline [Tran et al., 2023] with a new creator module, which can either share the same network as the solver policy or operate independently.

## C Understanding eva in Different Intuitive Ways

**Learning potential.** Our metric intuitively identifies the learning potential of a prompt by measuring the gap between the best and worst response to it from the solver. We reason, that prompts eliciting *both* high-reward and low-reward outcomes, reflect *learnable* tasks where the model is capable of improving but has not yet mastered, thereby implying learning potential (*cf.*, Jiang et al. [2021b]).

**Worst-case guarantees.** The minimax-regret objective, by design, leads to solvers that perform robustly across the prompt space, thus gives the worst-case guarantee. While exact equilibrium may not be attainable with approximation, our empirical results in § E.1 demonstrate robustness.

**Auto-curricula for the players.** With the stochastic policy, the advantage may be heuristically understood as the reward difference between *a base solver* and *a reference solver*. Rather than optimizing separate solvers [Dennis et al., 2020], we sample multiple times from the same policy to create the pair. In this way, the creator is incentivized to produce new prompts that are just out of the comfort zone of solvers [Chaiklin et al., 2003]:

- For overly challenging prompts, both solutions perform poorly, leading to a low proxy.
- For overly easy prompts, the base solution already performs well, again giving a low proxy.
- The optimal strategy is to find prompts that are just beyond the solver’s current capability.

**Auto-curricula inherent to Contrastive Optimization.** Contrastive preference optimization generalizes DPO and a family of algorithms (*cf.*, Hejna et al. [2023], Rafailov et al. [2023], Tang et al. [2024]), many of whose losses monotonically decrease as the contrastive ratio increases. Here, by Eq. 5 and Eq. 10, the *contrastive ratio* can be written via the *advantage-based proxy*:

$$A_{\min}^*(\mathbf{x}) = \beta \cdot \Delta_{\theta^*, \text{ref}}^{\mathbf{x}}. \quad (11)$$

By our proxy, we implicitly incentivize the creator to generate prompts that *bring the most contrastive responses*, which decrease the loss the most. This matches the curriculum learning literature, which prioritizes (in our case, *generatively* prioritizes) examples with smaller losses for better convergence and generalization [Bengio et al., 2009]. We hereby suggest the Contrastive Curriculum Hypothesis: in contrastive preference optimization, prioritizing prompts with higher contrastive ratio improves sample efficiency and generalization. We show initial empirical results on this in § E.1 and § E.4.

## D Additional Experimental Settings

**Datasets and models for training.** We use UltraFeedback [Cui et al., 2023] as the training dataset, which contains diverse high-quality prompts that are primarily human-generated. We use the instruction-finetuned GEMMA-2-9B [Team et al., 2024] as the primary model, which is a strong baseline for models of its size. In training, Each iteration uses 10K prompts (*i.e.*, 1/6 partition from UltraFeedback in classical training). We denote  $\theta_{t \rightarrow t+1}$  as the model trained w/ new human prompts based on the  $t$ -th checkpoint, and  $\theta_{t \rightarrow \bar{t}}$  as the one trained w/ evolved prompts from the  $t$ -th checkpoint w/o adding any new human prompts. Detailed re-implementation instructions can be found in § I.

**Evaluation settings.** We choose: (i) **AlpacaEval 2.0** [Dubois et al., 2024], which assesses general instruction following with 805 questions; (ii) **MT-Bench** [Zheng et al., 2023], which evaluates multi-turn instruction following with 80 hard questions in 8 categories; (iii) **Arena-Hard** [Li et al., 2024b], which is derived from 200K user queries on Chatbot Arena with 500 challenging prompts across 250 topics. We use gpt-4-1106 as the judge and gpt-4-0314 as the baseline for win rate.

**Optimization algorithms.** We focus on direct preference optimization and consider the following:

- **With reference policy:** DPO [Rafailov et al., 2023], SPPO [Wu et al., 2024].
- **Without reference policy:** SimPO [Meng et al., 2024], ORPO [Hong et al., 2024].

**Reward models as preference oracles.** We use ARMORM-8B [Wang et al., 2024] as our default reward model as the human-preference proxy, and consider the following for ablation studies:

- **Pointwise:** ARMORM-8B [Wang et al., 2024], SKYWORKRM-27B [Liu and Zeng, 2024].
- **Pairwise:** PAIRRM-0.4B [Jiang et al., 2023], PAIRRM-8B [Dong et al., 2024].

## E Ablation Studies

### E.1 The Choice of Informativeness Metrics: $\text{info}(\cdot)$

Metric	$\text{info}(\mathbf{x})$	Related Interpretations
$A_{\min}^*$ : worst-case optimal advantage	$ \min_{\mathbf{y}} r(\mathbf{x}, \mathbf{y}) - \max_{\mathbf{y}'} r(\mathbf{x}, \mathbf{y}') $	minimax regret [Savage, 1951]
$A_{\text{avg}}^*$ : average optimal advantage	$ \frac{1}{N} \sum_{\mathbf{y}} r(\mathbf{x}, \mathbf{y}) - \max_{\mathbf{y}'} r(\mathbf{x}, \mathbf{y}') $	Bayesian regret [Banos, 1968]
$A_{\text{dts}}^*$ : dueling optimal advantage	$ \max_{\mathbf{y} \neq \mathbf{y}^*} r(\mathbf{x}, \mathbf{y}) - \max_{\mathbf{y}'} r(\mathbf{x}, \mathbf{y}') $	dueling regret [Wu and Liu, 2016]

Table 2: The reward-advantage-based metrics that serve as the informativeness proxies for prompts.

Model Family ( $\rightarrow$ )	GEMMA-2-9B-IT					
Benchmark ( $\rightarrow$ )	Arena-Hard	MT-Bench			AlpacaEval 2.0	
Method ( $\downarrow$ ) / Metric ( $\rightarrow$ )	WR (%)	avg. score	1 <sup>st</sup> turn	2 <sup>nd</sup> turn	LC-WR (%)	WR (%)
$\theta_{0 \rightarrow 1}$ : DPO	51.6	8.66	9.01	8.32	55.01	51.68
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> (uniform)	57.5	8.71	9.02	8.40	53.43	53.98
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $\text{var}(\mathbf{r})$ )	54.8	8.66	9.13	8.20	54.58	52.55
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $\text{avg}(\mathbf{r})$ )	58.5	8.76	9.13	8.40	55.01	55.47
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $1/\text{avg}(\mathbf{r})$ )	56.7	8.79	9.13	8.45	55.04	54.97
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $1/A_{\min}^*$ )	52.3	8.64	8.96	8.31	53.84	52.92
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $A_{\text{avg}}^*$ ) (our variant)	60.0	8.85	9.08	8.61	<b>56.01</b>	<b>56.46</b>
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $A_{\text{dts}}^*$ ) (our variant)	60.0	8.86	<b>9.18</b>	8.52	55.96	56.09
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> ( $A_{\min}^*$ ) (our default)	<b>60.1</b> (+8.5)	<b>8.90</b>	9.04	<b>8.75</b> (+0.43)	55.35	55.53

Table 3: **Choice of informativeness metric.** Our informativeness metric by *advantage* achieves the best performances, comparing with others as the weight to sample prompts to evolve by the creator.

**Advantage as the informativeness metric outperforms baselines.** As in Table 3, **eva** offers an effective curriculum by the advantage-based proxy as the informativeness metric (bottom row):

- Comparing w/ *uniform evolving* (brown): Existing baselines generate prompts in a uniform manner [Yuan et al., 2024] w/o informativeness measure (*cf.*, the principle of insufficient reason [Keynes, 1921, Tobin et al., 2017]). Ours (red) concretely outperform, corroborating Das et al. [2024] that uniform learners can suffer sub-optimality gaps.
- Comparing w/ *other heuristics* (blue): Prior practices [Team et al., 2023] tried heuristics like prioritizing prompts w/ the most variance in its rewards or w/ the lowest/highest average. We find our advantage based methods (red) outperforms those heuristics.
- Comparing w/ the *inverse advantage* (purple): Contrary to curriculum learning, a line of works conjecture that examples w/ higher losses may be prioritized [Jiang et al., 2019,

Kawaguchi and Lu, 2020], which can be done by inverting our metric. We find it significantly *hurt* the alignment gain, corroborating Mindermann et al. [2022] that those examples are often noisy, unlearnable or irrelevant, meaning our curriculum is effective and practical.

- Among our *advantage variants* (red): We designed variants of our default advantage-based metric, as in Table 2; the default  $A_{\min}^*$  remains competitive among its peers. Together, the advantage-based principle provides a robust guideline for prompt sampling and evolving.

The lesson is that we must be selective about which are the promising to evolve, otherwise unlearnable, noisy or nave prompts may hinder learning. Our regret-inspired metric represents a solid baseline.

## E.2 The Effect of the sample-then-evolve procedure

Benchmark (→)	Arena-Hard	MT-Bench			AlpacaEval 2.0	
Method (↓) / Metric (→)	WR (%)	avg. score	1 <sup>st</sup> turn	2 <sup>nd</sup> turn	LC-WR (%)	WR (%)
$\theta_{0 \rightarrow 1}$ : DPO	51.6	8.66	9.01	8.32	55.01	51.68
$\theta_{1 \rightarrow \bar{1}}$ : [no evolve]-greedy	56.1	8.68	8.98	8.38	54.11	53.66
$\theta_{1 \rightarrow \bar{1}}$ : [no evolve]-sample	55.3	8.69	9.00	8.38	54.22	54.16
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> -greedy (our variant)	59.5	8.72	9.06	8.36	54.52	55.22
$\theta_{1 \rightarrow \bar{1}}$ : + <b>eva</b> -sample (our default)	<b>60.1</b>	<b>8.90</b>	9.04	<b>8.75</b>	<b>55.35</b>	<b>55.53</b>

Table 4: **Effect of evolving.** The blue are those training w/ only the informative subset and w/o evolving; we denote `-sample` for the default weighted sampling procedure in Algo 1, while using `-greedy` for the variant from the classical active data selection procedure (*cf.*, a recent work [Muldrew et al., 2024] and a pre-LLM work [Kawaguchi and Lu, 2020]), which selects data by a high-to-low ranking via the metric greedily. We show evolving brings a remarkable alignment gain (the red v.s. the blue); and as we evolve, sampling is more robust than being greedy (*cf.*, Russo et al. [2018]).

**The design of `evolve(·)` in **eva** is effective.** As in Table 4, we show:

- Removing the `evolve(·)` step: if we only do subset sampling or ordered selection, we still have gain, but not as much as w/ evolving (*e.g.*, **eva** brings 4.8% additional wins on AH).
- Altering the `sample(·)` step: if we greedily select prompts by the metric instead of using them as weights for importance sampling, the performance will be weaker as we evolve.

This shows that simply adaptive training within a fixed prompt distribution is unsatisfactory; our open-ended RLHF with *generative* prompt exploration gives a substantial headroom for self-improvement.

## E.3 Scaling Pointwise and Pairwise Reward Models

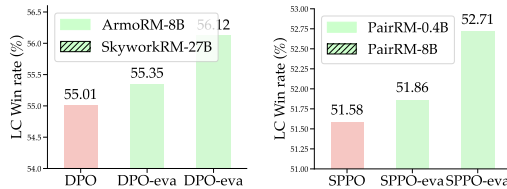


Figure 4: **eva scales with quality of reward models**, under pointwise RMs w/ DPO (*left*) and pairwise RMs w/ SPPO (*right*). Note SPPO handles general preferences thus requires pairwise RMs, and DPO relies on the Bradley-Terry assumption, for which pointwise RMs are suitable.

Figure 4 presents the length-controlled win rate of **eva** on AlpacaEval using pointwise and pairwise reward models of varying scales. The results give a clear trend: as the quality of reward models improve, **eva** brings higher alignment gain. The scaling observation shows the effectiveness of **eva** in exploiting more accurate reward signals to choose informative prompts for better alignment. One takeaway is interaction w/ the external world is essential for intelligence. The more accurate reward signals observed, the better the agent incentivize themselves to improve (*cf.*, Silver et al. [2021]).

## E.4 **eva** Improves Both Sample Efficiency and Generalization

We then continuously run the default *incremental training* (i.e., training from the last checkpoint w/ the evolved set in each iteration), as in Fig 5, **eva** presents *monotonic performance gain* over iterations, and surpasses that trained w/ new human prompts, implying the generalization benefit. Specifically, we conjecture that behaviors of the dashed/dotted lines relate to *loss of plasticity* [Ash and Adams, 2019, Dohare et al., 2023, Abbas et al., 2023, Xue et al., 2024]. Classical works resolve it by the optimization view (e.g., weight perturbing), whereas **eva** provides a view from data scheduling, potentially mimicking an *implicit regularizer for better generalization*.

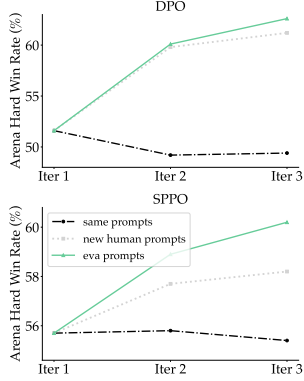


Figure 5: **Continual training.** **eva** stays robust w/ more iterations in incremental training.

The solutions found by **eva** cannot be recovered by training longer w/ a fixed distribution (the dashed), nor by naively sourcing new prompts w/o examining informativeness (the gray dotted), thus our generative data schedule is effective.

In Table 5, we ablate **eva** in *scratch training*, i.e., training w/ the full set (the evolved and the original data). **eva** is competitive in incremental training, thus *learns more effective with less data* – a nice bonus via minimax regret [Jiang et al., 2021a].

Benchmark (→)	Arena-Hard	MT-Bench	AlpacaEval 2.0
Method (↓) / Metric (→)	WR (%)	avg. score	LC-WR (%)
$\theta_0$ : SFT	41.3	8.57	47.11
$\theta_{0 \rightarrow 1}$ : DPO	51.6	8.66	55.01
$\theta_{0 \rightarrow 1}$ : <b>eva</b> (scratch)	59.8	8.88	54.59
$\theta_{1 \rightarrow 1}$ : <b>eva</b> (incremental)	<b>60.1</b>	<b>8.90</b>	<b>55.35</b>

Table 5: **Ablation on incremental v.s. scratch training.**

## F Additional Experimental Results

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 [Wang et al., 2024, Liu and Zeng, 2024].

Method (↓) / Dataset (→)	MUSR-TA	TruthfulQA-Gen	WMDP	GSM8K	GSM-Plus	MMLU-Pro
$\theta_0$ : SFT	38.80	34.76	58.62	24.64	18.62	52.08
$\theta_{0 \rightarrow 1}$ : DPO	38.40	34.76	58.45	24.56	18.50	52.63
$\theta_{1 \rightarrow 1}$ : <b>+ eva</b>	38.40	34.15	58.40	24.26	17.96	<b>53.03</b>
$\theta_{0 \rightarrow 1}$ : SPPO	40.80	34.15	58.72	24.79	18.42	52.70
$\theta_{1 \rightarrow 1}$ : <b>+ eva</b>	<b>41.20</b>	34.64	<b>58.94</b>	<b>25.40</b>	<b>18.88</b>	52.47

Table 6: **Performance on Downstream tasks.**

Model Family (→)	GEMMA-2-9B-IT							
Benchmark (→)	MT-Bench			Arena-Hard		AlpacaEval 2.0		
Method (↓) / Metric (→)	avg. score	1 <sup>st</sup> turn	2 <sup>nd</sup> turn	WR (%)	avg. len	LC (%)	WR (%)	avg. len
$\theta_{0 \rightarrow 1}$ : DPO	8.66	9.01	8.32	51.6	651	55.01	51.68	1883
$\theta_{1 \rightarrow 1}$ : <b>+ eva-i</b> (ARMO-8B)	<b>8.90</b>	9.04	8.75	60.1	725	55.35	55.53	2089
$\theta_{1 \rightarrow 1}$ : <b>+ eva-i</b> (SKYWORKRM-27B)	8.75	9.07	8.43	<b>60.3</b>	744	<b>56.12</b>	<b>56.40</b>	2097

Table 7: **Effect of (pointwise) reward models.**

Model Family (→)	GEMMA-2-9B-IT							
Benchmark (→)	MT-Bench			Arena-Hard		AlpacaEval 2.0		
Method (↓) / Metric (→)	avg. score	1 <sup>st</sup> turn	2 <sup>nd</sup> turn	WR (%)	avg. len	LC (%)	WR (%)	avg. len
$\theta_{0 \rightarrow 1}$ : SPPO	8.62	9.03	8.21	55.7	560	51.58	42.17	1850
$\theta_{1 \rightarrow 1}$ : <b>+ eva-i</b> (PAIRRM-0.4B)	8.78	<b>9.11</b>	8.45	58.9	567	51.86	43.04	1870
$\theta_{1 \rightarrow 1}$ : <b>+ eva-i</b> (PAIRRM-8B)	<b>8.89</b>	9.08	<b>8.70</b>	<b>60.2</b>	563	<b>52.71</b>	<b>44.52</b>	1848

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

## G Related Works

**Self-improving algorithms and iterative optimization.** This line of work focuses on iteratively generating samples from the response policy and continuously re-training the policy by selected self-generated samples. Major works include ReST [Gulcehre et al., 2023, Singh et al., 2023], STaR [Zelikman et al., 2022], RFT [Yuan et al., 2023], RAFT [Dong et al., 2023], self-improving LLMs [Huang et al., 2022, Yuan et al., 2024]; in the context of preference optimization, iterative DPO [Xu et al., 2023b, Tajwar et al., 2024, Tran et al., 2023, Xiong et al., 2024, Pang et al., 2024] has proven effective. Most works focus on self-training by improving in  $\mathcal{Y} \mid \mathcal{X}$ , while we *jointly optimize* both responses and prompts via generative exploration in the  $(\mathcal{X}, \mathcal{Y})$  space. Among them, we also distinctly present a game-theoretic framework with the minimax-regret principle as the guidance.

**Prompt synthesis for language models.** Existing works include Self-Instruct [Wang et al., 2022], WizardLM [Xu et al., 2023a, Luo et al., 2023], Self-Align [Sun et al., 2024], Glan [Li et al., 2024a], EvoPrompt [Guo et al., 2023], Magpie [Xu et al., 2024] and others [Long et al., 2024]. **eva** is an orthogonal contribution since any synthesis method can be plugged in as the `evolve(·)` for the creator. Importantly, our work presents a new reward-related metric to endow prompt the notion of informativeness, with new implications as in § C. We also focus on preference optimization algorithms, while those existing works primarily use synthesized prompts in an SFT-only way.

**Self-play and curriculum RL.** Agents trained on a fixed data distribution are often brittle and may struggle to adapt to the real world [Hughes et al., 2024]. Self-play [Samuel, 1959, Goodfellow et al., 2014, Silver et al., 2016] addresses this by having the agent learn through self-interaction, thus creating more diverse experiences and automatic curricula. In asymmetric self-play, the paradigm centers on “*Alice proposing a task, and Bob doing it*” [Sukhbaatar et al., 2017, Samvelyan et al., 2023, Beukman et al., 2024, Dennis et al., 2020]. Other nice game-theoretic frameworks that guide learning agents include Bai et al. [2020], Anil et al. [2021], Rajeswaran et al. [2020]. We revive the classical asymmetric self-play principle [Sutton et al., 2011] in optimizing language models. Unlike traditional curriculum RL [Parker-Holder et al., 2022], which usually renders environments from specified levels [Dennis et al., 2020], our approach is *generative* by nature, as we directly generate contexts from the auto-regressive language models.

**Self-play in RLHF.** A growing line of research frames RLHF as a *symmetric* self-play game, where both players are response players [Munos et al., 2023, Wu et al., 2024, Choi et al., 2024, Rosset et al., 2024]. However, these methods still rely on a fixed prompt distribution thus is sub-optimal. In contrast, we solve this by *asymmetric* self-play, enabling evolving prompt distributions for more generalizable language agents. During our work, we notice one concurrent paper adopting the asymmetric two-player setup [Zheng et al., 2024], however (i) it applies to adversarial attack tasks instead of general alignment benchmarks, (ii) it is incompatible w/ direct preference optimization, and (iii) it relies on the maxmin principle (which is known to be producing unlearnable environments [Dennis et al., 2020]) instead of the minimax *regret* principle [Fan, 1953, Savage, 1951] as we do. We also first precisely defined the new problem of open-ended RLHF, which generalizes over classical RLHF.

## H Future Directions

**eva** defines a new paradigm for alignment, opening up many new directions, *e.g.*, (i) extending to differentiable creator policies, combining w/ other `evolve(·)` methods; (ii) evolving for more iterations w/ on-policy solvers like RLOO [Ahmadian et al., 2024]; (iii) investigating exploration bonuses for diversity, coverage and extrapolation, and self-consuming loops [Gerstgrasser et al., 2024]; (iv) extending the game with more players for full automation (*e.g.*, rewarders, critics, rewriters, verifiers, retrievers); (v) extending from alignment to reasoning (*e.g.*, auto-conjecturing in theorem proving [Poesia et al., 2024] can be cast as asymmetric games), w/ process reward models and hierarchical tree search for creator and solver generations; (vii) exploring other metric like Fisher information for theoretical guarantees; (vii) scaling up w/ million-level data.

## I Appendix on Reproducibility

As mentioned, currently we put supplementary materials and additional experiment results on [anonymous.4open.science/r/eva-i-workshop/](https://anonymous.4open.science/r/eva-i-workshop/), where we intend to release code there, and all the datasets (generated prompts and responses) and models (with other models beyond the current family), *upon approval* – before then, we are more than happy to provide any clarification requested to help re-implement **eva** and replicate our results.

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 ( $\downarrow$ ) / Loss ( $\rightarrow$ )	DPO	ORPO	SimPO	SPPO
learning rate	5e-7	5e-7	8e-7	5e-7
learning rate scheduler	cosine	cosine	cosine	linear
$\beta$	0.05	/	10	0.001
$\gamma$	/	/	5	/
$\lambda$	/	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:

- $\theta_0$  : Base SFT model.
- $\theta_{0 \rightarrow 1}$  : initialize with  $\theta_0$ ; then train with the prompt split  $\mathcal{X}_1$  by self-generated responses from the initial model  $\theta_0$ .
- $\theta_{1 \rightarrow 2}$  : initialize with  $\theta_{0 \rightarrow 1}$ ; trained with the prompt split  $\mathcal{X}_2$  via by self-generated responses from the initial model  $\theta_{0 \rightarrow 1}$ .

For evolving prompts (*e.g.*, evolving  $\mathcal{X}_1$  to  $\mathcal{X}_1^*$ ), 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  $\mathcal{X}_1^{\text{info}}$ . We then iterate over in  $\mathcal{X}_1^{\text{info}}$  and call `EvolInstruct` [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  $\mathcal{X}_1^*$ . 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 our 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, OpenAI (default to be `gpt-4o-mini`) and transformers models. 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.

## J Examples on Prompts and Model Generations

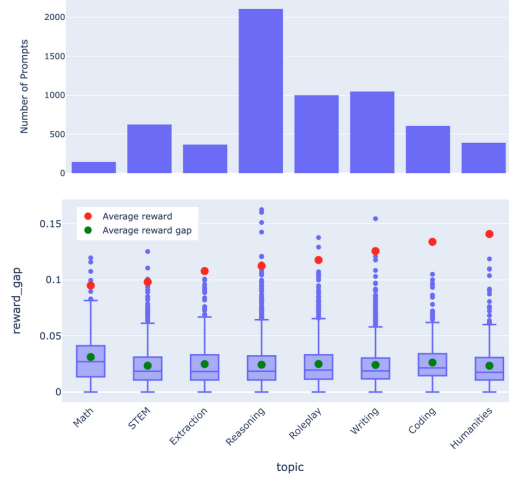


Figure 6: The initial prompt distribution of AlpacaEval by bart-large-mnli with 0-shot classification, which is imbalanced. For the reward distribution, the category with lowest average reward has the highest reward gap (*i.e.*, the default informativeness proxy), implying the potential to improve.

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 9: 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.

<b>1st-turn prompt:</b> When a number is divided by 10, the remainder is 4. What is the remainder when twice the number is divided by 4?	
<b>Answer from gemma-2-9b-it-dpo</b> 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!	<b>Answer from gemma-2-9b-it-dpo-eva</b> 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.
<b>2nd-turn prompt:</b> What about when twice the number is divided by 5?	
<b>Answer from gemma-2-9b-it-dpo</b> 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!	<b>Answer from gemma-2-9b-it-dpo-eva</b> 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.
<b>Judgment</b> (by gpt-4)	
<b>Judgment for gemma-2-9b-it-dpo</b> 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.	<b>Judgment for gemma-2-9b-it-dpo-eva</b> 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: <code>[[10]], "score": 10, "turn": 2, "tstamp": 1722846518.5543818</code>
Rating: <code>[[3]], "score": 3, "turn": 2, "tstamp": 1722737011.7768948</code>	

Table 10: Comparison of model generations on examples from MT-Bench.

<b>1st-turn prompt:</b> 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.	
<b>Answer from gemma-2-9b-it-dpo</b> <p>## Smartphone Showdown: [Model A] vs. [Model B]\n\nI. 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\nII. 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\nIII. Performance and Specs**\n* Processor, RAM, and storage comparison.\n* Benchmark scores and real-world performance analysis.\n* Battery life and charging capabilities.\n\nIV. Camera Capabilities**\n* Rear and front camera specs (megapixels, aperture, features).\n* Image quality comparison in various lighting conditions.\n* Video recording capabilities.\n\nV. Software and User Experience**\n* Operating system and user interface comparison.\n* App ecosystem and availability.\n* Software features and customization options.\n\nVI. Price and Value**\n* Compare the price points of both models.\n* Analyze the value proposition based on features and performance.\n\nVII. Conclusion**\n* Summarize the key differences and similarities.\n* Recommend the best phone for specific user profiles (e.g., budget-conscious, photography enthusiasts, gamers).</p>	<b>Answer from gemma-2-9b-it-dpo-eva</b> <p>## iPhone 14 Pro vs. Samsung Galaxy S23: Which Flagship Reigns Supreme?\n\nI. 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\nII. 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\nIII. 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\nIV. 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\nV. Battery Life and Charging**\n* Battery capacity and real-world usage comparison.\n* Fast charging speeds and wireless charging capabilities.\n\nVI. Price and Value**\n* Compare starting prices and different storage configurations.\n* Assess the overall value for money based on features and performance.\n\nVII. 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).</p>
<b>2nd-turn prompt:</b> Take your previous response and rephrase it as a limerick.	
<b>Answer from gemma-2-9b-it-dpo</b> <p>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!</p>	<b>Answer from gemma-2-9b-it-dpo-eva</b> <p>\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!</p>
<b>Judgment</b> (by gpt-4)	
<b>Judgment for gemma-2-9b-it-dpo</b> <p>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.</p> <p>Rating: [[8]], "score": 8, "turn": 2, "tstamp": 1722737024.0327556</p>	<b>Judgment for gemma-2-9b-it-dpo-eva</b> <p>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.</p> <p>Rating: [[9]], "score": 9, "turn": 2, "tstamp": 1722846542.9437995</p>

Table 11: (Continued) comparison of model generations on examples from MT-Bench.

<b>1st-turn prompt:</b> 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?	
<b>Answer from gemma-2-9b-it-dpo</b> 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\nThe part "Bananas > Apples" is true based on our price order.\n\nThe part "Bananas > Oranges" is also true based on our price order.\n\nLet me know if you'd like to try another one!	<b>Answer from gemma-2-9b-it-dpo-eva</b> 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\nLet me know if you'd like to try another one!
<b>2nd-turn prompt:</b> If the third statement is true. Is the first statement true, false, or uncertain? Please explain.	
<b>Answer from gemma-2-9b-it-dpo</b> 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\nLet me know if you have any other logic puzzles!	<b>Answer from gemma-2-9b-it-dpo-eva</b> 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.
<b>Judgment (by gpt-4)</b>	
<b>Judgment for gemma-2-9b-it-dpo</b> 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. \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.	<b>Judgment for gemma-2-9b-it-dpo-eva</b> 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. \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: [[2]], "score": 2, "turn": 2, "timestamp": 1722737017.2808955	Rating: [[10]], "score": 10, "turn": 2, "timestamp": 1722846527.2339694

Table 12: (Continued) comparison of model generations on examples from MT-Bench.

<p><b>1st-turn prompt:</b> Implement a function to find the median of two sorted arrays of different sizes with <math>O(1)</math> space complexity and <math>O(n)</math> time complexity.</p>	
<p><b>Answer from gemma-2-9b-it-dpo</b></p> <pre>python\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    Returns:\n        The median of the two sorted arrays.\n    """\n    m, n = len(nums1), len(nums2)\n    if m &gt; n:\n        nums1, nums2, m, n = nums2, nums1, n, m # Ensure m &lt;= n\n    imin, imax, halflen = 0, m, (m + n + 1) // 2\n    while imin &lt;= imax:\n        i = (imin + imax) // 2\n        j = halflen - i\n        if i &lt; m and nums2[j - 1] &gt; nums1[i]:\n            imin = i + 1 # i is too small, increase it\n        elif i &gt; 0 and nums1[i - 1] &gt; nums2[j]:\n            imax = i - 1 # i is too big, decrease it\n        else:\n            # i is perfect\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            if (m + n) % 2 == 1:\n                return max_left\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            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# Explanation:\n# 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# 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# - 'halflen' calculates the desired total number of elements on the left side of the partition (to find the median).\n# Partitioning:\n# - In each iteration, 'i' divides 'nums1' into two parts: 'nums1[0:i]' and 'nums1[i:m]'. 'j' is calculated to divide 'nums2' so that the total number of elements on the left side is 'halflen'. # 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] &gt; nums1[i]', 'i' is too small, so we move 'imin' to 'i + 1'. # - If 'nums1[i - 1] &gt; nums2[j]', 'i' is too large, so we move 'imax' to 'i - 1'. # 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. # - If the total length is odd, 'max_left' is the median. # - If the total length is even, the median is the average of 'max_left' and 'min_right'. Time Complexity:\n# O(log(min(m, n))) because we perform binary search on the shorter array.\nSpace Complexity:\n# O(1) as we only use a constant amount of extra space for variables.</pre>	<p><b>Answer from gemma-2-9b-it-dpo-eva</b></p> <pre>python\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    Returns:\n        The median of the two sorted arrays.\n    """\n    m, n = len(nums1), len(nums2)\n    if m &gt; n:\n        nums1, nums2, m, n = nums2, nums1, n, m # Ensure m &lt;= n\n    imin, imax, halflen = 0, m, (m + n + 1) // 2\n    while imin &lt;= imax:\n        i = (imin + imax) // 2\n        j = halflen - i\n        if i &lt; m and nums2[j - 1] &gt; nums1[i]:\n            imin = i + 1 # i is too small, increase it\n        elif i &gt; 0 and nums1[i - 1] &gt; nums2[j]:\n            imax = i - 1 # i is too big, decrease it\n        else:\n            # i is perfect\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            if (m + n) % 2 == 1:\n                return max_left\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            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# Explanation:\n# 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# 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# - 'halflen' calculates the desired total number of elements on the left side of the partition (to find the median).\n# Partitioning:\n# - In each iteration, 'i' divides 'nums1', and 'j' divides 'nums2' such that the total number of elements on the left side is 'halflen'. # Adjusting the Search Range:\n# - If 'nums2[j - 1] &gt; nums1[i]', it means 'i' is too small, so we need to move the partition point in 'nums1' to the right ('imin = i + 1'). # - If 'nums1[i - 1] &gt; nums2[j]', 'i' is too large, so we move it to the left ('imax = i - 1'). # 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. # - If even, the median is the average of 'max_left' and 'min_right'. Time Complexity:\n# O(log(min(m, n))) because we are performing binary search on the shorter array.\nSpace Complexity:\n# O(1) as we are only using a constant amount of extra space for variables.</pre>
<p><b>Judgment</b> (by gpt-4)</p>	
<p><b>Judgment for gemma-2-9b-it-dpo</b></p> <p>The assistant's answer is correct and provides a detailed explanation of the code. The assistant correctly states that the time complexity is <math>O(\log(\min(m, n)))</math> and the space complexity is <math>O(1)</math>, 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 <math>O(n)</math> time complexity. The assistant should have clarified that it's not possible to find the median of two sorted arrays of different sizes with <math>O(1)</math> space complexity and <math>O(n)</math> time complexity, as stated in the reference answer.</p> <p>Rating: <code>[[8]], "score": 8, "turn": 1, "timestamp": 1722737514.2011044</code></p>	<p><b>Judgment for gemma-2-9b-it-dpo-eva</b></p> <p>The assistant's answer is correct and provides a detailed explanation of the code. The assistant correctly states that the time complexity is <math>O(\log(\min(m, n)))</math> and the space complexity is <math>O(1)</math>, 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.</p> <p>Rating: <code>[[10]], "score": 10, "turn": 1, "timestamp": 1722846533.772981</code></p>

Table 13: (Continued) comparison of model generations on examples from MT-Bench.

<b>1st-turn prompt:</b> 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.	
<b>Answer from gemma-2-9b-it-dpo</b> <pre> '''python def find_kth_smallest(list1, list2, k):     """     Finds the kth smallest element in the union     of two sorted lists.     Args:         list1: The first sorted list.         list2: The second sorted list.         k: The desired kth smallest element.     Returns:         The kth smallest element in the union of the two lists.     """     m, n = len(list1), len(list2)     i, j = 0, 0     min_heap = []     # Build a min-heap with the first elements of both lists     for x in range(min(m, n)):         heapq.heappush(min_heap, (list1[x], x, 0))         # (value, list_index, position_in_list)     heapq.heappush(min_heap, (list2[0], 0, 1))     # Extract k smallest elements from the heap     for _ in range(k - 1):         _, _, _ = heapq.heappop(min_heap)     # The top element of the heap is the kth smallest     return heapq.heappop(min_heap)[0] # Example usage: list1 = [2, 5, 8, 12] list2 = [1, 3, 7, 9, 10] nk = 5 kth_smallest = find_kth_smallest(list1, list2, k) print(f"The {k}th smallest element is: {kth_smallest}") """ Explanation: - Initialization: 'm' and 'n' store the lengths of the input lists. 'i' and 'j' are pointers to track the current position in each list. 'min_heap' is a min-heap used to efficiently store and retrieve the smallest elements encountered so far. - Building the Min-Heap: The code iterates through the first 'min(m, n)' elements of both lists. 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. - Extracting k-1 Smallest Elements: The 'for' loop runs 'k - 1' times, effectively extracting the 'k-1' smallest elements from the 'min_heap'. - Finding the kth Smallest: After the loop, the top element of the 'min_heap' contains the 'k'th smallest element. 'heapq.heappop(min_heap)' removes and returns this element. - Time Complexity: Building the min-heap: O(m + n). Extracting k-1 elements: O(k * log(k)). Overall: O(m + n + k * log(k)), which is approximately O(m + n) since 'k' is usually much smaller than 'm + n'. Let me know if you have any other questions! </pre>	<b>Answer from gemma-2-9b-it-dpo-eva</b> <pre> '''python def find_kth_smallest(list1, list2, k):     """     Finds the kth smallest element in the union     of two sorted lists.     Args:         list1: The first sorted list.         list2: The second sorted list.         k: The desired rank of the smallest element.     Returns:         The kth smallest element in the union of the lists.     """     i = 0     # Index for list1     j = 0     # Index for list2     merged = []     # Store the merged elements     while i &lt; len(list1) and j &lt; len(list2):         if list1[i] &lt;= list2[j]:             merged.append(list1[i])             i += 1         else:             merged.append(list2[j])             j += 1     # Add any remaining elements from list1     while i &lt; len(list1):         merged.append(list1[i])         i += 1     # Add any remaining elements from list2     while j &lt; len(list2):         merged.append(list2[j])         j += 1     return merged[k - 1] # Return the kth element (index k-1) # Example usage: list1 = [2, 5, 8, 12] list2 = [1, 3, 7, 9, 10] nk = 5 kth_smallest = find_kth_smallest(list1, list2, k) print(f"The {k}th smallest element is: {kth_smallest}") """ Explanation: - Initialization: 'i' and 'j' are index pointers for 'list1' and 'list2' respectively, starting at 0. 'merged' is an empty list to store the merged sorted elements. - Merging: The 'while' loop continues as long as both 'i' and 'j' are within the bounds of their respective lists. In each iteration, it compares the elements at 'list1[i]' and 'list2[j]'. The smaller element is appended to 'merged', and the corresponding index ('i' or 'j') is incremented. - Adding Remaining Elements: After one list is exhausted, the remaining elements from the other list are directly appended to 'merged'. - Returning the kth Element: Finally, 'merged[k - 1]' is returned, which represents the kth smallest element in the union (remember that list indices start from 0). - Linear Complexity: The algorithm iterates through each element in both lists at most once during the merging process. 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. </pre>
<b>Judgment</b> (by gpt-4)	
<b>Judgment for gemma-2-9b-it-dpo</b> <p>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 <math>O(m + n + k \cdot \log(k))</math>, which is not linear. The assistant's solution uses a min-heap, which is not necessary and adds unnecessary complexity to the solution.</p> <p>Rating: <code>[[2]]</code>, "score": 2, "turn": 1, "timestamp": 1722737030.6411152</p>	<b>Judgment for gemma-2-9b-it-dpo-eva</b> <p>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.</p> <p>Rating: <code>[[8]]</code>, "score": 8, "turn": 1, "timestamp": 1722846543.867601</p>

Table 14: (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.	
<p><b>Answer from gemma-2-9b-it-dpo</b></p> <p>You are absolutely right! There's a more efficient algorithm with a time complexity of <math>O(m + n)</math> without using a heap.</p> <p>Here's the implementation:</p> <pre> def find_kth_smallest_optimized(list1, list2, k):     """Finds the kth smallest element in the union of two sorted lists with O(m + n) time complexity.      Args:         list1: The first sorted list.         list2: The second sorted list.         k: The desired kth smallest element.      Returns:         The kth smallest element in the union of the two lists.      """     m, n = len(list1), len(list2)     # Merge the lists while keeping track of the kth smallest     result = []     i, j = 0, 0     while len(result) &lt; k:         if list1[i] &lt;= list2[j]:             result.append(list1[i])             i += 1         else:             result.append(list2[j])             j += 1     elif i &lt; m:         result.append(list1[i])         i += 1     else:         result.append(list2[j])         j += 1     return result[k - 1]  # Example usage list1 = [2, 5, 8, 12] list2 = [1, 3, 7, 9, 10] k = 5 kth_smallest = find_kth_smallest_optimized(list1, list2, k) print(f"The {k}th smallest element is: {kth_smallest}")  """Explanation: Initialization: 'm' and 'n' store the lengths of the lists. 'i' and 'j' are pointers for 'list1' and 'list2', respectively. 'result' is a list to store the merged and sorted elements. Merging and Tracking: The 'while' loop continues until 'k' elements are in the 'result' list. In each iteration, it compares the current elements from 'list1' and 'list2'. The smaller element is appended to 'result', and the corresponding pointer ('i' or 'j') is incremented. If one list is exhausted, the remaining elements from the other list are appended to 'result'. Returning the kth Smallest: Finally, 'result[k - 1]' contains the 'k'th smallest element. Time Complexity: The algorithm iterates through both lists once, resulting in a time complexity of O(m + n). Let me know if you have any more questions! """ </pre>	<p><b>Answer from gemma-2-9b-it-dpo-eva</b></p> <p>You are right to ask! While the previous algorithm has linear time complexity, we can achieve an even better solution with <math>O(\log(\min(m, n)))</math> time complexity using a slightly more sophisticated approach.</p> <p>The key idea is to leverage the sorted nature of the lists and use a binary search strategy.</p> <p>Here's the implementation:</p> <pre> def find_kth_smallest_optimized(list1, list2, k):     """Finds the kth smallest element in the union of two sorted lists with O(log(min(m, n))) time complexity.      Args:         list1: The first sorted list.         list2: The second sorted list.         k: The desired rank of the smallest element.      Returns:         The kth smallest element in the union of the lists.      """     m, n = len(list1), len(list2)     if m &gt; n:         list1, list2, m, n = list2, list1, n, m     # Ensure list1 is the shorter one     left, right = 0, m     while left &lt; right:         partitionX = (left + right) // 2         partitionY = k - partitionX         maxLeftX = list1[partitionX - 1] if partitionX &gt; 0 else float('-inf')         minRightX = list1[partitionX] if partitionX &lt; m else float('inf')         maxLeftY = list2[partitionY - 1] if partitionY &gt; 0 else float('-inf')         minRightY = list2[partitionY] if partitionY &lt; n else float('inf')         if maxLeftX &lt;= minRightY and maxLeftY &lt;= minRightX:             return list2[partitionY - 1] if partitionY == 0:             return list1[partitionX - 1]         elif maxLeftX &gt; minRightY:             right = partitionX - 1         # Too many elements on the right side of list1         elif maxLeftY &gt; minRightX:             left = partitionX + 1         # Too many elements on the right side of list2     """Explanation: Shorten Lists: If 'list1' is longer than 'list2', swap them to ensure 'list1' is always the shorter one. This simplifies the logic. Binary Search: 'left' and 'right' pointers define the search space within 'list1'. 'partitionX' is the middle point in 'list1', and 'partitionY' is calculated to divide the remaining elements in 'list2' such that 'partitionX + partitionY = k'. 'maxLeftX', 'minRightX', 'maxLeftY', and 'minRightY' represent the largest element to the left and smallest element to the right of the partitions in both lists. Comparison and Adjustment: 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'. If 'maxLeftX' is greater than 'minRightY', it indicates that we need to search in the left subarray of 'list1' (i.e., decrease 'right'). Otherwise, we search in the right subarray of 'list1' (i.e., increase 'left'). Termination: The binary search continues until 'left' crosses 'right', indicating the correct partition has been found. Time Complexity: The binary search divides the search space in half at each step. 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. """ </pre>
<p><b>Judgment</b> (by gpt-4)</p>	
<p><b>Judgment for gemma-2-9b-it-dpo</b></p> <p>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 <math>O(m + n + k \log(k))</math>. 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 <math>O(m + n)</math>, 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 <math>O(\log(m) + \log(n))</math>, as the reference answer correctly does.</p> <p>Rating: <code>[[2]], "score": 2, "turn": 2, "timestamp": 1722737031.5033472</code></p>	<p><b>Judgment for gemma-2-9b-it-dpo-eva</b></p> <p>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.</p> <p>Rating: <code>[[9]], "score": 9, "turn": 2, "timestamp": 1722846556.6828268</code></p>

Table 15: (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.
- 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.
- Cem Anil, Guodong Zhang, Yuhuai Wu, and Roger Grosse. [Learning to give checkable answers with prover-verifier games](#). *arXiv preprint arXiv:2108.12099*, 2021.
- Jordan T Ash and Ryan P Adams. [On the difficulty of warm-starting neural network training](#). *arXiv preprint arXiv:1910.08475*, 2019.
- Yu Bai, Chi Jin, and Tiancheng Yu. [Near-optimal reinforcement learning with self-play](#). *Advances in neural information processing systems*, 33:2159–2170, 2020.
- 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*, 2024.
- Seth Chaiklin et al. [The zone of proximal development in Vygotskys analysis of learning and instruction](#). *Vygotskys educational theory in cultural context*, 1(2):39–64, 2003.
- Eugene Choi, Arash Ahmadian, Matthieu Geist, Olivier 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 Vinitzky, 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.
- 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.

- 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.
- 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.
- 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.
- Joey Hejna, Rafael Rafailov, Harshit Sikchi, Chelsea Finn, Scott Niekum, W Bradley Knox, and Dorsa Sadigh. [Contrastive preference 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 Rocktäschel. [Open-Endedness is Essential for Artificial Superhuman Intelligence](#). *arXiv preprint arXiv:2406.04268*, 2024.
- Angela H Jiang, Daniel L-K Wong, Giulio Zhou, David G Andersen, Jeffrey Dean, Gregory R Ganger, Gauri Joshi, Michael Kaminsky, 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, 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.
- 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). <https://huggingface.co/Skywork>, September 2024. URL <https://huggingface.co/Skywork>.
- 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.
- 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 Hölting, 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.
- 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 Santacrose, 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.
- Hans-Paul Schwefel. [Evolutionsstrategien fr die numerische Optimierung](#). Springer, 1977.
- Claude Elwood Shannon. [A mathematical theory of communication](#). *The Bell system technical journal*, 27(3):379–423, 1948.
- 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.
- 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.
- 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*, 2024.
- 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.
- 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.

- 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.
- 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.
- 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.