Reward Model Overoptimisation in Iterated RLHF

Lorenz Wolf
UCL Centre for Artificial Intelligence
Department of Computer Science
University College London
lorenz.wolf.22@ucl.ac.uk

Robert Kirk
UK AI Security Institute
email

Mirco Musolesi
UCL Centre for Artificial Intelligence
Department of Computer Science
University College London
Department of Computer Science and Engineering
University of Bologna
m.musolesi@ucl.ac.uk

Abstract

Reinforcement learning from human feedback (RLHF) is a widely used method for aligning large language models with human preferences. However, RLHF often suffers from reward model overoptimisation, in which models overfit to the reward function, resulting in non-generalisable policies that exploit the idiosyncrasies and peculiarities of the reward function. A common mitigation is *iterated RLHF*, in which reward models are repeatedly retrained with updated human feedback and policies are re-optimised. Despite its increasing adoption, the dynamics of overoptimisation in this setting remain poorly understood. In this work, we present the first comprehensive study of overoptimisation in iterated RLHF. We systematically analyse key design choices - how reward model training data is transferred across iterations, which reward function is used for optimisation, and how policies are initialised. Using the controlled AlpacaFarm benchmark, we observe that overoptimisation tends to decrease over successive iterations, as reward models increasingly approximate ground-truth preferences. However, performance gains diminish over time, and while reinitialising from the base policy is robust, it limits optimisation flexibility. Other initialisation strategies often fail to recover from early overoptimisation. These findings offer actionable insights for building more stable and generalisable RLHF pipelines.

1 Introduction

Reinforcement learning from human feedback (RLHF) has become the standard method for aligning large language models with human preferences [47, 27, 2]. However, RLHF faces a critical vulnerability: reward model overoptimisation [11]. As fine-tuning progresses, models learn to overfit to the trained reward function - achieving high scores without genuinely satisfying human intent. This creates brittle policies that exploit loopholes rather than developing robust behaviours, leading to systems that appear aligned during training but fail catastrophically when deployed. Iterated RLHF represents a promising approach to combat this problem. By repeatedly collecting new preferences on the latest policy outputs, retraining the reward model, and fine-tuning the policy [2, 41], practitioners aim to iteratively close the gap between proxy and true reward. Despite its widespread adoption in industry [47, 27, 2], it remains uncertain whether iterated RLHF genuinely resolves overoptimisation,

2nd Workshop on Aligning Reinforcement Learning Experimentalists and Theorists (ARLET 2025).

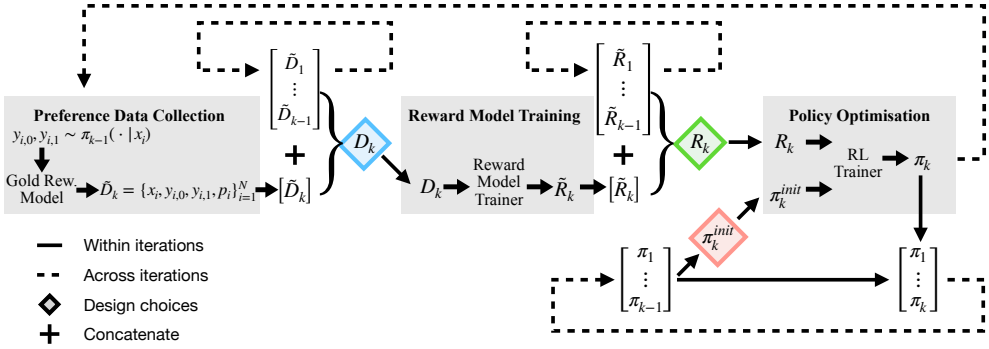


Figure 1: The Iterated RLHF framework performs multiple rounds of preference data collection, reward modelling, and policy optimisation. Our research reveals three design choices that dramatically impact performance: (1) how preference data is managed across iterations, (2) which reward function formulation to optimise, and (3) how policies are initialised at each stage. Effectively configuring these elements can significantly reduce overoptimisation while improving true reward.

merely postpones the inevitable exploitation of the reward model akin to persistent adversarial policies [12], or perpetuates a recurring cycle of overoptimisation in different forms [35].

In this work, we present the first systematic investigation into reward model overoptimisation in iterated RLHF. We identify three pivotal design choices, highlighted in Figure 1, that critically influence the success or failure of the process: *preference data management* (i.e., whether to aggregate or isolate preference data across iterations), *reward function formulation* (i.e., the choice of reward signal to optimize in subsequent training rounds), and *policy initialisation* (i.e., the strategy for initialising the policy at the start of each fine-tuning cycle).

Our key contributions can be summarised as:

- We present the first formal investigation of overoptimisation dynamics across multiple RLHF iterations, relaxing assumptions made in previous work.
- We discuss a systematic evaluation of key design choices with quantitative evidence of their impact on performance and overoptimisation.
- We provide guidelines for practitioners implementing iterated RLHF, including specific recommendations for preference data management, reward function selection, and policy initialisation strategies.

Using a gold-standard reward model to simulate human labellers [5, 11] on the AlpacaFarm dataset [38] and working exclusively with open-source models, our experiments yield several key insights: Reward models become increasingly robust across iterations, leading to higher gold reward scores (Figure 2). Performance gains diminish after three iterations for most methods. Concatenating preference data across iterations dramatically outperforms other approaches. Small but persistent overoptimisation remains after four iterations regardless of design choices.

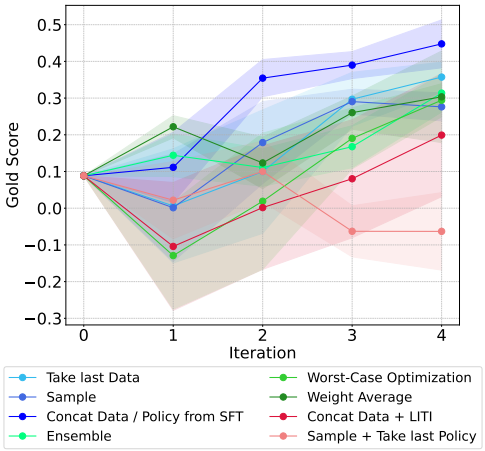


Figure 2: Our design choices for preference data management, reward function formulation, and policy initialisation significantly impact performance across iterations. Concatenating preference data yields dramatic improvements in true reward scores, while equal-sampling with last-policy initialization performs worst (see Section 5). These findings offer concrete guidance for practitioners implementing robust RLHF systems.

Our results demonstrate that while iterated RLHF significantly improves reward model robustness, it does not fully eliminate overoptimisation. This underscores the need for continued research into more robust alignment methods that can withstand sophisticated specification gaming [16] by increasingly capable language models.

2 Related work

RLHF is the standard for aligning large language models to human preference data. The iterated approach has been first used by the authors in [2] to fix robustness and calibration issues, attributed to lack of data in the high score regime and has since gained in popularity [32, 41, 44, 1, 7, 45]. Besides training on newly collected preferences, an iterated scheme to train reward models from synthetically generated preference data has been proposed in [39] and shown to improve performance on the reward model benchmark RewardBench [18], but the authors focus on iterated training of an evaluator and do not study overoptimisation nor the design choices we consider. In the context of Direct Preference Optimisation (DPO) [31] offline, online and hybrid approaches repeatedly collecting new preference data have been investigated mostly in terms of sample efficiency [41, 6, 26, 23]. More broadly iterated methods have been investigated for machine teaching [19] and to resolve feedback loops caused by model deployment in supervised learning [29] and also performative RL [22].

Overoptimisation is a common issue in RL, and evidence of this has been frequently reported in the RLHF literature [47, 37, 11, 35]. A promising method to mitigate overoptimisation is using reward model ensembles combined with conservative optimisation as proposed by [5]. Several works further explore reward model ensembles in RLHF [9, 21]. Notably, in [33] the authors introduce weight averaged reward models (WARM) alleviating the inference cost of multiple reward models during training. Following [5] and [11] in tackling reward model overoptimisation, several works propose alternative approaches including reward model distillation [10], hidden state regularisation [43], and more [42, 24, 20, 13]. One commonly reported mode of overoptimisation is length bias [35, 28], which can be tackled by disentangling reward signals related to response length from content quality [4]. To the best of our knowledge, the literature lacks a systematic investigation into overoptimisation in iterated Reinforcement Learning from Human Feedback (RLHF). Such an investigation is not only necessary but also fundamentally important for a deeper understanding and meaningful improvement of fine-tuning methods based on this technique.

3 Iterated Reinforcement Learning from Human Feedback

In this section, we first outline the process of a single iteration of RLHF and then extend it to the iterated framework. The RLH pipeline consists of the following three steps: 1. Collection of a preference dataset; 2. Reward model training; 3. Policy optimisation on the reward model. Though not an integral part of the RLHF pipeline, it is common in practice for step 1 to be preceded by supervised fine-tuning on labelled examples.

3.1 Single-iteration RLHF

Preference data collection. We begin with a policy π^{sft}, obtained by supervised fine-tuning (in practical terms, a policy checkpoint), and use it for collecting preference data. More specifically, the preference dataset $\mathcal{D}$ consists of tuples $\{x_i, y_{i,0}, y_{i,1}, p_i\}$ for $i = 1, ..., N$, where $x_i \in \mathcal{X}$ is a prompt or instruction, $y_{i,j} \sim \pi^{sft}(\cdot|x_i)$ with $j = 0, 1$ are two synthesised responses from the supervised fine-tuned policy checkpoint π^{sft}, and p_i is the preference label indicating whether $y_{i,0}$ is preferred over $y_{i,1}$. Similarly to [5, 11], the preference labels p_i are simulated with a gold reward model denoted by $R^\star$, which is significantly larger in terms of parameter size than the proxy reward models, as an approximation for human labellers that would be used in the real RLHF process instead.

Reward model training. The proxy reward model R_ϕ is initialised from model checkpoint R^{init}, with a randomly initialised prediction head, and subsequently trained by minimizing the cross-entropy loss on the preference dataset $\mathcal{D}$. It is standard to use the Bradley-Terry model [3], under which the probability of preferring the answer y_0 over y_1 given prompt x is given by

$$\mathbb{P}\left(y_0 \succ y_1|x\right) = \frac{1}{1 + \exp\left(R\left(x, y_1\right) - R\left(x, y_0\right)\right)}. \tag{1}$$

Policy optimisation. Having trained the proxy reward model R_ϕ, the policy π_θ is initialised from π^{sft} and then fine-tuned to optimise R_ϕ. This is commonly achieved with the proximal policy

optimization (PPO) algorithm [34]. In order to prevent overoptimisation of the proxy reward model and regularise π_θ to not diverge too drastically from its initialisation, a Kullback-Leibler divergence (KL) penalty is used. This yields the overall reward maximised as

$$R^{\text{PPO}}(x, y) = R_\phi(x, y) - \beta \log \left[\frac{\pi_\theta(y \mid x)}{\pi^{sft}(y \mid x)} \right], \tag{2}$$

where β controls the strength of the KL penalty (unless specified otherwise we set $\beta = 1 \times 10^{-4}$). This procedure, which only collects preferences once in the entire pipeline, has an important disadvantage. Reward models have been found to be poorly calibrated in the higher reward regime [2] and trained policies overoptimise the proxy reward model leading to unstable fine-tuned policies [30, 11, 47]. Notably, policy optimization induces a divergence between the distributions $\pi_\theta(y|x)$ and $\pi^{\text{sft}}(y|x)$. This causes the optimised policy to generate outputs that are different from those seen in the training data $\mathcal{D}$. As a result, the reward model R_ϕ, which was trained on the data $\mathcal{D}$, is now being evaluated on data that it has not seen before (out of distribution).

3.2 Iterated RLHF and design choices

The problem of the divergence between the distributions $\pi_\theta(y|x)$ and $\pi^{sft}(y|x)$ is the one addressed by iterated RLHF. In this process, multiple iterations of steps 1-3 of the RLHF pipeline (namely, collection of preference data, reward model training, and policy optimisation)are repeated as shown in Figure 1. Just as in the single-iteration setting, we start from the checkpoint π^{sft} and initialise the reward model from R^{init} with a randomly initialised prediction head. However, there are multiple design choices to be made when choosing how exactly to perform iterated RLHF training. We now describe the process in more detail, highlighting the design choices throughout. Please refer to Algorithm 1 for a schematic of the entire process. For simplicity of notation, we omit explicit references to the policy and reward model parameters θ and ϕ, using the subscript k to index iterations instead. During the kth iteration of RLHF, we use the policy from the previous one, denoted by π_{k-1} to synthesise pairs of responses for the new preference data denoted by $\tilde{\mathcal{D}}_k$.

Indeed, using all policies is unnecessary as it equates to reapplying preference data, but at a higher cost. This new data enables the training of a proxy reward model for which the current policy's output is in-distribution, potentially mitigating the issue of overoptimisation. Taking into account previous iterations, we now have access to the list of preference data $[\tilde{\mathcal{D}}_1, ..., \tilde{\mathcal{D}}_k]$. Here we face the first design choice:

How do we combine the list of k preference datasets into a single training dataset $\mathcal{D}_k$?

Combining preference data. Given a list of k preference datasets, the responses in each of which have been generated by different policies $\pi_1, ..., \pi_{k-1}$, we identify three possible options to consolidate them into a single training dataset. The first option (Figure 3.a) is to simply set $\mathcal{D}_k = \tilde{\mathcal{D}}_k$, only training the reward model on the preference data collected in the current iteration. The second option at the other extreme (i.e., no inter-iteration transfer) is to concatenate all datasets (Figure 3.b). Reusing all the data at each iteration is expected to result in decreased overoptimisation and better approximation with respect to the true reward function. However, this comes with a reward model training computational cost that scales linearly with the number of iterations. Finally, balancing training time and information transfer, we keep the size of the reward model training data constant across iterations by sampling a subsets $\tilde{\mathcal{D}}_i$ for $i = 1, ..., k$ and concatenating the subsets to form $\mathcal{D}_k$ (Figure 3.c). Once the training data $\mathcal{D}_k$ has been obtained, the proxy reward model $\tilde{R}_k$ can be trained on it. $\tilde{R}_k$ is initialised from the same base model in all iterations. Having trained the reward model, we now arrive at the second critical design choice:

How do we transfer information from the list of all previously trained proxy reward models $[\tilde{R}_1, ..., \tilde{R}_k]$ into a single reward function R_k that can be optimised by the policy?

Combining reward models. The reward model is the crucial piece in obtaining generalisable and consistent policies in RLHF, and it is even more important over multiple iterations as effects compound. Given the list $[\tilde{R}_1, ..., \tilde{R}_k]$ containing the k proxy reward models leading up to the current iteration the task is to obtain a robust reward function to be optimised. We note that this design choice can be considered in parallel to the combination of preference data, as both target the same outcome of transferring information from previous iterations to the reward function.

Algorithm 1 Iterated RLHF (design choices highlighted)

Inputs: Prompt dataset $X = \{x_i\}_{i=1}^N$, π^{sft}, R^{init}, $R^\star$, # of iterations n_{iter}
$\pi_0 \leftarrow \pi^{sft}$
for $k = 1$ **to** n_{iter} **do**
  $y_{i,0}, y_{i,1} \sim \pi_{k-1}(x_i)\ \forall x_i \in X$
  $p_i \leftarrow R^\star(x_i, y_{i,0}, y_{i,1})\ \forall x_i \in D$
  $\tilde{\mathcal{D}}_k \leftarrow \{x_i, y_{i,0}, y_{i,1}, p_i\}_{i=1}^N$
  $\mathcal{D}_k \leftarrow \text{CombineData}([\tilde{\mathcal{D}}_1, ..., \tilde{\mathcal{D}}_k])$
  $\tilde{R}_k \leftarrow \text{TrainRM}(R^{init}, \mathcal{D}_k)$
  $R_k \leftarrow \text{CombineRM}([\tilde{R}_1, ..., \tilde{R}_k])$
  $\pi_k^{init} \leftarrow \text{CombineΠ}([\pi_0, ..., \pi_{k-1}])$
  $\pi_k \leftarrow \text{TrainRL}(\pi_k^{\text{init}}, R_k)$
end for
return π_k

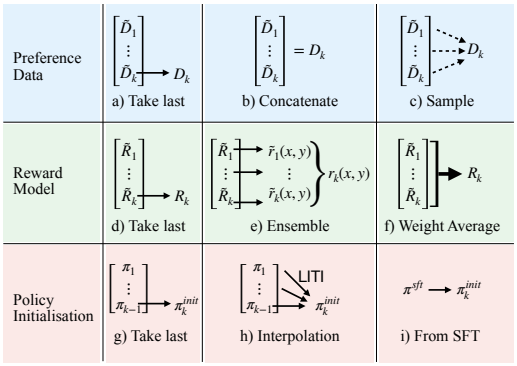


Figure 3: Design choices for Iterated RLHF (Algorithm 1). Options include how to combine preference data (latest only, concat, or sample), transfer reward models (last, ensemble, or weight averaged), and initialize policies (last, interpolate, or from SFT). These choices determine how learning signals are propagated through each iteration.

To achieve this task we investigate three types of solutions. The first only uses the most recently trained proxy reward model setting $R_k = \tilde{R}_k$ (Figure 3.d), hence there is no utilisation of previously trained reward models. In contrast, the second option ensembles all previously trained proxy RMs taking the mean of the individual rewards (Figure 3.e) [5]. Since reward model ensembles showed limited improvements in [5] we also evaluate worst-case optimisation (WCO), i.e. optimising the minimum $R_k(x, y) = \underset{i=1,...,k}{min} \tilde{R}_i(x, y)$. This option comes with the disadvantage of requiring inference on k reward models in parallel. To address the computational cost, we also consider weight averaged reward models (see Figure 3.f) by performing task arithmetic [15]. More formally, given a sequence of reward models $\tilde{R}_1, ..., \tilde{R}_k$, which are parameterised by $\tilde{\phi}_1, ..., \tilde{\phi}_k$, respectively, we obtain the proxy reward function R_k parameterised by ϕ_k as follows: The ensemble uses $R_k(x, y) = \frac{\sum_{i=1}^k \tilde{R}_i(x,y)}{k}$ and to obtain the weight averaged reward model we set $\phi_k = \frac{\sum_{i=1}^k \tilde{\phi}_i}{k}$. Having obtained the reward function, the next and final step of each iteration is to optimise it, which leads us to the third and final design choice:

Given π^{sft} and the fine-tuned policies $\pi_1, ..., \pi_{k-1}$, how can we choose π_k^{init} to balance efficiency and robustness against overoptimisation?

Policy initialisation. The final design choice concerns the initialisation of the policy, i.e., how π_k^{init} is chosen. The authors of [2] initialise the policy from π^{sft} at every iteration, not taking into consideration previously performed computation. We call this initialisation *From SFT* shown in Figure 3.i. As alternative, we use linear interpolation towards initialisation (*LITI*) [32], which was inspired by WiSE-FT proposed by [40]. With *LITI*, shown in Figure3.h, we set $\pi_k^{init} = (1 - \eta)\pi_{k-1}^{init} + \eta\pi_{k-1}$, where η is a hyperparameter balances the optimisation of R_{k-1}. Taking $\eta = 1$ corresponds to initialising the current policy from the previously fine-tuned one, setting $\pi_k^{init} = \pi_{k-1}$. Since continuing fine-tuning of the most recent policy fully relies on the previous iterations, it may suffer from entropy collapse leading to no optimisation in later iterations. Continuing with the fine-tuned policy carries risks if undesirable behaviour learned in previous iterations cannot be unlearned. Note, when performing *LITI*, the policy is regularised with the KL between the policy and its initialisation π_k^{init}.

4 Evaluating overoptimisation in Iterated RLHF

In Section 3 we formalised the process of iterated RLHF and highlighted the critical design choices. In this section, we detail our evaluation setup, emphasizing the quantification of overoptimisation and examining how its progression over iterations is influenced by different design choices.

4.1 Training setup

Our evaluation setup follows extensive prior works that study overoptimisation in the single iteration RLHF in a controlled and simulated manner [5, 11]. Similarly to [5] we use instructions from the *AlpacaFarm* dataset [8] for reward model training and policy optimisation. The preference data $\tilde{\mathcal{D}}_k$ collected at each iteration contains preferences for a subset of 1000 instructions in the preference split of AlpacaFarm. Preference labels p_i are simulated with the 7 billion parameter `Reward-Model-AlpacaFarm-Human` [8], which is also used by [5]. It is worth noting again the significant difference in parameter sizes between the proxy reward models and the gold reward model, justifying the use of the gold reward model as a proxy for human labellers. Similarly to [5], to obtain π^{sft}, we performed supervised fine-tuning on the `pythia-410m` model[1] on the AlpacaFarm SFT split. We chose `pythia-410m` as it achieves an appropriate balance between computational cost and experimental rigour for our investigation. The authors in [11] also found that policy size did not affect the shape of the overoptimisation curve in their setting, further justifying this choice of policy. We initialise proxy reward models $\tilde{R}_k$ from the huggingface checkpoint `tlc4418/pythia_70m_sft` provided by [5][2], as well as the larger `pythia-160m`, with a randomly initialised prediction head [5]. We train reward models for 5 epochs with a learning rate of 1×10^{-5} [5]. For policy optimisation, we perform 6000 steps of PPO on the unlabelled split of AlpacaFarm. The learning rate is set to 1×10^{-6} and a constant KL penalty of 1×10^{-4} is used. The full specifications of the hyperparameters for reward model training and policy optimisation, and the prompt format are given in Appendix C.

We perform a total of 4 iterations per method and report the results of the final iteration in comparison to the initial one. All results presented in our performance evaluation are reported for 8 random seeds, except for policy initialisation *From SFT* with the *Take last* configuration for both preference data and reward model, for which we only obtained 4 random seeds due to compute constraints. We note that this is still above the commonly reported 3 random seeds. To aggregate seeds in both gold score and KL we collect all seeds per iteration, bucket data points by KL. We then plot the mean and standard deviation of the gold rewards per bucket against the KL.

4.2 Measuring reward model overoptimisation with the Maximum Mean Discrepancy

The standard methodology for investigating reward model overoptimisation is to compare the achieved mean reward on the proxy and gold reward functions on a hold-out set [5, 25, 11]. However, this analysis does not capture discrepancies in the long-tail, i.e., in the high-reward regime, that have a larger impact on the policy optimisation. In this work we propose to compare two reward models based on their distributions of rewards. To this end, we evaluate the policy and reward models on the 2000 unseen instructions contained in the validation split of AlpacaFarm at every 300 steps during policy optimisation. Our approach to measuring differences between reward functions consists of two steps, the first of which is a standardisation that ensures reward functions that lead to the same ordering of policies when optimised are treated as equal (see Apendix B). In the second step, we use the maximum mean discrepancy (MMD) [14] to measure the discrepancy between the two reward functions. In particular, we utilise this method to compare the proxy reward models trained at each iteration with the gold-reward model $R^\star$. For full details and a justification of the validity of this method we refer the reader to Appendix A.

5 Experimental results

When comparing different methods, we primarily focus on their performance in the final iteration, as this iteration consistently outperforms previous ones for all algorithms. Additionally, it demonstrates the reward-KL curves produced by each method. We also compare the performance of methods across multiple iterations, to see how the KL-reward curves change through the iterations.

5.1 Iterated RLHF can close the gap between proxy and gold reward functions

Before investigating the differences between the design choices, we focus on the progression of reward model robustness across iterations more generally. In Figure 4, we show how performing multiple iterations of RLHF, concatenating all preference data to train the reward model, and re-initialising the policy from π^{sft} at each iteration decreases the gap between the gold reward function

[1] https://huggingface.co/EleutherAI/pythia-410m

[2] https://huggingface.co/tlc4418/pythia_70m_sft

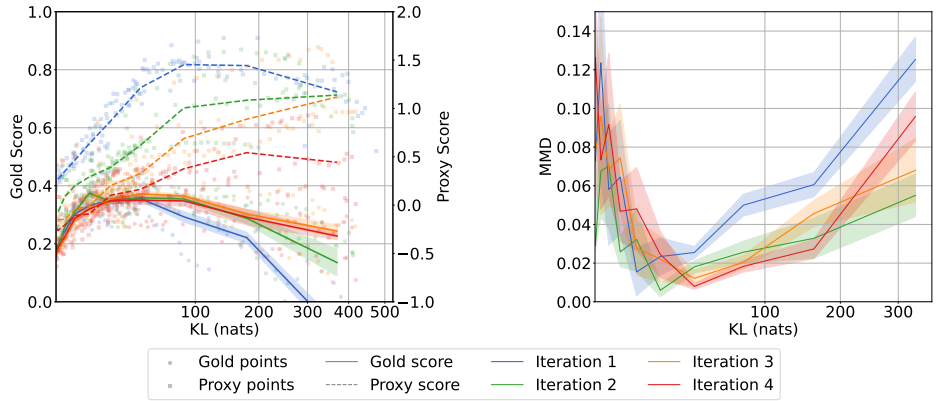


Figure 4: Progression of proxy and gold reward alignment across RLHF iterations with policy reinitialization from π^{sft} and concatenated preference data. Mean gold and proxy scores (with standard error across seeds) indicate a narrowing gap across iterations, demonstrating improved robustness of the proxy reward model and alignment with the gold reward. However, diminishing returns are visible in later iterations. Maximum Mean Discrepancy (MMD) between proxy and gold reward distributions on the hold-out set shows initial convergence, but reveals a consistent rise at higher KL values, especially in early iterations, highlighting persistent distributional shifts and misalignments not captured by mean scores alone.

and the proxy. As iterations progress, the proxy reward model becomes more robust and increasingly aligned with the gold reward model on the distribution observed during policy optimisation.

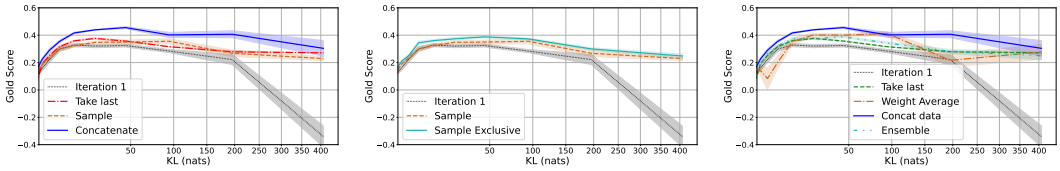
Furthermore, the KL-reward Pareto front advances with each iteration, although improvements diminish as the distance between proxy and gold reward curves shrinks in later iterations. Despite these diminishing returns, there remains scope to better align gold and proxy reward functions. Comparing reward distributions across iterations further reveals that, after the policy closely approximates the output distribution in $\mathcal{D}_k$, the MMD increases again in the high-KL regime for all iterations, especially rapidly in the initial iteration (see Figure 4). Additionally, the rate at which the proxy-gold reward gap closes varies considerably among methods (see Appendix E.1), highlighting the importance of investigating design choices described in Section 3.

5.2 Combining preference data

Scaling reward model training data is most effective. We first focus on comparing methods for combining preference datasets. To isolate the effects of varying the combination strategy, we fix the policy initialization to *From SFT* (Figure 3.i) and reward models are combined using the *Take last* (Figure 3.d) approach. As shown in Figure 5a, all methods demonstrate significant improvements over a single iteration, particularly in preventing performance collapse at higher KL divergences.

The *Concatenate* (Figure 3.b) strategy achieves consistently higher Gold Scores, especially in the KL range of 50-200 nats, where it maintains a performance advantage over other methods. While *Take last* (Figure 3.a) and *Sample* (Figure 3.c) approaches show similar trends and substantial improvements over iteration 1, they do not quite match the performance of full data concatenation. This result is coherent with the finding that increasing training dataset size reduces reward model overoptimisation [11], explaining why the sampling strategy is outperformed by concatenating all datasets. A critical observation is that beyond $\mathrm{KL} \approx 200$, the baseline iteration 1 experiences severe performance degradation due to overoptimisation, dropping to negative gold scores. In contrast, all iterative approaches maintain positive performance even at high KL values, demonstrating their effectiveness in mitigating overoptimisation. This ranking of methods is not only observed in the final iteration, but is already exhibited as early as the second iteration as shown in Figure 2 and in Appendix E.2.

Ensuring full coverage of the prompts when sampling matters less. While the sampling strategy slightly outperformed taking only the newest preference dataset, it did not achieve the same level of performance as concatenating all data. Here we take a closer look at the sampling strategy. In Figure 5b standard sampling with potential prompt repetition (*Sample*) and sampling where each prompt appears exactly once *Sample Exclusive*. While *Sample Exclusive* shows a slight advantage in the mid-KL regime (100-250 nats), the differences are modest, suggesting that prompt repetition



(a) Combining Preference Data (b) Ensuring Prompt Coverage (c) Combining Reward Models

Figure 5: Iterated RLHF benefits most from scaling reward model training data. (a) Concatenating all preference data across iterations consistently outperforms sampling or using only recent data, particularly at mid KL values (50–200 nats), and best mitigates overoptimisation. (b) Sampling with or without prompt repetition yields similar results, suggesting prompt coverage has limited impact. (c) Parameter-space reward model combinations like ensembling and weight averaging offer efficiency gains but do not surpass the simpler *Take last* approach. Concatenation remains most effective, motivating efficient alternatives.

plays a minor role in model performance and mitigating overoptimisation. These observations also hold throughout the earlier iterations (see Appendix E.2), highlighting that while data combination strategies are effective at preventing overoptimisation, the computational cost of maintaining and training on growing datasets remains a necessity as more efficient methods are unable to achieve the same performance as *Concatenate*. This motivates our exploration of alternative approaches focusing on the combination of reward models in the parameter space, which could potentially offer similar benefits with reduced computational overhead.

5.3 Combining reward models

No free lunch by merging reward models. The method of concatenating all preference data, previously shown to be most effective, represents the level of performance we aim to achieve. As shown in Figure 5c, all approaches demonstrate similar patterns of improvement in the early KL regions (up to ≈ 50 nats), reaching comparable performance levels. The *Weight Average* (Figure 3.f) and *Ensemble* (Figure 3.e) methods show consistent performance throughout, while maintaining computational efficiency compared to the data concatenation approach. However, using a reward model ensemble scales inference time and memory requirements during policy optimisation. We observe the mean objective does not yield significant performance gains over the *Take Last* approach relying on a single reward model (consistent with the findings for the single iteration setting in [5]). While weight averaged reward models have been shown to outperform ensembles [33], we do not observe significant differences except on the mid-KL regime. In contrast to [5, 33], we ensemble and weight average models that have been trained on data with significantly different joint distribution over pairs (x, y). Regardless, both methods still provide significant improvements when comparing the fourth and first iterations. The various reward model combination methods in RLHF perform similarly, suggesting computational efficiency should drive selection.

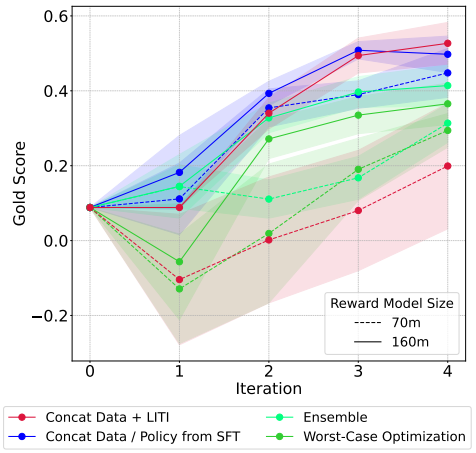
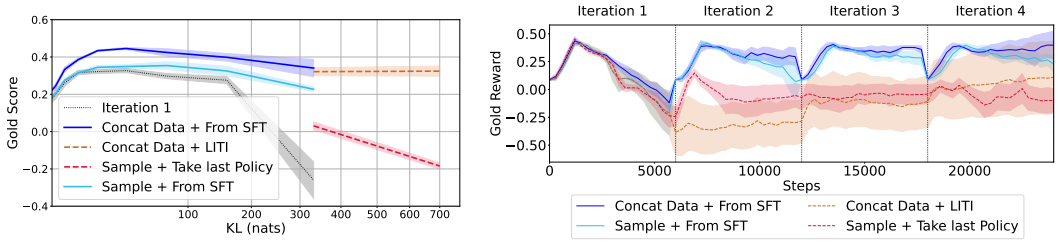


Figure 6: Impact of reward model size and design choices on gold score across iterations. Larger reward models (160M, solid lines) consistently outperform smaller ones (70M, dashed), with the biggest gains seen in *Ensemble* and *Worst-Case Optimisation*—suggesting that these strategies benefit most from increased reward model capacity. While *From SFT* remains the most stable across scales, *LITI* shows steady improvement, especially with the larger model.

Larger reward models benefit more from combining reward models. We now investigate how scaling the reward model size affects performance in iterative RLHF. While concatenating all preference data with policy initialisation from the SFT checkpoint remains the most robust approach, we observe that alternative reward model strategies benefit significantly from increased reward model capacity. As shown in Figure 6, performance differences between the 70M and 160M reward models



(a) Fourth Iteration

(b) Across Iterations

Figure 7: Effect of policy initialisation on overoptimisation and recovery across iterations. At iteration 4, initialising *From SFT* is most robust, avoiding overoptimisation through repeated resets and data aggregation. In contrast, *LITI* and *Take last* start from high KL due to cumulative divergence. When Plotting reward against timesteps and across iterations we observe that overoptimised policies are hard to recover: *Take last* degrades further, while *LITI* shows steady improvement, though neither fully closes the gap to *From SFT*.

are most pronounced for *Ensemble* and *Worst-Case Optimisation*, with both methods substantially improving at the larger scale and approaching the performance of the data concatenation baseline by the fourth iteration. This suggests that while reward model combination methods did not match the effectiveness of preference data concatenation at smaller scales, their potential is unlocked with more expressive reward models. These results highlight that design choices affecting reward model size not only influence individual model accuracy but can significantly enhance the utility of design choices combining reward models in iterated RLHF settings. We next examine if policy initialization strategies can complement reward modelling and preference aggregation to prevent overoptimisation.

5.4 Policy initialisation

Initialising from SFT is the most robust. Finally, comparing the policy initialisation methods we observe that no method improves on the KL-reward Pareto front achieved by concatenating all preference data and initialising the policy from the SFT checkpoint (Figure 7a). Sampling the preference data is similarly robust, highlighting that initialising with *From SFT* results in generally reduced overoptimisation. Note, *LITI* and *Take last* start from significantly larger KL due the compounding of KL through repeated initialisation increasingly further away from π^{sft} in the KL space. Resetting the policy at each iteration combined with the aggregation of preference data results in consistently less overoptimisation and more performant policies. Although, initialisation with π^{sft} limits the flexibility and potential gains that could be realised by continued optimisation.

Overoptimised policies are hard to recover from. While *From SFT* is reset at the end of each iteration, *LITI* and *Take last* have to recover form the initial overoptimisation, as shown in Figure 7b. The behaviour in earlier iterations reveals the consistent performance improvements attained with *LITI*. On the other hand, *Take last* is unable to recover after overoptimising again in the second iteration, despite the counterpart, sampling preference data but initialising *From SFT*, improving with each iteration. Due to entropy decreasing caused by the prolonged optimisation when using the *Take last* initialisation, the mean gold reward does not exceed zero in the third and fourth iterations. Despite *LITI* improving on average across multiple seeds, we observe that linear interpolation is also unable to recover strongly overoptimised seeds (see Appendix E.4). Thus, while *From SFT* is most robust, it is also limited by the repeated initialisation from π^{sft}.

Policy interpolation works better with larger reward models. We hypothesise that *LITI* could achieve similar or higher gold scores than *From SFT* after more iterations. Supporting this, our experiments with a larger reward model show that *LITI* benefits substantially from increased reward model capacity (see Figure 6). This improvement likely stems both from better-calibrated gradients that support recovery, and from the fact that larger reward models tend to overoptimise less aggressively [11], resulting in safer intermediate policies and more stable interpolation paths. These findings highlight the importance of early stopping and reward model design when using policy initialisation methods other than *From SFT*, and suggest that *LITI* may become increasingly competitive as reward model expressiveness scales.

6 Limitations

Our study focuses on controlled settings, using modestly sized policy models (Pythia-410M) and reward models (70M, 160M) on the AlpacaFarm benchmark with a static "gold" reward model to simulate human feedback. However, this setup, consistent with prior work [5, 33, 46] (also in terms of model size), enables systematic investigation of iterative RLHF while ensuring results remain interpretable and comparable. Moreover, scaling laws suggest policy size is not the main driver of overoptimisation and that scale effects are smooth [11], indicating that our findings and design choices are likely to transfer to larger models, even if the degree or speed of overoptimisation differs. Using a single dataset (AlpacaFarm) enabled controlled ablations but may not capture the diversity of real-world tasks. We also note that a static "gold" reward model, abstracts away the noisy and evolving nature of human preferences. However, this is standard practice in the field [5, 11] to ensure reproducibility and mitigate cost of human labelling. We ran four iterations, enough to observe plateaus and overoptimisation trends, but further scaling was prohibitive given the computational resources of our institution. Nonetheless, our work offers key insights and guidance for designing more robust iterative RLHF pipelines and lays groundwork for future research on larger scales and real-world settings.

7 Conclusion

In this work we have presented the first investigation of reward model overoptimisation in iterated RLHF. Through simulations with a gold-standard reward model and analysing distributional discrepancies, we have demonstrated that overoptimisation diminishes across iterations as reward models better approximate the ground truth. However, improvements begin to plateau after three iterations. While completely eliminating overoptimisation remains unattainable, we have identified base policy initialisation as the most robust approach, despite its reduced optimisation flexibility. Our analysis provides practical guidelines for implementing iterated RLHF and illuminates fundamental trade-offs in iterative preference learning, establishing a foundation for future research in reliable RLHF systems.

References

[1] Leonard Adolphs, Tianyu Gao, Jing Xu, Kurt Shuster, Sainbayar Sukhbaatar, and Jason Weston. The CRINGE loss: Learning what language not to model. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL'23)*, 2023.

[2] Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, Nicholas Joseph, Saurav Kadavath, Jackson Kernion, Tom Conerly, Sheer El-Showk, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, Tristan Hume, Scott Johnston, Shauna Kravec, Liane Lovitt, Neel Nanda, Catherine Olsson, Dario Amodei, Tom Brown, Jack Clark, Sam McCandlish, Chris Olah, Ben Mann, and Jared Kaplan. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*, 2022.

[3] Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

[4] Lichang Chen, Chen Zhu, Jiuhai Chen, Davit Soselia, Tianyi Zhou, Tom Goldstein, Heng Huang, Mohammad Shoeybi, and Bryan Catanzaro. ODIN: Disentangled reward mitigates hacking in RLHF. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, 2024.

[5] Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. Reward model ensembles help mitigate overoptimization. In *Proceedings of the 12th International Conference on Learning Representations (ICLR'24)*, 2024.

[6] Nirjhar Das, Souradip Chakraborty, Aldo Pacchiano, and Sayak Ray Chowdhury. Provably Sample Efficient RLHF via Active Preference Optimization. *arXiv preprint arXiv:2402.10500*, 2024.

[7] 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. *Transactions on Machine Learning Research*, September 2024.

[8] Yann Dubois, Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. AlpacaFarm: A Simulation Framework for Methods that Learn from Human Feedback. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS'23)*, 2023.

[9] Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking. In *Proceedings of the 1st Conference on Language Modeling (COLM'24)*, 2024.

[10] Adam Fisch, Jacob Eisenstein, Vicky Zayats, Alekh Agarwal, Ahmad Beirami, Chirag Nagpal, Pete Shaw, and Jonathan Berant. Robust preference optimization through reward model distillation. *arXiv preprint arXiv:2405.19316*, 2024.

[11] Leo Gao, John Schulman, and Jacob Hilton. Scaling Laws for Reward Model Overoptimization. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, 2023.

[12] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial Policies: Attacking Deep Reinforcement Learning. In *Proceedings of the 8th International Conference on Learning Representations (ICLR'20)*, 2020.

[13] Alexey Gorbatovski, Boris Shaposhnikov, Alexey Malakhov, Nikita Surnachev, Yaroslav Aksenov, Ian Maksimov, Nikita Balagansky, and Daniil Gavrilov. Learn your reference model for real good alignment. *arXiv preprint arXiv:2404.09656*, 2024.

[14] Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012.

[15] Gabriel Ilharco, Marco Tulio Ribeiro, Mitchell Wortsman, Suchin Gururangan, Ludwig Schmidt, Hannaneh Hajishirzi, and Ali Farhadi. Editing models with task arithmetic. In *Proceedings of the 11th International Conference on Learning Representations (ICLR'23)*, 2023.

[16] Victoria Krakovna, Jonathan Uesato, Vladimir Mikulik, Matthew Rahtz, Tom Everitt, Ramana Kumar, Zac Kenton, Jan Leike, and Shane Legg. Specification gaming: the flip side of ai ingenuity. `https://deepmind.google/discover/blog/specification-gaming-the-flip-side-of-ai-ingenuity/`, 2020. Accessed: 2025-05-02.

[17] Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi-Rui Tam, Keith Stevens, Abdullah Barhoum, Nguyen Minh Duc, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri, David Glushkov, Arnav Dantuluri, Andrew Maguire, Christoph Schuhmann, Huu Nguyen, and Alexander Mattick. OpenAssistant Conversations – Democratizing Large Language Model Alignment. In *NeurIPS 2023 Datasets and Benchmarks*, 2023.

[18] Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating Reward Models for Language Modeling. *arXiv preprint arXiv:2403.13787*, 2024.

[19] Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B. Smith, James M. Rehg, and Le Song. Iterative machine teaching. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 2149–2158, 06–11 Aug 2017.

[20] Zhihan Liu, Miao Lu, Shenao Zhang, Boyi Liu, Hongyi Guo, Yingxiang Yang, Jose Blanchet, and Zhaoran Wang. Provably Mitigating Overoptimization in RLHF: Your SFT Loss is Implicitly an Adversarial Regularizer. In *Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS'24)*, 2024.

[21] Xingzhou Lou, Dong Yan, Wei Shen, Yuzi Yan, Jian Xie, and Junge Zhang. Uncertainty-aware reward model: Teaching reward models to know what is unknown. *arXiv preprint arXiv:2410.00847*, 2024.

[22] Debmalya Mandal, Stelios Triantafyllou, and Goran Radanovic. Performative reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

[23] Viraj Mehta, Vikramjeet Das, Ojash Neopane, Yijia Dai, Ilija Bogunovic, Jeff Schneider, and Willie Neiswanger. Sample Efficient Reinforcement Learning from Human Feedback via Active Exploration. *arXiv preprint arXiv:2312.00267*, 2023.

[24] Yuchun Miao, Sen Zhang, Liang Ding, Rong Bao, Lefei Zhang, and Dacheng Tao. InfoRM: Mitigating Reward Hacking in RLHF via Information-Theoretic Reward Modeling. In *Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS'24)*, 2024.

[25] Ted Moskovitz, Aaditya K. Singh, DJ Strouse, Tuomas Sandholm, Ruslan Salakhutdinov, Anca D. Dragan, and Stephen McAleer. Confronting Reward Model Overoptimization with Constrained RLHF. In *Proceedings of the 12th International Conference on Learning Representations (ICLR'24)*, 2024.

[26] William Muldrew, Peter Hayes, Mingtian Zhang, and David Barber. Active preference learning for large language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, 2024.

[27] Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

[28] Ryan Park, Rafael Rafailov, Stefano Ermon, and Chelsea Finn. Disentangling length from quality in direct preference optimization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4998–5017, 2024.

[29] Juan Perdomo, Tijana Zrnic, Celestine Mendler-Dünner, and Moritz Hardt. Performative prediction. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*, 2020.

[30] Rafael Rafailov, Yaswanth Chittepu, Ryan Park, Harshit Sikchi, Joey Hejna, W. Bradley Knox, Chelsea Finn, and Scott Niekum. Scaling laws for reward model overoptimization in direct alignment algorithms. In *Proceedings of the 38th Annual Conference on Neural Information Processsing Systems (NeurIPS'24)*, 2024.

[31] 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. In *Proceedings of the 37th Annual Conference on Neural Information Processing Systems (NeurIPS'23)*, 2023.

[32] Alexandre Ramé, Johan Ferret, Nino Vieillard, Robert Dadashi, Léonard Hussenot, Pierre-Louis Cedoz, Pier Giuseppe Sessa, Sertan Girgin, Arthur Douillard, and Olivier Bachem. WARP: On the Benefits of Weight Averaged Rewarded Policies. *arXiv preprint arXiv:2406.16768*, 2024.

[33] Alexandre Ramé, Nino Vieillard, Léonard Hussenot, Robert Dadashi, Geoffrey Cideron, Olivier Bachem, and Johan Ferret. WARM: On the Benefits of Weight Averaged Reward Models, 2024.

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

[35] Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A Long Way to Go: Investigating Length Correlations in RLHF. In *Proceedings of the 1st Conference on Language Modeling (COLM'24)*, 2024.

[36] Joar Skalse, Lucy Farnik, Sumeet Ramesh Motwani, Erik Jenner, Adam Gleave, and Alessandro Abate. STARC: A General Framework For Quantifying Differences Between Reward Functions. In *Proceedings of the 12th International Conference on Machine Learning (ICML'24)*, 2024.

[37] Nisan Stiennon, Long Ouyang, Jeff Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul Christiano. Learning to summarize from human feedback. In *Proceedings of the 34th Annual Conference on Neural Information Processing Systems (NeurIPS'20)*, 2020.

[38] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford Alpaca: An Instruction-following LLaMA model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

[39] Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. Self-taught evaluators. *arXiv preprint arXiv:2408.02666*, 2024.

[40] Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. Robust fine-tuning of zero-shot models. In *Proceedings of the 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR'22)*, 2022.

[41] 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 *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, 2024.

[42] Adam X. Yang, Maxime Robeyns, Thomas Coste, Jun Wang, Haitham Bou-Ammar, and Laurence Aitchison. Bayesian Reward Models for LLM Alignment. In *Proceedings of the ICLR 2024 Workshop on Secure and Trustworthy Large Language Models*, 2024.

[43] Rui Yang, Ruomeng Ding, Yong Lin, Huan Zhang, and Tong Zhang. Regularizing Hidden States Enables Learning Generalizable Reward Model for LLMs. In *Proceedings of the 38th Annual Conference on Neural Information Processing Systems (NeurIPS'24)*, 2024.

[44] Chenlu Ye, Wei Xiong, Yuheng Zhang, Nan Jiang, and Tong Zhang. Online Iterative Reinforcement Learning from Human Feedback with General Preference Model. In *Proceedings of the 41st Annual Conference on Neural Information Processing Systems (NeurIPS'24)*, 2024.

[45] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, 2024.

[46] Banghua Zhu, Michael I. Jordan, and Jiantao Jiao. Iterative Data Smoothing: Mitigating Reward Overfitting and Overoptimization in RLHF, 2024.

[47] Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2020.

A Reward model comparison with the Maximum Mean Discrepancy

Formally, our goal is to compare any two reward functions R_{ϕ_1} and R_{ϕ_2}. As the first step, we scale both reward functions to have mean zero and variance one. This ensures that reward functions, which differ only by an affine transformation, are treated as equal to one another after scaling. For details about this result, please refer to Appendix B. This is desirable since affine transformations do not affect the ordering over policies induced by the original and transformed reward functions when they are optimised [36].

As the second step, we compute the discrepancy between R_{ϕ_1} and R_{ϕ_2}. While we have reward functions in principle, during training, only samples of rewards from the true and proxy are observed. Given that prompts are identically and independently distributed $x_i \overset{i.i.d.}{\sim} \rho$ and $y_i \sim \pi_\theta(\cdot|x_i)$, we obtain that the observed rewards $r_i = R_\phi(x_i, y_i)$ are i.i.d samples (details in Appendix B). As a consequence, we can rely on the Maximum Mean Discrepancy (MMD) to measure the discrepancy between distributions of observed rewards from R_{ϕ_1} and R_{ϕ_2}. The MMD compares two distributions based on their distances in the feature space determined by the chosen kernel. It is known for its strong theoretical guarantees, and it is commonly used in the two sample testing literature [14]. We use the popular squared exponential kernel.

Given samples $\mathbf{r}_{\phi_1} := \{r_{\phi_1,1}, ..., r_{\phi_1,n}\}$ and $\mathbf{r}_{\phi_2} := \{r_{\phi_2,1}, ..., r_{\phi_1,n}\}$ an unbiased empirical estimate of the MMD is obtained by

$$\begin{aligned}\mathrm{MMD}_u^2[\mathbf{r}_{\phi_1}, \mathbf{r}_{\phi_2}] =& \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k\left(r_{\phi_1,i}, r_{\phi_1,j}\right) \\ &+ \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k\left(r_{\phi_2,i}, r_{\phi_2,j}\right) \\ &- \frac{2}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k\left(r_{\phi_1,i}, r_{\phi_2,j}\right).\end{aligned}$$

Note here that observations $\mathbf{r}_{\phi_1}$ and $\mathbf{r}_{\phi_2}$ cannot be assumed to be independent, since when comparing reward models across iterations and proxy reward models with the gold reward model, independence is not guaranteed.

This two-step procedure allows us to perform a detailed comparison of reward models going beyond the measurement of the mean gold reward.

B Proofs

Proposition B.1. *Let $R_{\phi_1}, R_{\phi_2} \in \mathcal{R}$ be two reward functions and suppose they differ by an affine transformation, i.e. $R_{\phi_2} = a \cdot R_{\phi_1} + b$ for some $a \in \mathbb{R}^+$ and $b \in \mathbb{R}$. Then $R_{\phi_1'} = R_{\phi_2'}$, where $R_{\phi_i'} = \frac{1}{\sigma_i} \cdot (R_{\phi_i} - \mu_i)$ with σ_i the standard deviation of R_{ϕ_i} and μ_i the mean.*

Proof of Proposition B.1. First note that $R_2 = a' \cdot R_1' + b'$, with $a' = a \cdot \sigma_1 \in \mathbb{R}+$ and $b' = b + a \cdot \mu_1$. We have that $\mu_2 = \mathbb{E}(R_2) = b'$ and $\sigma_2 = a'$. Hence

$$\begin{aligned} R_2' &= \frac{R_2 - \mu_2}{\sigma_2} && (3)\\ &= \frac{R_2 - b'}{a'} && (4)\\ &= \frac{a' R_1' + b' - b'}{a'} && (5)\\ &= R_1'. && (6)\end{aligned}$$

Proposition B.2. *Given i.i.d. observations $x_1, ..., x_n$ from random variable $x \sim \rho$, and a policy π_θ, we have that observations of rewards $r_1, ..., r_n$, where $r_i = R_\phi(x_i, y_i)$ for a deterministic reward function R_ϕ and $y_i \sim \pi_\theta(\cdot|x_i)$ for $i = 1, ..., n$, are i.i.d. observations of a random variable we denote by Z.*

Proof of Proposition B.2. Given that X_i are independent and identically distributed (i.i.d.) and that $Y_i \sim \pi(\cdot|X_i)$, we first show that Y_i are i.i.d..

To determine if Y_i are independent, we need to check if the joint distribution of any pair (Y_i, Y_j) for $i \neq j$ factorizes into the product of their marginal distributions.

Since X_i are i.i.d., we have:

$$P(X_i, X_j) = P(X_i)P(X_j) \text{ for } i \neq j.$$

Given $Y_i \sim \pi(\cdot \mid X_i)$, Y_i and Y_j are conditionally independent given X_i, X_j for $i \neq j$ and the conditional distribution of Y_i given X_i is independent of X_j for $j \neq i$, such that

$$P(Y_i, Y_j \mid X_i, X_j) = P(Y_i \mid X_i) P(Y_j \mid X_j)$$

Using the law of total probability, the joint distribution $P(Y_i, Y_j)$ can be written as

$$P(Y_i, Y_j) = \iint P(Y_i, Y_j \mid X_i, X_j) P(X_i, X_j) \, dX_i dX_j.$$

Substituting the factored form of the conditional and marginal distributions, we get

$$P(Y_i, Y_j) = \iint P(Y_i \mid X_i) P(Y_j \mid X_j) P(X_i) P(X_j) \, dX_i dX_j.$$

Since $P(X_i)$ and $P(X_j)$ are independent, this simplifies to

$$P(Y_i, Y_j) = \left(\int P(Y_i \mid X_i) P(X_i) \, dX_i \right) \times \left(\int P(Y_j \mid X_j) P(X_j) \, dX_j \right). \tag{7}$$

$$\tag{8}$$

This shows that

$$P(Y_i, Y_j) = P(Y_i) P(Y_j),$$

which means Y_i and Y_j are independent for $i \neq j$.

We now check if Y_i are identically distributed. Since $Y_i \sim \pi(\cdot \mid X_i)$ and X_i are i.i.d., the marginal distribution of Y_i is obtained by marginalizing over X_i, which yields

$$P(Y_i = y) = \int P(Y_i = y \mid X_i = x) P(X_i = x) \, dx.$$

Given that X_i are identically distributed, the distribution $P(X_i)$ is the same for all i. Therefore, the marginal distribution $P(Y_i)$ is the same for all i, indicating that Y_i are identically distributed.

Now, given $R_i = r(X_i, Y_i)$ where r is some deterministic function, we need to determine whether R_i are i.i.d., given that X_i are i.i.d. and $Y_i \sim \pi(\cdot \mid X_i)$.

Since X_i are i.i.d., X_i and X_j are independent for $i \neq j$. We have established that Y_i and Y_j are also independent for $i \neq j$. Because r is a deterministic function, R_i is fully determined by (X_i, Y_i). Specifically

$$R_i = r(X_i, Y_i) \text{ and } R_j = r(X_j, Y_j).$$

Given that (X_i, Y_i) and (X_j, Y_j) are independent pairs, it follows that R_i and R_j are also independent. This is because the independence of (X_i, Y_i) and (X_j, Y_j) implies that the mapping through r does not introduce any new dependency between R_i and R_j.

Next, we need to check if R_i are identically distributed. Since X_i are i.i.d. and $Y_i \sim p(\cdot \mid X_i)$, the distribution of (X_i, Y_i) is the same for all i. The function r is deterministic and applies the same transformation to each pair (X_i, Y_i). Therefore, the distribution of $R_i = r(X_i, Y_i)$ will be the same for all i. This concludes the proof.

C Additional experimental details

C.1 Hyperparameters

Our hyperparameter settings mostly align with those used by the authors in [5]. The parameters for supervised fin-tuning are given in Table 1, reward model training hyperparameters are specified in Table 2, PPO parameters are given in Table 3, and the hyperparameters for synthesis with a policy are provided in Table 4.

Table 1: SFT hyperparameters.

Parameter	Value
Learning rate	$8e-6$
Epochs	3
Batch size	4

Table 2: RM hyperparameters.

Parameter	Value
Learning rate	$1e-5$
Epochs	5
Batch size	32

Table 3: PPO hyperparameters.

Parameter	Value
Learning rate	$1e-6$
Cosine annealing scheduler	$1e-7$
PPO steps	6000
Batch size	32
Number of rollouts	256
Chunk size	32
Clipping range & value	0.2
GAE lambda	0.95

C.2 Dataset

We use the instructions and inputs contained in the popular alpacafarm dataset [8, 38]. The entire dataset contains $52,000$ samples split into "sft" (10k), "preference" (20k), "unlabled" (20k), and "val" (2k). We use the "val" split strictly only for validation. The instructions for the reward model training are sampled from the "preference" split and the instructions for PPO are sampled from the "unlabled" split.

C.3 Prompt format

We follow the prompt format used in [5, 17], which is that of the v2 format used in Open Assitant. It uses special tokens `<|prompter|>` and `<|assistant|>`, and is consistent with the `GPTNeoXTokenizer` class.

To generate answers the model is prompted with the concatenation of instruction and input (if present), where inputs begin on a new line. The entire prompt begins with the special token `<|prompter|>` and ends with the end-of-text token `<|endoftext|>` to indicate the end of the instruction followed by the `<|assistant|>` token to start generating the answer.

In the case of the reward model the prompt should additionally contain an answer to the instruction, which is appended to the initial prompt and again ended with the `<|endoftext|>` token.

Examples for both generation and reward modelling are given in Table 5.

C.4 Computational setup and cost

All experiments were run on a single Nvidia A100. Running the full pipeline consisting of all 3 RLHF steps for 4 iterations takes approximately 35 hours per seed and configuration. Subsequently labelling

Table 4: Generation hyperparameters.

PARAMETER	VALUE
MAX INSTRUCTION LENGTH	520
MAX NEW TOKENS	256
PPO EPOCHS	4
TOP-P	0.9 (1.0 FOR PPO)
TOP-K	0
TEMPERATURE	1.0

Table 5: Example answer generation and reward modeling prompts with proper formatting.

Answer generation prompt	**Reward modelling prompt**
`<|prompter|>Categorize the following items as either furniture or kitchen items.\nChair, Knife, Fork<|endoftext|> <|assistant|>`	`<|prompter|>Categorize the following items as either furniture or kitchen items.\nChair, Knife, Fork<|endoftext|> <|assistant|> Furniture: Chair, Kitchen: Knife, Fork<|endoftext|>`

the results with the 7B gold reward model takes approximately 18h when using an evaluation set of size 2000 and evaluating every 300 steps.

D Iterated RLHF and Performative Prediction

We note that the framework of performative prediction applies to our setting. In fact, when performing iterated RLHF we are simulating performative prediction or more specifically a version of strategic classification. We have that a reward model R_ϕ induces a potentially different distribution $\mathcal{D}(\phi)$ over instances (x, y) where continuations y are obtained from the policy π_θ optimised for R_ϕ, which yields that a reward model $R_{\phi_{PO}}$ is performatively optimal if $\phi_{PO} = \arg\min_\phi \mathbb{E}_{(x,y)\sim\mathcal{D}(\phi)} \ell((x, y, \phi))$.

Furthermore, [29] call a model $R_{\phi_{PS}}$ performatively stable if

$$\phi_{PS} = \arg\min_\phi \mathbb{E}_{(x,y)\sim\mathcal{D}(\phi_{ps})} \ell((x, y, \phi)).$$

Intuitively, retraining a performatively stable reward model after optimizing against it will yield the same reward model. As such the reward model would not be over-optimised and still perform optimally on its induced distribution. In their Theorem 3.5 the authors of [29] provide 3 conditions under which the reward model obtained from repeated iterations of RLHF converges to a unique performatively stable reward model at a linear rate. We require the loss to be β-jointly smooth and γ-strongly convex, and the map $\mathcal{D}(\cdot)$ from reward model parameters to the distribution of prompt continuation pairs to be ϵ-sensitive [29]. Since as part of the map $\mathcal{D}(\cdot)$ the policy is optimised with PPO, where small changes in the reward model can lead to significant changes in the optimal policy, this mapping is generally not ϵ-sensitive. As a consequence, linear convergence is not guaranteed. Note, that we may still aim for close to linear convergence by making adjustments to satisfy the stated conditions.

E Additional results

E.1 Closing the gap between proxy and gold reward function

Here we provide additional experimental results for taking the last preference dataset and sampling the preference datasets with equal proportion. In terms of the rate at which the gap between proxy and gold reward functions is reduced over iterations, the sampling strategy (see Figure 8) falls in between concatenating all preference data and taking only the last dataset (see Figure 9).

E.2 Additional results for combining preference data

In Figure 10 we provide the individual seeds for methods combining preference data across all iterations and in Figures 11 and 12 we provide the results for the sampling strategies. Figure 13 shows the MMD across iterations when only using the most recent preference dataset.

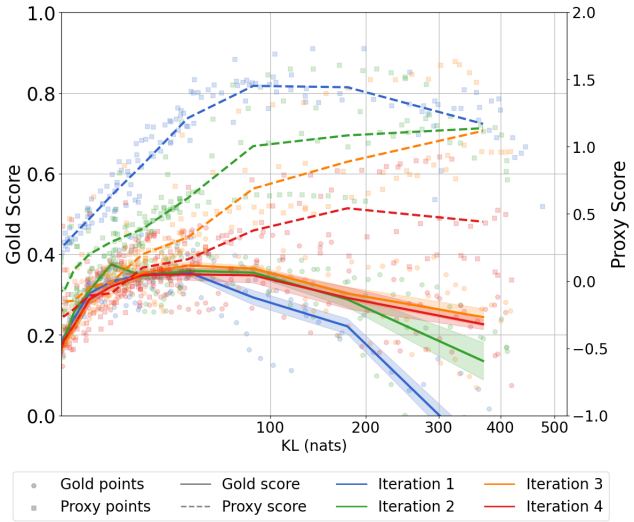


Figure 8: The gap between gold and proxy reward function when sampling from all preferences dataset equally to form the reward model training data.

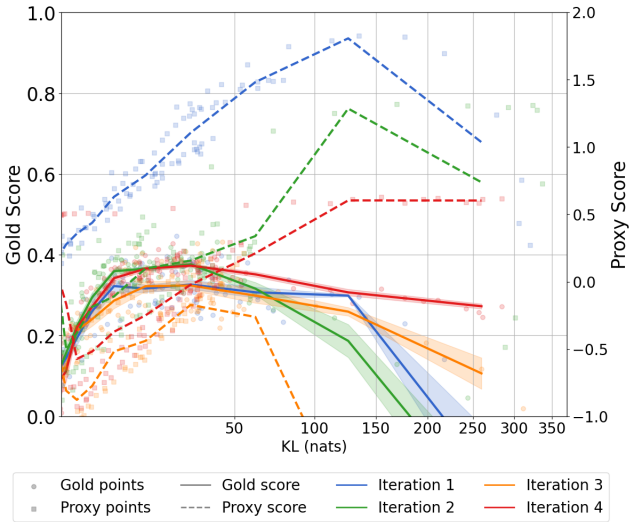


Figure 9: The gap between gold and proxy reward function when only taking the last preferences dataset for reward model training.

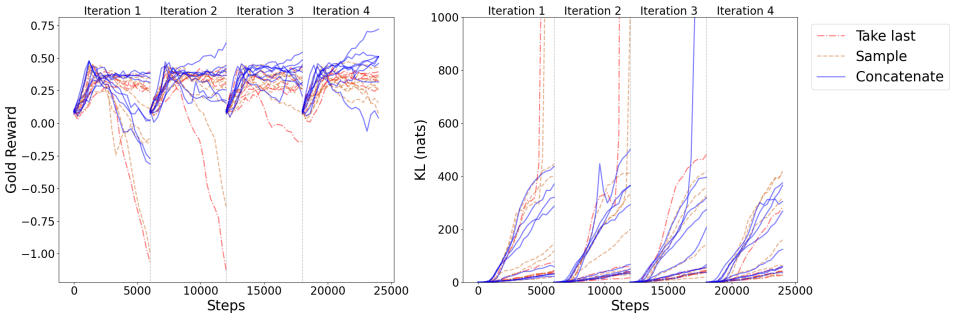


Figure 10: Gold score and KL of individual seeds across iterations for varying preference data combination methods.

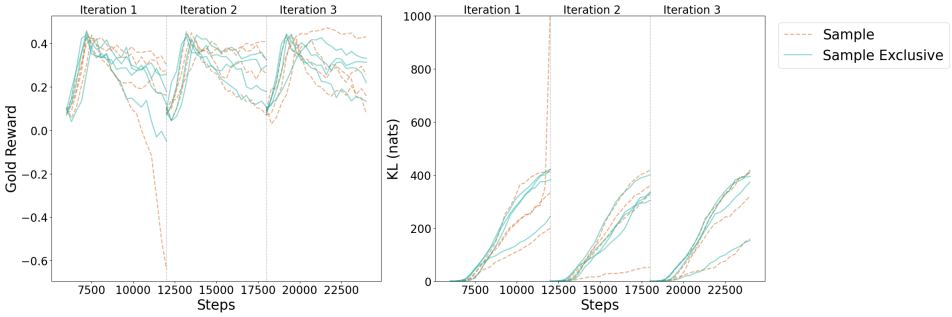


Figure 11: Gold score and KL of individual seeds across iterations comparing sampling with full coverage of the prompts vs random sampling.

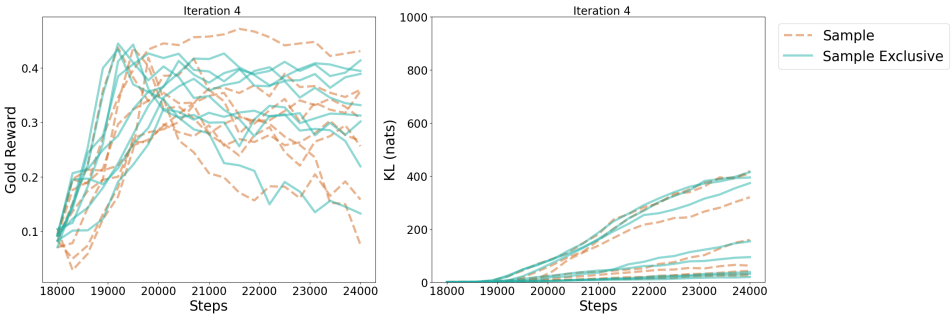


Figure 12: Gold score and KL of individual seeds in the fourth iteration comparing sampling with full coverage of the prompts vs random sampling.

E.3 Additional results for reward model transfer

Here we provide additional results for methods addressing reward model transfer. Figure 14 and 15 show the individual training seeds of the methods across iterations.

E.4 Additional results for policy initialisation

Here we provide additional results for the policy initialisation methods (Figures 16 and 17). In particular, we plot the runs associated with each seed, highlighting seeds that are strongly overoptimised and can not be recovered by the respective methods.

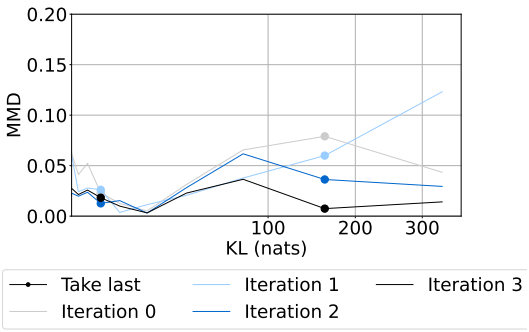


Figure 13: Taking the last preference dataset results in consistently low MMD, in the final iteration.

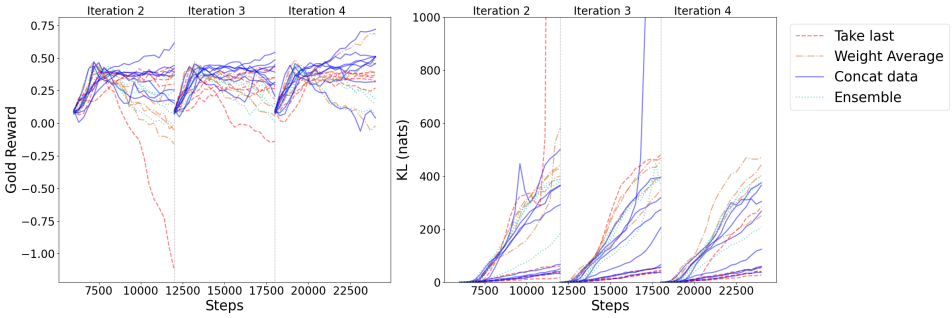


Figure 14: Gold score and KL of individual seeds across iterations comparing reward function choices.

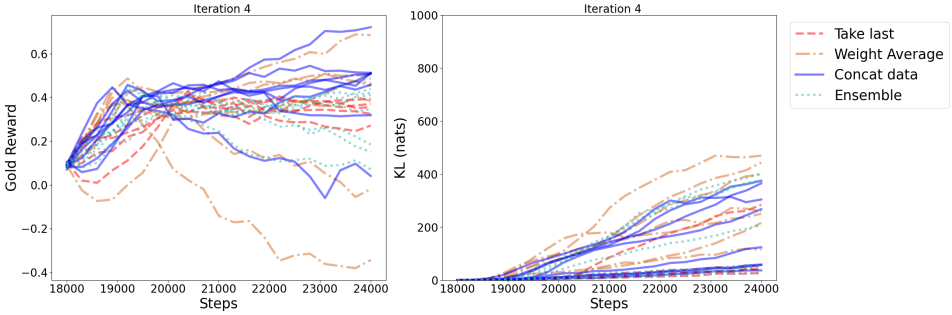


Figure 15: Gold score and KL of individual seeds in the fourth iteration comparing reward function choices.

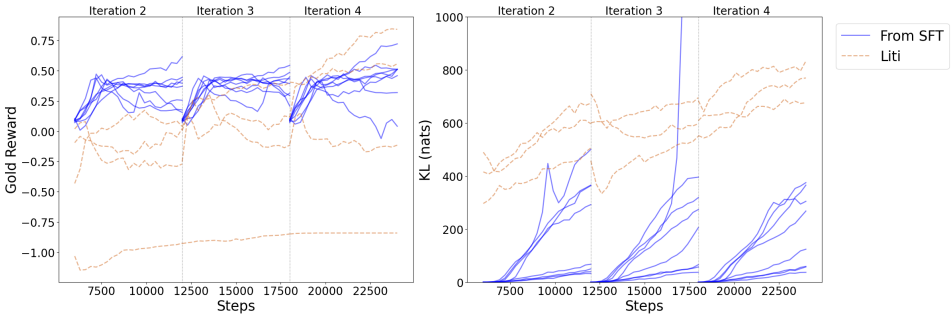


Figure 16: Gold score and KL of individual seeds across iterations comparing policy initialisation methods.

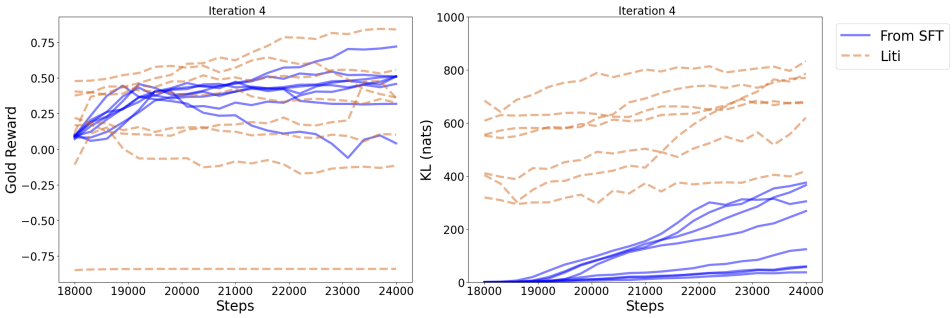


Figure 17: Gold score and KL of individual seeds in the final iteration comparing policy initialisation choices.