REWARD MODEL OVEROPTIMISATION IN ITERATED RLHF

Anonymous authors
Paper under double-blind review

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 its idiosyncrasies. 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 evaluate several iterated RLHF pipelines and measure their effect on overoptimisation. In particular, we systematically analyse 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 with policies of 410M and 1.4B parameters and comparing the distributions of proxy and gold rewards, we observe that overoptimisation decreases over successive iterations as reward models increasingly approximate ground-truth preferences, although the gains shrink in later iterations. Reinitialising the policy from the supervised fine-tuned model at every iteration is the most robust choice. Continuing from the previous policy works as long as that policy has not overoptimised, which larger reward models make possible, while an overoptimised policy is hard to recover. Concatenating the preference data of all iterations is the most effective mitigation, and together with policy resets it is the pipeline we recommend.

1 INTRODUCTION

Reinforcement learning from human feedback (RLHF) has become the standard method for aligning large language models with human preferences (Ziegler et al., 2020; Ouyang et al., 2022; Bai et al., 2022). However, RLHF faces a critical vulnerability: reward model overoptimisation (Gao et al., 2023). 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 (Bai et al., 2022; Xiong et al., 2024), practitioners aim to iteratively close the gap between proxy and true reward. Despite its widespread adoption in industry (Ziegler et al., 2020; Ouyang et al., 2022; Bai et al., 2022), it remains uncertain whether iterated RLHF genuinely resolves overoptimisation, merely postpones the inevitable exploitation of the reward model

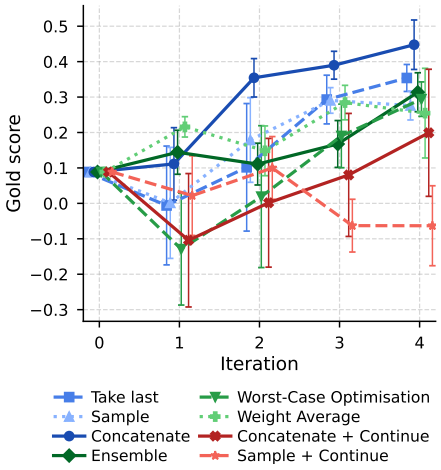


Figure 1: Iterated RLHF design choices in preference data management, reward function formulation, and policy initialisation strongly affect ground truth performance and overoptimisation.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

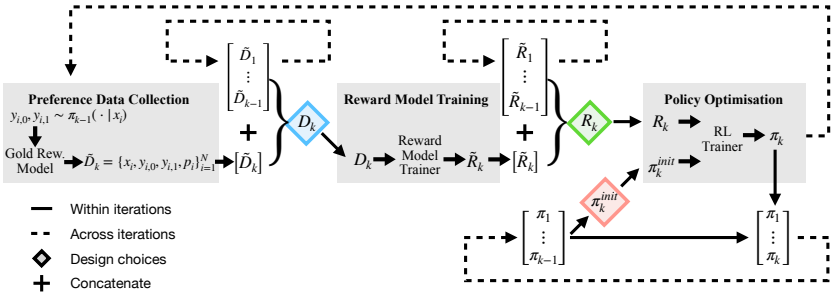


Figure 2: The Iterated RLHF framework performs multiple rounds of preference data collection, reward modelling, and policy optimisation. Our research reveals three design choices that impact performance: (1) how preference data is managed across iterations, (2) which reward function formulation to optimise,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

akin to persistent adversarial policies (Gleave et al., 2020), or perpetuates a recurring cycle of overoptimisation in different forms (Singhal et al., 2024).

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 2, 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 optimise 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 formalise iterated RLHF and its three design choices, and frame the process as performative prediction, which motivates data aggregation, reward model averaging and policy resets.
- We perform a systematic evaluation of key design choices with quantitative evidence of their impact on performance and overoptimisation.
- We provide practical guidelines for 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 (Coste et al., 2024; Gao et al., 2023) on the AlpacaFarm dataset (Taori et al., 2023) 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 1). Performance gains diminish in later iterations for most methods. Concatenating preference data across iterations outperforms other approaches. These patterns hold when the policy is scaled from 410M to 1.4B parameters. Small but persistent overoptimisation remains after four iterations regardless of design choices. 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 (Krakovna et al., 2020) by increasingly capable language models.

2 Related Work

RLHF is the standard for aligning large language models to human preference data. The iterated approach was first discussed by Bai et al. (2022) to fix robustness and calibration issues, attributed to lack of data in the high score regime and has since gained in popularity (Ramé et al., 2024; Xiong et al., 2024; Ye et al., 2024; Adolphs et al., 2023; Dong et al., 2024; Yuan et al., 2024). Besides training on newly collected preferences, an iterated scheme to train reward models from synthetically generated preference data has been proposed by Wang et al. (2024) and shown to improve performance on the reward model benchmark RewardBench (Lambert et al., 2025), but the authors do not study overoptimisation nor the design choices we consider. In the context of Direct Preference Optimisation (DPO) (Rafailov et al., 2023), offline, online, and hybrid approaches have

been investigated mostly in terms of sample efficiency (Xiong et al., 2024; Das et al., 2025; Muldrew et al., 2024; Mehta et al., 2025). More broadly iterated methods have been investigated for machine teaching (Liu et al., 2017) and to resolve feedback loops caused by model deployment in supervised learning (Perdomo et al., 2020) and also performative RL (Mandal et al., 2023).

Overoptimisation is a common issue in RL, and evidence of this has been frequently reported in the RLHF literature (Ziegler et al., 2020; Stiennon et al., 2020; Gao et al., 2023; Singhal et al., 2024). A promising method to mitigate overoptimisation is using reward model ensembles combined with conservative optimisation (Coste et al., 2024). Several works further explore reward model ensembles in RLHF (Eisenstein et al., 2024; Lou et al., 2024). Notably, Ramé et al. (2024) introduce weight averaged reward models (WARM) alleviating the inference cost of multiple reward models during training. Following Coste et al. (2024) and Gao et al. (2023) in tackling reward model overoptimisation, several works propose alternative approaches including reward model distillation (Fisch et al., 2025), hidden state regularisation (Yang et al., 2024b), and more (Yang et al., 2024a; Miao et al., 2024; Liu et al., 2024; Gorbatovski et al., 2025). One commonly reported mode of overoptimisation is length bias (Singhal et al., 2024; Park et al., 2024), which can be tackled by disentangling reward signals related to response length from content quality (Chen et al., 2024). Recent work by Gauthier et al. (2026) studies how policies can anticipate future reward-model updates. We instead study the stability of the standard iterative RLHF loop under key design choices. 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 RLHF 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. In Appendix A we frame iterated RLHF as performative prediction (Perdomo et al., 2020), which gives conditions for the reward model to converge across iterations and motivates data aggregation, reward model averaging and policy resets.

3.1 Single-iteration RLHF

Preference data collection. We start from a supervised fine-tuned policy π^{sft} and use it to collect preference data. The dataset $\mathcal{D}$ contains tuples $\{x_i, y_{i,0}, y_{i,1}, p_i\}$ for $i = 1, ..., N$, where $x_i \in \mathcal{X}$ is a prompt, $y_{i,j} \sim \pi^{\text{sft}}(\cdot|x_i)$ for $j = 0, 1$ are two responses from π^{sft}, and p_i indicates whether $y_{i,0}$ is preferred over $y_{i,1}$. Following Coste et al. (2024) and Gao et al. (2023), preferences p_i are simulated using a gold reward model $R^\star$, which is significantly larger in terms of parameter size than the proxy reward models, serving as approximation for human labels in RLHF. This use of the gold reward model is the standard approach for investigating overoptimisation without incurring significant costs and time bottlenecks due to human labelling (Coste et al., 2024; Gao et al., 2023). In Appendix E we conduct additional experiments with 25% label noise in the preference data.

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 minimising the cross-entropy loss on the preference dataset $\mathcal{D}$.

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 optimisation (PPO) algorithm (Schulman et al., 2017). 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 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 (Bai et al., 2022) and trained policies overoptimise the proxy reward model leading to unstable fine-tuned policies (Rafailov et al., 2024; Gao et al., 2023; Ziegler et al., 2020). Notably, policy optimisation 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.

Algorithm 1 Iterated RLHF (design choices highlighted).

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

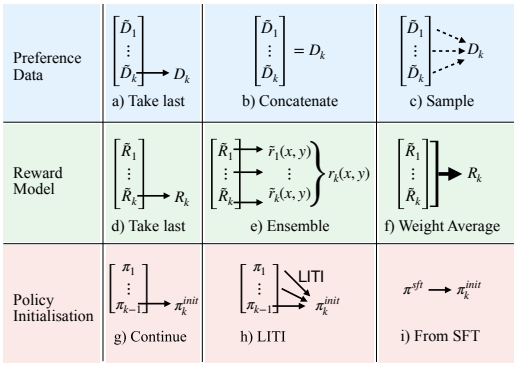


Figure 3: Design choices for Iterated RLHF (Algorithm 1). Options include how to combine preference data, transfer reward models, and initialise policies. These choices determine how learning signals are propagated through each iteration.

3.2 Iterated RLHF and design choices

The problem of the divergence between the distributions $\pi_\theta(y|x)$ and $\pi^{\text{sft}}(y|x)$ 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 2. 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 k^{th} 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 $\tilde{\mathcal{D}}_k$.

Sampling responses from the earlier policies again is unnecessary, since their preference data are already available from previous iterations. 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 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., full 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 subset of each $\tilde{\mathcal{D}}_i$ 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 form a single reward function R_k from the list of all previously trained proxy reward models $[\tilde{R}_1, ..., \tilde{R}_k]$?

Combining reward models. Errors of the reward model are inherited by the policy optimised against it and, over multiple iterations, by the preference data that policy generates. 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.

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) (Coste et al., 2024). Since reward model ensembles showed limited improvements in Coste et al. (2024) we also evaluate the ensemble with worst-case optimisation (WCO), i.e., optimising the minimum $R_k(x, y) = \min_{i=1,...,k} \tilde{R}_i(x, y)$. 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 (Ilharco et al., 2023). 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 the weight averaged reward model applies task arithmetic with $\lambda = 0.5$ on top of a base reward model ϕ_0 (Appendix C), setting $\phi_k = \phi_0 + \lambda \sum_{i=1}^{k} (\tilde{\phi}_i - \phi_0)$. 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. Bai et al. (2022) initialise the policy from π^{sft} at every iteration, so that the policy of the previous iteration is used only to sample preference data. We call this initialisation *From SFT* shown in Figure 3.i. As alternative, we use linear interpolation towards initialisation (*LITI*) (Ramé et al., 2024), which was inspired by WiSE-FT proposed by (Wortsman et al., 2022). With *LITI*, shown in Figure 3.h, we interpolate the previous policy towards π^{sft} and set $\pi_k^{\text{init}} = (1 - \eta)\pi^{\text{sft}} + \eta\pi_{k-1}$, where the hyperparameter η controls how much of the optimisation against R_{k-1} is retained. Taking $\eta = 1$ corresponds to initialising the current policy from the previously fine-tuned one, setting $\pi_k^{\text{init}} = \pi_{k-1}$, which we call *Continue* (Figure 3.g). 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 have formalised the process of iterated RLHF and highlighted the critical design choices. In this section, we detail our evaluation setup, emphasising the quantification of overoptimisation and examining how its progression over iterations is influenced by different design choices.

Training setup. Our evaluation setup follows extensive prior works that study overoptimisation in the single iteration RLHF in a controlled and simulated manner (Coste et al., 2024; Gao et al., 2023). Similarly to Coste et al. (2024) we use instructions from the *AlpacaFarm* dataset (Dubois et al., 2023) for reward model training and policy optimisation. The preference data $\tilde{\mathcal{D}}_k$ collected at each iteration contains preferences for the same subset of 1000 instructions from the preference split of AlpacaFarm. Preference labels p_i are simulated with the 7 billion parameter `Reward-Model-AlpacaFarm-Human` (Dubois et al., 2023). Similarly to Coste et al. (2024), to obtain π^{sft}, we performed supervised fine-tuning on the `pythia-410m` model (Biderman et al., 2023) on the AlpacaFarm SFT split. We use `pythia-410m` for the full study and repeat the policy-initialisation experiments with the 1.4B SFT policy released by Coste et al. (2024) to test whether the findings transfer (Section 5.4, Appendix F.7). We initialise proxy reward models $\tilde{R}_k$ from the HuggingFace checkpoint `pythia_70m_sft` provided by Coste et al. (2024), as well as the larger `pythia-160m`, with a randomly initialised prediction head (Coste et al., 2024). For policy optimisation, we perform 6000 steps of PPO on the unlabelled split of AlpacaFarm with a

constant KL penalty. 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 core results presented in our performance evaluation are reported for 8 random seeds, the 1.4B policy with 3 seeds. In curves against the KL to π^{sft}, each seed contributes the value it reaches at that KL, interpolated between evaluations, and a seed that stops at a lower KL keeps its final value. We plot the mean over seeds with one standard error (Appendix C.2).

Measuring overoptimisation with the Maximum Mean Discrepancy. The standard methodology for investigating reward model overoptimisation is to compare mean rewards on proxy vs. gold reward functions over a hold-out set (Coste et al., 2024; Moskovitz et al., 2024; Gao et al., 2023). This compares only the means of the two reward distributions. We instead compare reward models by their distributions of rewards, evaluating 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. In the first step, we standardise each reward function with the mean and standard deviation of its rewards on samples from π^{sft}, so that reward functions differing by a positive affine transformation, which induce the same ordering of policies, are treated as equal (see Appendix B.1). In the second step, we use the maximum mean discrepancy (MMD) (Gretton et al., 2012) 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 B.

5 Experimental results

When comparing different methods, we primarily focus on their performance in the final iteration. We also compare the performance of methods across multiple iterations, to see how the KL-reward curves change through the iterations. Additionally, we evaluate the final reward models for generalisation to external test sets with RewardBench (Lambert et al., 2025) (see Appendix D).

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 4a, 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 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, though the gains over the previous iteration decrease as the gap between the proxy and gold reward shrinks. We argue this is due to the proxy reward model progressively converging towards the gold reward model on the distribution induced by policy optimisation, which limits the scope for further improvement. In addition, data saturation may occur once additional preference data provides little novel information. However, there remains scope to better align gold and proxy reward functions. Comparing reward distributions across iterations further shows that the MMD grows with the KL in every iteration, fastest in the first (see Figure 4b). From the second iteration on it stays lower at moderate KL, while at the largest KL, which only the furthest-drifting seeds reach, the later iterations match the first. The MMD captures one aspect of overoptimisation, how far the proxy's assessment of the policy's outputs departs from the gold's, and the gold and proxy curves of Figure 4a capture another, whether gains on the proxy still translate into gains on the gold reward. At the largest KL the later reward models still overstate the gains, since their proxy reward grows well beyond a gold reward that on average still rises, whereas in the first iteration the gold reward of these seeds falls. Additionally, the rate at which the proxy-gold reward gap closes varies considerably among methods (see Appendix F.1), highlighting the importance of investigating design choices described in Section 3. On TL;DR summarisation the gap between the proxy and the gold reward likewise decreases across iterations (Appendix F.8).

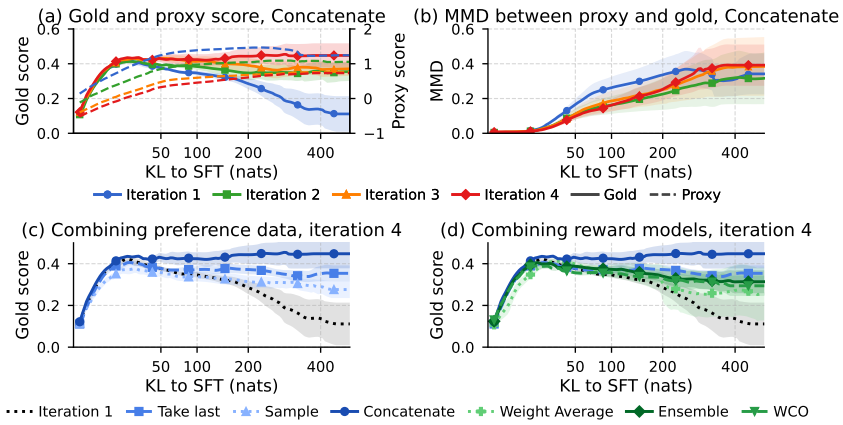


Figure 4: Overoptimisation decreases across iterations and concatenating preference data is the most effective design choice. (a) Gold (solid) and proxy (dashed) scores of *Concatenate* against the KL, whose gap narrows across iterations at diminishing returns. (b) The MMD between proxy and gold grows with the KL, fastest in the first iteration. (c) Concatenating all preference data mitigates overoptimisation best. (d) Combining reward models in parameter space gains efficiency but trails both *Concatenate* and *Take last*.

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 initialisation to *From SFT* and reward models are combined using the *Take last* approach. As shown in Figure 4c, all methods demonstrate significant improvements over a single iteration, particularly in preventing performance collapse at higher KL divergences.

The *Concatenate* strategy achieves consistently higher gold scores. While *Take last* and *Sample* approaches show similar trends and substantial improvements over iteration 1, they do not match the performance of full data concatenation. We attribute this robustness to the stabilisation of the "performative loop" (Appendix A). *Concatenate* approximates training on the mixture of all previous policy distributions (Corollary A.4), which anchors the reward model and keeps the feedback loop from following the idiosyncrasies of the current policy, whereas *Take last* trains only on the most recent distribution shift. A critical observation is that beyond $\mathrm{KL} \approx 200$, the baseline iteration 1 experiences severe performance degradation due to overoptimisation, dropping to the level of the SFT policy. In contrast, all iterative approaches maintain positive performance even at high KL values, demonstrating their effectiveness in mitigating overoptimisation. The lead of *Concatenate* is established from the second iteration on, as shown in Figure 1 and in Appendix F.2. To ablate the effect of the growing reward model training set over iterations, we run a single iteration with four times the preference data sampled from π^{sft}, covering four times as many prompts. While this matches the maximum gold reward of the fourth *Concatenate* iteration, it does not yield the same robustness, and the policy strongly overoptimises later in training, where the gold reward of the fourth *Concatenate* iteration declines much less (Appendix F.3).

Sampling, with or without prompt repetition, performs similarly, implying limited impact of prompt coverage (Appendix F.2).

5.3 Combining reward models

No free lunch by merging reward models. Concatenating all preference data, previously the most effective method, serves as our performance baseline. As shown in Figure 4d, all approaches improve similarly in early KL regions (up to ≈ 50 nats), reaching comparable performance. Neither *Weight Average*, which keeps the cost of a single reward model, nor *Ensemble*, which increases inference time and memory use, improves on the *Take last* approach, consistent with Coste et al. (2024). More specifically, Figure 4d shows that *Ensemble* does not outperform at the 70M scale. While weight averaging has been reported to outperform ensembles (Ramé et al., 2024), we observe it falling below the ensemble at high KL. In contrast to this prior work, we combine models trained on data with significantly different joint distribution over pairs (x, y). Regardless, both methods 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.

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 5, *Ensemble* and *Worst-Case Optimisation* gain more from the larger reward model than *Concatenate* and narrow the gap to 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, they become more competitive options with more expressive reward models.

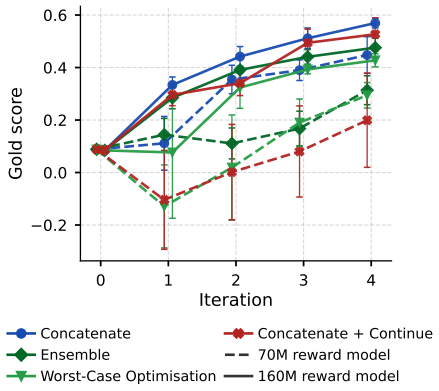


Figure 5: End-of-iteration gold score with 70M (dashed) and 160M (solid) reward models. With the larger reward model every method improves across iterations, and *Ensemble*, *Worst-Case Optimisation* and *Continue* end close to *Concatenate*.

5.4 Policy initialisation

Initialising from SFT is the most robust. Finally, comparing the policy initialisation methods we observe that none reaches the gold score of concatenating all preference data and initialising the policy from the SFT checkpoint, and the warm-started methods end at several times its KL (Figure 6). Sampling the preference data is similarly robust (Figure 4c, Appendix F.2), highlighting that initialising with *From SFT* results in generally reduced overoptimisation. Note, *Continue* starts each iteration from the KL accumulated in the previous ones, increasingly further from π^{sft}. Resetting the policy at each iteration combined with the aggregation of preference data results in consistently less overoptimisation and more performant policies. This aligns with Remark A.6, according to which resetting the policy bounds the cumulative KL divergence from the base distribution. However, 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, *Continue* and *LITI* have to recover from the initial overoptimisation, as shown in Figure 6. The KL to π^{sft} of *Continue* grows monotonically across iterations while its gold score stays close to the level of π^{sft}, whereas the counterpart trained on the same preference data but reset *From SFT* improves with each iteration. With *Take last*, every reward model is trained on samples from a policy that had itself overoptimised, and a policy reset from π^{sft} still improves under it, so the retrained reward models carry a usable signal even then.

Continue starts each iteration from the final policy of the previous one, whose responses have degenerated into repeated token sequences, and the two *Continue* arms show the highest trigram repetition and the lowest vocabulary diversity of all methods (Appendices F.11 and G). The policy exploits idiosyncrasies of the proxy reward model that a reward model retrained on samples from this region does not correct, so the exploitation compounds across iterations. Policy entropy collapses in a single seed only (Appendix F.9), so this is a mode of reward model exploitation distinct from entropy collapse. Interpolating towards π^{sft} (*LITI*) restores the gold score at every restart, and the restart holds after a healthy iteration. After a collapsed iteration the interpolated policy drifts back into a degenerate mode early in the next one, where the retrained reward model scores its responses low, but a policy whose responses have become nearly uniform across prompts yields almost no gradient to leave it (Appendix F.5). In these failing restarts the proxy falls together with the gold score, so for *LITI* the warm-started policy is what fails, and *LITI* ends at the level of *Continue* at the 410M scale and above it at the 1.4B scale. Thus, while *From SFT* is most robust, it is also limited by the repeated initialisation from π^{sft}.

Larger policies. Scaling the policy to 1.4B (right panels of Figure 6) shows the same overoptimisation patterns between design choices. Across iterations, the reward models of the first two iterations are exploited, the second at a higher KL than the first, and with *Concatenate* the reward models retrained

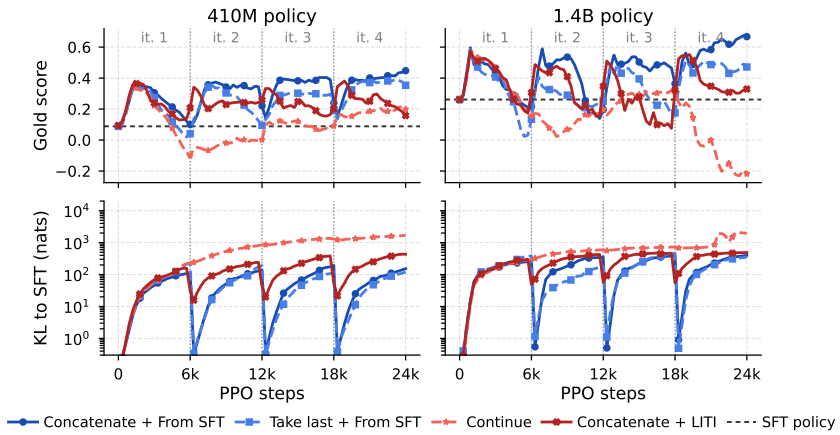


Figure 6: Gold score (top) and KL to π^{sft} (bottom) over four iterations at 410M and 1.4B policy size.

from the third iteration on remove the collapse. Between design choices, *Concatenate* ends highest and is still improving at the end of iteration 4, *Take last* recovers at a lower level, while *Continue* cannot recover and collapses in the final iteration. This aligns with the observation of Gao et al. (2023) that policy size is not a core driver of overoptimisation.

Continuing the policy works with larger reward models. With the 160M reward model *Continue* improves in every iteration and ends close to *From SFT* (Figure 5), while with the 70M reward model the first iteration overoptimises strongly and later iterations recover only part of the loss. Larger reward models are overoptimised more slowly (Gao et al., 2023), so the first iteration keeps most of its gain, the continued policy starts every later iteration from a policy that has not collapsed, and it drifts less in each subsequent iteration (Appendix F.6). *From SFT* remains the safe choice with less expressive reward models, and continued optimisation becomes viable as reward model capacity grows.

6 Limitations

Our experimental setup, consistent with prior work (Coste et al., 2024; Ramé et al., 2024; Zhu et al., 2024) (also in terms of model size and dataset), enables systematic investigation of iterative RLHF ensuring results remain interpretable and comparable. We verify at 1.4B that the findings transfer across a threefold increase in policy size, using three seeds; larger policies remain untested, and scaling laws suggest that the speed of overoptimisation may differ at larger scales (Gao et al., 2023). We also note that the use of a "gold" reward model is standard practice in the field (Coste et al., 2024; Gao et al., 2023) to ensure reproducibility and mitigate cost of human labelling. In Appendix E, we conduct experiments with simulated label noise, which show that our conclusions extend to this setting. Preference drift remains an interesting problem that is beyond the scope of this paper. We ran four iterations, which were sufficient to observe plateaus and overoptimisation trends, but further scaling was prohibitive given the computational resources of our institution.

7 Conclusion

In this work we have investigated 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. At both policy sizes the gains shrink after the largest one, which arrives in the second iteration at 410M and in the third at 1.4B. We recommend *Concatenate*, whose reward model training cost grows with the number of iterations but remains small compared with generation and policy optimisation (Appendix C). It is worth noting that larger reward models exhibit stronger performance when using *Ensemble* and *WCO* strategies. While completely eliminating overoptimisation remains unattainable, we have identified initialisation *From SFT* as the most robust approach at both policy sizes, 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.

AI USE STATEMENT

In this work, we used generative AI tools for checking and improving the text of the paper and for writing code for data analysis and visualisation. They were also used for testing the code for the experiments, which was written by humans. We have reviewed all AI-assisted work. All code and text were checked and verified by the authors. We take full responsibility for the final content of this work, including text, claims and artefacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We have taken several steps to ensure the reproducibility of our work. All models and datasets used in our study are publicly available, and Section 4 names the checkpoints we build on and describes the training and evaluation procedures. Appendix B details the comparison of reward models with the MMD and Appendix C reports the full set of hyperparameters, the prompt format and the computational cost. To account for variability, all core experiments are conducted across eight random seeds and the 1.4B experiments across three. We commit to releasing our code upon acceptance of the paper to further facilitate reproducibility.

REFERENCES

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.

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.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of the 40th International Conference on Machine Learning (ICML'23)*, 2023.

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.

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.

Nirjhar Das, Souradip Chakraborty, Aldo Pacchiano, and Sayak Ray Chowdhury. Active Preference Optimization for Sample Efficient RLHF. In *Proceedings of the 2025 Joint European Conference on Machine Learning and Knowledge Discovery in Databases (ECML'25)*, 2025.

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.

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.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, Peter Shaw, and Jonathan Berant. 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.

Adam Fisch, Jacob Eisenstein, Vicky Zayats, Alekh Agarwal, Ahmad Beirami, Chirag Nagpal, Pete Shaw, and Jonathan Berant. Robust preference optimization through reward model distillation. *Transactions on Machine Learning Research*, February 2025.

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.

Etienne Gauthier, Francis Bach, and Michael I Jordan. Explaining and Preventing Alignment Collapse in Iterative RLHF. *arXiv preprint arXiv:2605.04266*, 2026.

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.

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. In *Proceedings of the 13th International Conference on Learning Representations (ICLR'25)*, 2025.

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.

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.

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: 2026-09-25.

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.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, et al. RewardBench: Evaluating reward models for language modeling. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1755–1797, 2025.

Weiyang Liu, Bo Dai, Ahmad Humayun, Charlene Tay, Chen Yu, Linda B. Smith, James M. Rehg, and Le Song. Iterative Machine Teaching. In *Proceedings of the 34th International Conference on Machine Learning (ICML'17)*, 2017.

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.

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.

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

Viraj Mehta, Syrine Belakaria, Vikramjeet Das, Ojash Neopane, Yijia Dai, Ilija Bogunovic, Stefano Ermon Barbara E Engelhardt, Jeff Schneider, and Willie Neiswanger. Sample Efficient Preference Alignment in LLMs via Active Exploration. In *Proceedings of the 2nd Conference on Language Modeling (COLM'25)*, 2025.

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.

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.

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.

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.

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*, pp. 4998–5017, 2024.

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.

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.

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.

Alexandre Ramé, Nino Vieillard, L'eonard Hussenot, Robert Dadashi, Geoffrey Cideron, Olivier Bachem, and Johan Ferret. WARM: On the Benefits of Weight Averaged Reward Models. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, 2024.

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.

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

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.

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.

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.

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.

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.

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.

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.

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*, 2024a.

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)*, 2024b.

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.

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.

Banghua Zhu, Michael I. Jordan, and Jiantao Jiao. Iterative Data Smoothing: Mitigating Reward Overfitting and Overoptimization in RLHF. In *Proceedings of the 41st International Conference on Machine Learning (ICML'24)*, 2024.

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 THEORETICAL FRAMEWORK: ITERATED RLHF AND PERFORMATIVE PREDICTION

A.1 OVERVIEW

We note that the framework of performative prediction (Perdomo et al., 2020) can be applied 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 $D(\phi)$ over instances (x, y) where continuations y are obtained from the policy π_θ optimised for R_ϕ, which yields that a reward model $R_{\phi_{\mathrm{PO}}}$ is performatively optimal if $\phi_{\mathrm{PO}} = \arg\min_\phi \mathbb{E}_{(x,y)\sim D(\phi)} \ell(\phi; (x, y))$. Furthermore, a model $R_{\phi_{\mathrm{PS}}}$ is defined as performatively stable if $\phi_{\mathrm{PS}} = \arg\min_\phi \mathbb{E}_{(x,y)\sim D(\phi_{\mathrm{PS}})} \ell(\phi; (x, y))$.

Intuitively, retraining a performatively stable reward model after optimising 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 Theorem 3.5 Perdomo et al. (2020) 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 $D(\cdot)$ from reward model parameters to the distribution of prompt continuation pairs to be ε-sensitive. Since as part of the map $D(\cdot)$ 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.

In the following subsections we expand on the overview above and present a concise theoretical account of iterated RLHF. We use ϕ to denote reward-model parameters and θ to denote policy parameters. Our presentation casts iterated RLHF as a performative prediction problem, and then derives sufficient conditions for convergence as well as a set of practical propositions that explain common empirical mitigations (data aggregation, reward model ensembles, and policy resetting).

A.2 SETUP

Let π_θ be a stochastic policy parameterised by $\theta \in \Theta$. Let R_ϕ be a learned reward model parameterised by $\phi \in \Phi$. We denote by $\Pi(R_\phi) \mapsto \pi_{\theta(\phi)}$ the policy optimisation operator that (approximately) returns a policy optimised with respect to R_ϕ. Running $\pi_{\theta(\phi)}$ in the environment (or simulator) induces a distribution over prompts and model responses; we write $D(\phi)$ for the resulting distribution over observed preference pairs or *(input, response)* tuples (x, y).

A reward model is trained by empirical risk minimisation on data sampled from the distribution induced by the current reward model through policy optimisation. Concretely, given a loss function $\ell(\phi; (x, y))$ (for example cross-entropy or a surrogate for pairwise preference loss), the standard iterated update considered throughout this work, with t indexing iterations, can be defined as follows:

$$\phi^{(t+1)} = \arg\min_\phi \ \mathbb{E}_{(x,y)\sim D(\phi^{(t)})} \big[\ell(\phi; (x, y))\big]. \tag{1}$$

This framing matches the performative prediction viewpoint: the object being learned (the reward model) affects the data distribution through the downstream policy it induces.

A.3 PERFORMATIVE STABILITY

Definition A.1 (Performative stability)**.** A reward model parameter ϕ^* is called performatively stable if it is a fixed point of the update equation 1, i.e.,

$$\phi^* = \arg\min_\phi \ \mathbb{E}_{(x,y)\sim D(\phi^*)} \big[\ell(\phi; (x, y))\big]. \tag{2}$$

At a performatively stable point, retraining the reward model on data produced by the policy it induces produces no change. Iterated RLHF can therefore be interpreted as an algorithmic attempt to reach such a fixed point.

A.4 Convergence guarantees

The following restates Theorem 3.5 of Perdomo et al. (2020) in our notation.

Theorem A.2 (Convergence to a performatively stable point)**.** *Suppose the loss $\ell(\phi; z)$, with $z = (x, y)$, is γ-strongly convex in ϕ and β-jointly smooth, that is, $\nabla_\phi \ell(\phi; z)$ is β-Lipschitz in ϕ and in z. Suppose further that the map $\phi \mapsto D(\phi)$ is ε-sensitive, $W_1(D(\phi), D(\phi')) \leq \varepsilon\|\phi - \phi'\|_2$, with W_1 the Wasserstein-1 distance. If $\varepsilon < \gamma/\beta$, the iterates of the update equation 1 satisfy $\|\phi^{(t)} - \phi^*\|_2 \leq (\varepsilon\beta/\gamma)^t\|\phi^{(0)} - \phi^*\|_2$ and converge linearly to a unique performatively stable point ϕ^*.*

Proof sketch. For a fixed distribution, strong convexity gives a unique minimiser. Joint smoothness and ε-sensitivity bound the distance between the minimisers for $D(\phi)$ and $D(\phi')$ by $(\varepsilon\beta/\gamma)\|\phi - \phi'\|_2$, so the update is a contraction whenever $\varepsilon < \gamma/\beta$, and Banach's fixed point theorem gives a unique fixed point and linear convergence.

Discussion. The theorem isolates two failure modes in practice: (i) the loss used to train reward models is rarely globally strongly convex in modern neural parameterisations, and (ii) modern policy optimisers (PPO, SAC, etc.) can induce highly non-Lipschitz changes in the data distribution, i.e., small changes to ϕ may yield large shifts in $\pi_{\theta(\phi)}$ and hence in $D(\phi)$. Consequently, the sufficient conditions above are not satisfied in general RLHF pipelines, but they nevertheless clarify why certain regularisers and protections (e.g., constraining policy updates, aggregating data) promote stable behaviour.

A.5 Preference data aggregation

Proposition A.3 (Data aggregation reduces estimation error)**.** *Let the reward model be trained by empirical risk minimisation on a dataset $\mathcal{S}$ of size N. Standard concentration bounds for N independent samples give an expected generalisation error of the empirical minimiser of order $O(1/\sqrt{N})$. If datasets $\mathcal{S}_1, \ldots, \mathcal{S}_T$ are collected sequentially, with $\mathcal{S}_t$ drawn from $D(\phi^{(t)})$ and $\phi^{(t)}$ trained on $\mathcal{S}_1, \ldots, \mathcal{S}_{t-1}$, and are concatenated to form $\mathcal{S}_{\text{tot}}$ with total size $N_{\text{tot}} = \sum_{t=1}^{T} N_t$, the samples are not independent, but each sample in $\mathcal{S}_t$ is drawn from $D(\phi^{(t)})$ conditionally on the data collected before it. The estimation error with respect to the mixture $\sum_{t=1}^{T}(N_t/N_{\text{tot}})\, D(\phi^{(t)})$ still decreases as $O(1/\sqrt{N_{\text{tot}}})$.*

Proof sketch. For a fixed ϕ, the centred losses $\ell(\phi; z_i) - \mathbb{E}_{z\sim D(\phi^{(t(i))})}[\ell(\phi; z)]$, where $t(i)$ is the iteration in which sample z_i was collected, form a martingale difference sequence with respect to the data collected so far. For a bounded loss the Azuma–Hoeffding inequality bounds their average by $O(1/\sqrt{N_{\text{tot}}})$, and a union bound over a finite or discretised reward model class gives the uniform bound required for the empirical minimiser.

Corollary A.4. *Training on aggregated data approximates training on the mixture distribution $D_{\text{mix}} = \frac{1}{T}\sum_{t=1}^{T} D(\phi^{(t)})$, reducing variance and decreasing sensitivity to idiosyncrasies of any single iteration.*

Discussion. Aggregation stabilises training in two ways: it increases effective sample size (reducing estimation noise) and smooths the effective data generating process, which can reduce the Lipschitz constant of $\phi \mapsto D(\phi)$ empirically.

A.6 Reward-model ensembles and transfer

Proposition A.5 (Averaging reduces squared error)**.** *Let $R_{\phi_i} = R^\star + \delta_i$ be K proxy reward models with additive errors δ_i. Define the ensemble reward as $R_{\text{ens}} = \frac{1}{K}\sum_{i=1}^{K} R_{\phi_i} = R^\star + \frac{1}{K}\sum_{i=1}^{K}\delta_i$. Then*

$$\|R_{\text{ens}} - R^\star\|_2^2 = \Big\| \frac{1}{K} \sum_{i=1}^{K} \delta_i \Big\|_2^2 \le \frac{1}{K} \sum_{i=1}^{K} \|\delta_i\|_2^2. \tag{3}$$

Proof. Let $\bar{\delta} = \frac{1}{K} \sum_{i=1}^{K} \delta_i$. Expanding the squares gives

$$\frac{1}{K} \sum_{i=1}^{K} \|\delta_i\|_2^2 - \|\bar{\delta}\|_2^2 = \frac{1}{K} \sum_{i=1}^{K} \|\delta_i - \bar{\delta}\|_2^2 \ge 0, \tag{4}$$

so the squared error of the ensemble is never larger than the average squared error of its members, and the two differ by the spread of the member errors around their mean. The inequality also follows from Jensen's inequality, since $\|\cdot\|_2^2$ is convex.

Discussion. When errors δ_i are approximately zero-mean and weakly correlated, ensembles can substantially reduce the magnitude of systematic errors that policies can exploit. Worst-case ensemble strategies (e.g., conservative lower-bound ensembles) further limit reward overestimation.

A.7 Policy initialisation and reset strategies

Let π_{θ_0} denote a base supervised fine-tuned (SFT) policy. Let us define the Kullback–Leibler divergence between two policies by $D_{\text{KL}}(\pi_\theta \| \pi_{\theta_0})$.

Remark A.6 (Resetting bounds policy drift). If at every iteration the policy optimisation is initialised from the base policy π_{θ_0} (i.e. we re-start optimisation from θ_0), then the accumulated divergence from the base policy over iterations is bounded by the per-iteration optimisation step sizes. In contrast, warm-starting from the previous iterate θ_{t-1} can lead to cumulative drift: divergences can add across iterations and become large.

Discussion. Resetting is an effective empirical safeguard against runaway behaviour and can improve reproducibility at the cost of reduced per-iteration adaptivity.

A.8 Overoptimisation (error-to-gap) bound

Proposition A.7 (Error–to–gap bound)**.** *Suppose the reward model approximation error is uniformly bounded,* $|R_\phi(y) - R^\star(y)| \le \Delta$ *for all outputs* y*. Then any output* $\hat{y} \in \arg\max_y R_\phi(y)$ *satisfies* $R^\star(\hat{y}) \ge \max_y R^\star(y) - 2\Delta$.

Proof. For $y^* \in \arg\max_y R^\star(y)$, we have $R^\star(\hat{y}) \ge R_\phi(\hat{y}) - \Delta \ge R_\phi(y^*) - \Delta \ge R^\star(y^*) - 2\Delta$.

Discussion. Bounding the sup-norm error of the reward model controls the extent to which an optimiser can overestimate the true reward. The preceding propositions (aggregation and ensembling) are practical mechanisms for reducing Δ and hence for limiting overoptimisation. Furthermore, framing iterated RLHF as a performative prediction problem clarifies both desirable algorithmic choices and structural failure modes. Under favourable convexity and Lipschitz conditions one recovers a contraction argument guaranteeing convergence to a unique performatively stable reward model. In realistic RLHF pipelines these conditions fail, but the theory explains why mitigation strategies (i.e., data aggregation, reward model ensembles, and policy resets) improve stability: they reduce estimation variance, shrink reward-model error, and bound policy drift. Together these tools help iterated RLHF approximate performatively stable equilibria in practice, even when exact theoretical conditions are not met.

B 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 on samples from π^{sft}, and apply the same scaling at every evaluation. This ensures that reward functions which differ only by a positive affine transformation are treated as equal to one another after scaling (Appendix B.1). This is desirable since positive affine transformations do not affect the ordering over policies induced by the original and transformed reward functions when they are optimised Skalse et al. (2024). Because the reference distribution is fixed, the value at an evaluation depends only on the rewards observed at that evaluation.

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, $x_i \overset{\text{i.i.d.}}{\sim} \rho$ with ρ the prompt distribution, 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.1). 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 (Gretton et al., 2012). We use the squared exponential kernel $k(r, r') = \exp(-(r - r')^2/2)$ with unit bandwidth, where $r, r' \in \mathbb{R}$ are standardised rewards.

Given samples $\mathbf{r}_{\phi_1} := \{r_{\phi_1,1}, ..., r_{\phi_1,n}\}$ and $\mathbf{r}_{\phi_2} := \{r_{\phi_2,1}, ..., r_{\phi_2,n}\}$, where $r_{\phi_m,i}$ is the standardised reward that R_{ϕ_m} assigns to the i-th of the n evaluation responses, the estimate MMD_u^2 of the squared MMD (Gretton et al., 2012) is obtained by

$$\begin{aligned}\text{MMD}_u^2[\mathbf{r}_{\phi_1}, \mathbf{r}_{\phi_2}] = &\frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k\left(r_{\phi_1,i}, r_{\phi_1,j}\right) \\ &+ \frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k\left(r_{\phi_2,i}, r_{\phi_2,j}\right) \\ &- \frac{2}{n^2} \sum_{i=1}^{n} \sum_{j=1}^{n} k\left(r_{\phi_1,i}, r_{\phi_2,j}\right).\end{aligned} \tag{5}$$

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.1 PROOFS

Proposition B.1. *Let R_{ϕ_1}, R_{ϕ_2} 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 and μ_i the mean of R_{ϕ_i} on samples from π^{sft}.*

Proof of Proposition B.1. Write R_i for R_{ϕ_i} and R_i' for $R_{\phi_i'}$. 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

$$R_2' = \frac{R_2 - \mu_2}{\sigma_2} \tag{6}$$

$$= \frac{R_2 - b'}{a'} \tag{7}$$

$$= \frac{a' R_1' + b' - b'}{a'} \tag{8}$$

$$= R_1'. \tag{9}$$

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. We write X_i, Y_i and R_i for the random variables whose realisations are x_i, y_i and r_i. 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:

$$P(X_i, X_j) = P(X_i)P(X_j) \text{ for } i \neq j. \tag{10}$$

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

$$P(Y_i, Y_j \mid X_i, X_j) = P(Y_i \mid X_i) P(Y_j \mid X_j) \tag{11}$$

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

$$P(Y_i, Y_j) = \iint P(Y_i, Y_j \mid X_i, X_j) P(X_i, X_j) \, dX_i dX_j. \tag{12}$$

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

$$P(Y_i, Y_j) = \iint P(Y_i \mid X_i) P(Y_j \mid X_j) P(X_i) P(X_j) \, dX_i dX_j. \tag{13}$$

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

$$P(Y_i, Y_j) = \left(\int P(Y_i \mid X_i) P(X_i) \, dX_i \right) \times \left(\int P(Y_j \mid X_j) P(X_j) \, dX_j \right). \tag{14}$$

This shows that

$$P(Y_i, Y_j) = P(Y_i) P(Y_j), \tag{15}$$

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

$$P(Y_i = y) = \int P(Y_i = y \mid X_i = x) P(X_i = x) \, dx. \tag{16}$$

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 $R_i = R_\phi(X_i, Y_i)$ where R_ϕ is a 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_ϕ is a deterministic function, R_i is fully determined by (X_i, Y_i). Specifically

$$R_i = R_\phi(X_i, Y_i) \text{ and } R_j = R_\phi(X_j, Y_j). \tag{17}$$

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_ϕ 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 $Y_i \sim \pi(\cdot \mid X_i)$, the distribution of (X_i, Y_i) is the same for all i. The function R_ϕ is deterministic and applies the same transformation to each pair (X_i, Y_i). Therefore, the distribution of $R_i = R_\phi(X_i, Y_i)$ 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 Coste et al. (2024). The parameters for supervised fine-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. The base reward model ϕ_0 of *Weight Average* is R^{init} trained for one epoch on the AlpacaFarm preference split. In the first iteration this arm optimises the reward model whose weights are the mean of ϕ_0 and $\tilde{R}_1$. With $\lambda = 0.5$ the combined reward model equals the average of the proxy reward models for $k = 2$ and extrapolates along their mean task vector for larger k.

Table 1: SFT hyperparameters.

Parameter	Value
Learning rate	$8\text{e}-6$
Epochs	3
Batch size	4

Table 2: RM hyperparameters.

Parameter	Value
Learning rate	$1\text{e}-5$
Epochs	5
Batch size	32

Table 3: PPO hyperparameters.

Parameter	Value
Learning rate	$1\text{e}-6$
Cosine annealing scheduler	$1\text{e}-7$
KL penalty coefficient (constant)	$1\text{e}-4$
PPO steps	6000
LITI interpolation weight η	0.3
Batch size	32
Number of rollouts	256
Chunk size	32
Clipping range & value	0.2
GAE lambda	0.95

C.2 Aggregation across seeds

For curves against the KL to π^{sft}, each seed contributes its value at the point where its run first reaches that KL, interpolated between evaluations. A seed that stops at a lower KL keeps its final value beyond that point. The curves show the mean over seeds with one standard error. Figures 1 and 5 show the mean at the end of each iteration, and Figures 6 and 20 the mean at each PPO step, using the geometric mean for the KL. The remaining appendix figures against the KL average all evaluations that fall into the same KL range.

C.3 Dataset

We use the instructions and inputs contained in the popular AlpacaFarm dataset (Dubois et al., 2023; Taori et al., 2023). The entire dataset contains $52,000$ samples split into "sft" (10k), "preference" (20k), "unlabeled" (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 "unlabeled" split.

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

C.4 PROMPT FORMAT

We follow the prompt format used in (Coste et al., 2024; Köpf et al., 2023), which is that of the v2 format used in Open Assistant. 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.

Table 5: Example answer generation and reward modelling 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|>

C.5 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 the results with the 7B gold reward model takes approximately 18h when using an evaluation set of size 2000 and evaluating every 300 steps.

Inference overhead for reward models. Reward model ensembles and WCO incur an inference cost proportional to the number of models K (in our case K models must be evaluated). The number of models to be evaluated grows with each iteration. Weight averaging has the significant advantage of zero inference overhead compared to a single model, as the combination happens in parameter space.

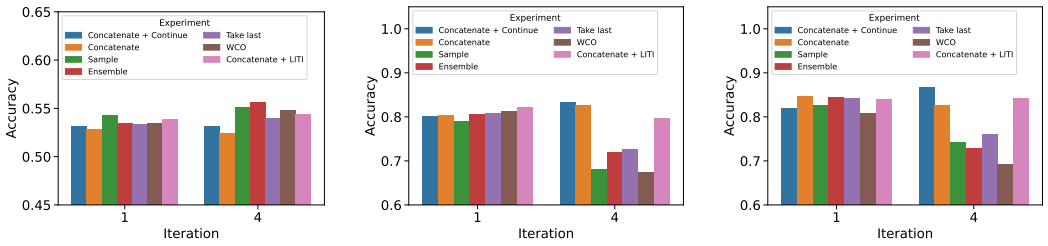
Cost performance trade-offs. We summarise the trade-offs based on our results as follows:

- *Concatenate + From SFT (High Robustness, Moderate Cost Increase):* While the reward model training cost scales linearly for *Concatenate*, it is worth noting that RM training is often faster than the generation and PPO phases of the pipeline. Thus, the overhead of concatenating data is often negligible compared to the cost of a failed run.
- *Take last + Ensemble/WCO (High Cost):* This has high inference costs during PPO (number of forward passes scales linearly with the iteration) but constant reward model training cost. We observed that for smaller models (70M), the computational overhead of ensembles does

not yield relative gains over simple data aggregation. For larger reward models performance is more comparable.

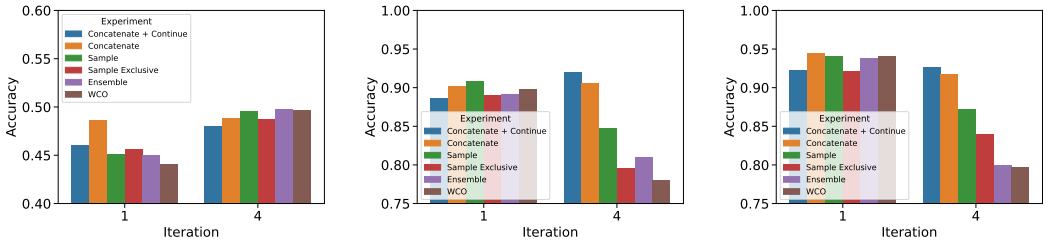
- *Weight Average (High Efficiency):* This method has no inference overhead and standard training costs, i.e., it is the most efficient. However, our results show that it is less effective at mitigating overoptimisation than for example *Concatenate*.

Our findings suggest that the cheapest method, *Take last*, ends below *Concatenate* at both policy sizes. Therefore, spending the marginal extra compute on *Concatenate* is likely the most efficient option in the long-run.



(a) Overall accuracy (b) AlpacaEval Easy (c) AlpacaEval Hard

Figure 7: RewardBench performance for 70M reward models.



(a) Overall Accuracy (b) AlpacaEval Easy (c) AlpacaEval Hard

Figure 8: RewardBench performance for 160M reward models.

D REWARD MODEL EVALUATION ON REWARDBENCH

We evaluate the reward models from the first and final iterations obtained via the different design choices on RewardBench (Lambert et al., 2025). In particular, since we train on AlpacaFarm, we report the performance on the AlpacaEval Easy and Hard splits as well as the overall accuracy averaged across all subsets of the benchmark (Figure 7 and Figure 8). *Sample Exclusive* is the variant of *Sample* in which each prompt appears exactly once in the reward model training data (Appendix F.2).

The change in accuracy on the AlpacaEval Hard and Easy subsets from the first to the final iteration separates the design choices by how they combine preference data (Table 6). At both reward model sizes, every arm trained on concatenated preference data keeps its accuracy, with changes between -0.03 and $+0.05$, whereas every other arm loses between 0.06 and 0.14. We read this as the concatenated data preserving the reward models' accuracy on responses far from the current policy, independently of the gold reward the arm reaches.

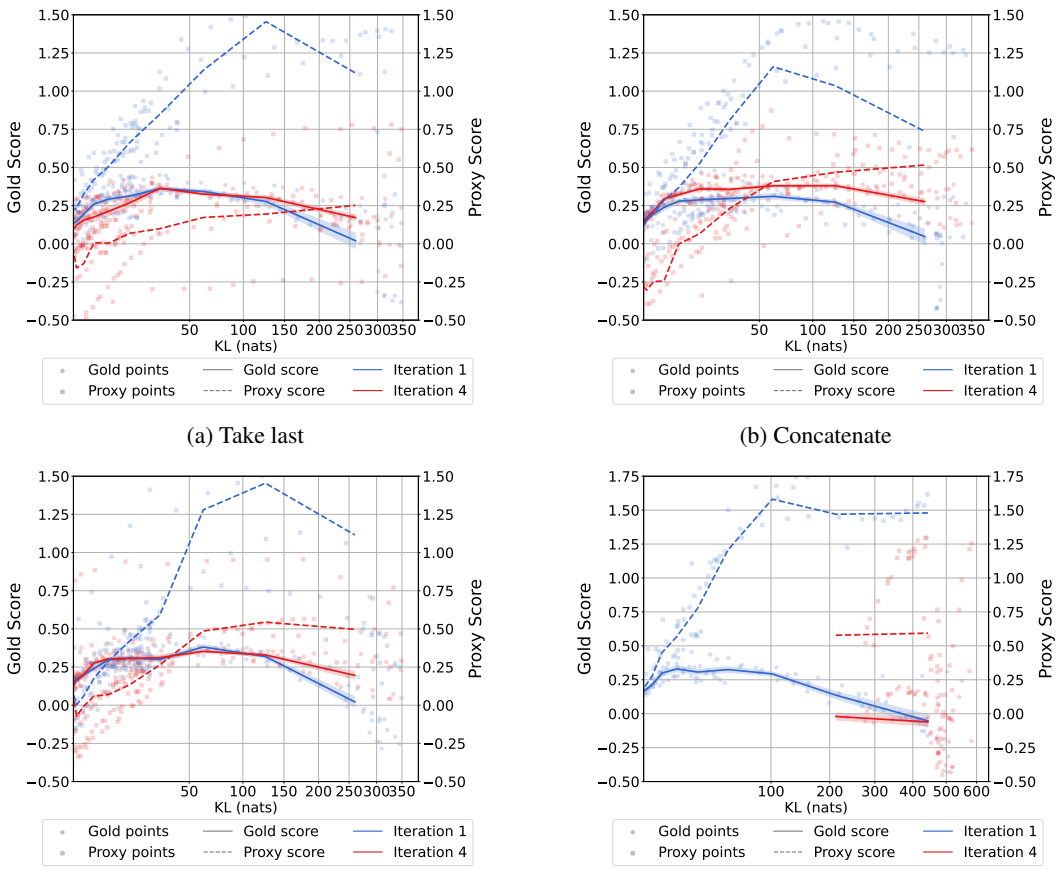
While this evaluation on RewardBench provides an additional perspective on the generalisation of the proxy reward models, we note that RewardBench performance is less diagnostic for our setting than gold-reward alignment. Due to distribution shift, reward models in later iterations may effectively mitigate overoptimisation even if their accuracy on the RewardBench test set is lower.

Table 6: RewardBench results grouped by design choice and reward model size.

RM Size	Experiment	Iteration	Accuracy	AlpacaEval Hard	AlpacaEval Easy	AlpacaEval Length
70m	WCO	4	0.548	0.692	0.673	0.587
		1	0.534	0.807	0.813	0.609
	Take last	4	0.540	0.759	0.727	0.618
		1	0.534	0.841	0.807	0.582
	Sample	4	0.551	0.742	0.681	0.609
		1	0.543	0.825	0.789	0.604
	Ensemble	4	0.557	0.728	0.719	0.601
		1	0.534	0.845	0.805	0.584
	Concatenate	4	0.524	0.825	0.826	0.582
		1	0.529	0.846	0.802	0.597
	Concatenate + Continue	4	0.531	0.866	0.833	0.587
		1	0.532	0.820	0.801	0.613
	Concatenate + LITI	4	0.545	0.841	0.796	0.596
		1	0.539	0.840	0.821	0.621
160m	WCO	4	0.497	0.797	0.780	0.568
		1	0.441	0.941	0.897	0.530
	Sample Exclusive	4	0.487	0.840	0.795	0.565
		1	0.456	0.921	0.890	0.543
	Sample	4	0.496	0.872	0.848	0.583
		1	0.451	0.941	0.909	0.534
	Ensemble	4	0.498	0.800	0.810	0.576
		1	0.450	0.938	0.891	0.520
	Concatenate	4	0.489	0.917	0.905	0.628
		1	0.486	0.945	0.901	0.541
	Concatenate + Continue	4	0.480	0.926	0.920	0.617
		1	0.460	0.922	0.886	0.537

E Results with Label Noise

To simulate noisy preferences we conduct experiments with 25% label noise. In particular, in the preference labelling phase, each preference label is flipped with probability 0.25. This probability has been found to be empirically consistent with real-world data and has been used to study label noise in prior works (Coste et al., 2024). In Figure 9 we observe that the hierarchy among design choices as well as the trends across iterations are consistent with the results obtained without label noise. We still observe severe overoptimisation in the initial iteration, which some design choices can mitigate towards the final iteration. Additionally, all methods achieve lower gold reward, which is an expected effect of label noise.



(a) Take last

(b) Concatenate

(c) Ensemble

(d) Concatenate + Continue

Figure 9: Ablation with 25% label noise in the preference labels the 70M reward model is trained on. The hierarchy of design choices as well as the trends across iterations are consistent with the corresponding runs without label noise. Overall, all methods achieve lower gold reward than without label noise, which is expected.

F Additional results

F.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 10) falls in between concatenating all preference data and taking only the last dataset (see Figure 11).

F.2 Additional results for combining preference data

In Figure 13 we provide the individual seeds for methods combining preference data across all iterations and in Figures 14 and 15 we provide the results for the sampling strategies.

Ensuring full coverage of the prompts when sampling matters less. Like taking only the newest preference dataset, the sampling strategy did not achieve the same level of performance as concatenating all data. Here we take a closer look at the sampling strategy. Figure 12 compares standard sampling with potential prompt repetition (*Sample*) and sampling where each prompt appears exactly once (*Sample Exclusive*). The differences are minor, suggesting that prompt repetition has a limited impact on performance or overoptimisation. This pattern also holds in earlier iterations (Figure 14), highlighting that while data combination strategies are effective at preventing

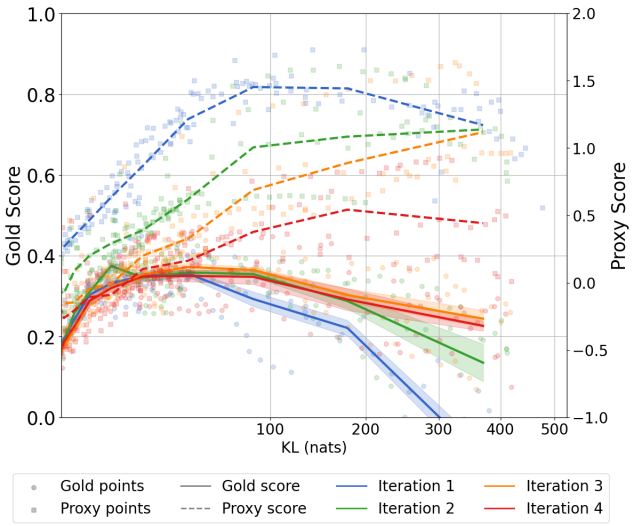


Figure 10: The gap between gold and proxy reward function when sampling from all preferences dataset equally to form the reward model training data.

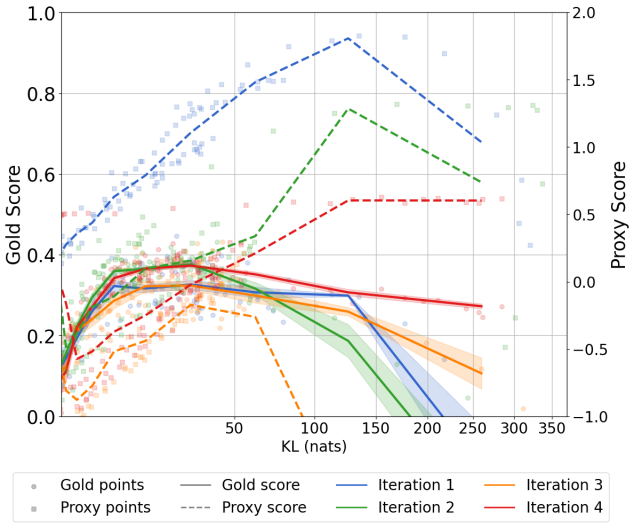


Figure 11: The gap between gold and proxy reward function when only taking the last preferences dataset for reward model training.

overoptimisation, the computational cost of maintaining and training on growing datasets remains, as more efficient methods are unable to achieve the same performance as *Concatenate*. This motivates exploring reward-model combination in parameter space to achieve similar gains with less overhead.

F.3 Data volume versus iteration

To test whether the benefit of iterated retraining is a data-volume effect, we train a single iteration on four times the preference data, all sampled from π^{sft}, matching the total amount of data seen by *Concatenate* after four iterations. Figure 16 compares the resulting policy with the first and fourth *Concatenate* iterations on the same seeds. Scores are shown relative to each arm's value at π^{sft} because proxy scores are standardised per reward model, and shaded bands are standard errors over the evaluations pooled in each KL bin. The control matches the number of preference pairs and exceeds the number of prompts, since its reward model is trained on 4000 distinct prompts whereas *Concatenate* re-samples the same 1000 prompts in every iteration. Despite this advantage,

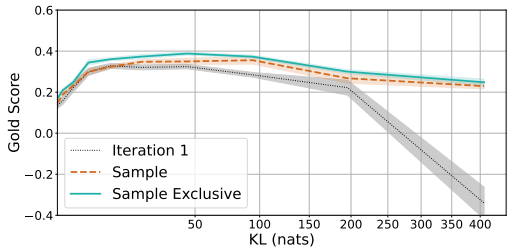


Figure 12: Ensuring prompt coverage. Sampling, with or without prompt repetition, performs similarly, implying limited impact of prompt coverage.

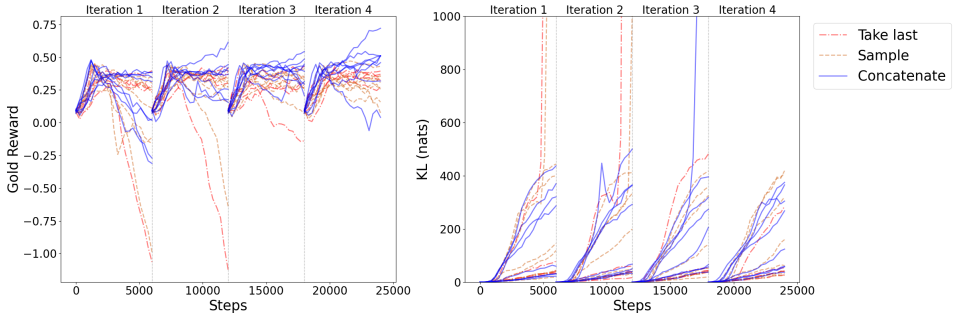


Figure 13: Gold score and KL of individual seeds across iterations for varying preference data combination methods.

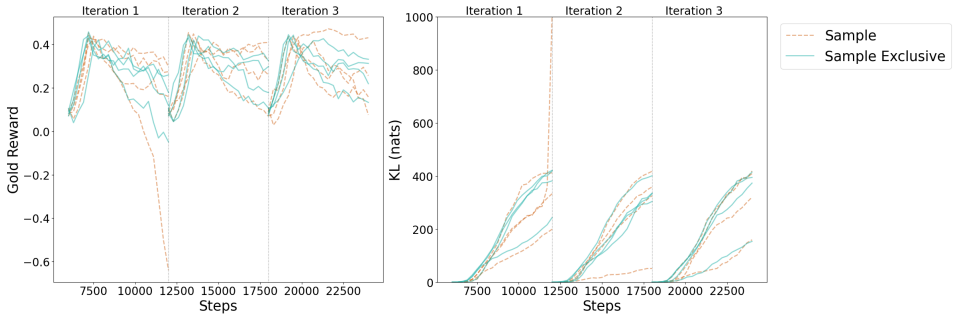


Figure 14: Gold score and KL of individual seeds across iterations comparing sampling with full coverage of the prompts vs random sampling.

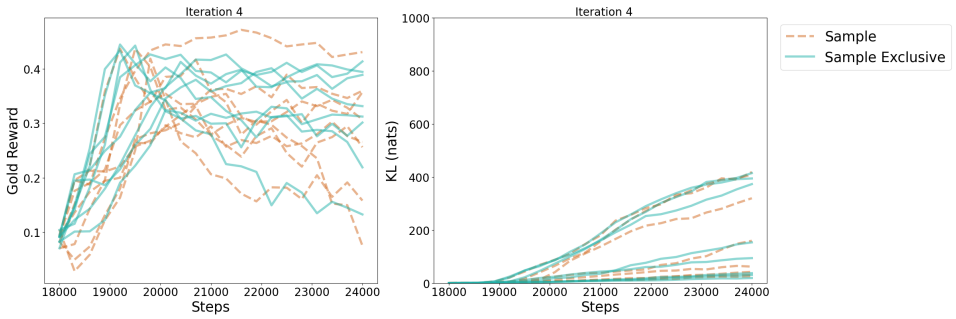


Figure 15: Gold score and KL of individual seeds in the fourth iteration comparing sampling with full coverage of the prompts vs random sampling.

the additional data only reproduces the maximum gold reward of the fourth *Concatenate* iteration and

delays overoptimisation relative to the $1\times$ baseline. Later in training the gold reward of the $4\times$ policy decreases while its proxy reward stays high, whereas the gold reward of the fourth *Concatenate* iteration declines much less. We attribute the difference to the coverage of the reward model training data. The $4\times$ reward model has only seen responses from π^{sft} and is therefore well calibrated close to π^{sft}, where its gold reward is highest, but has no data from the regions the policy reaches later in optimisation. Three quarters of the training data of the fourth *Concatenate* reward model are sampled from previously optimised policies, so its training distribution covers the response space that policy optimisation moves into, which makes it the better calibrated proxy in the regime where overoptimisation occurs.

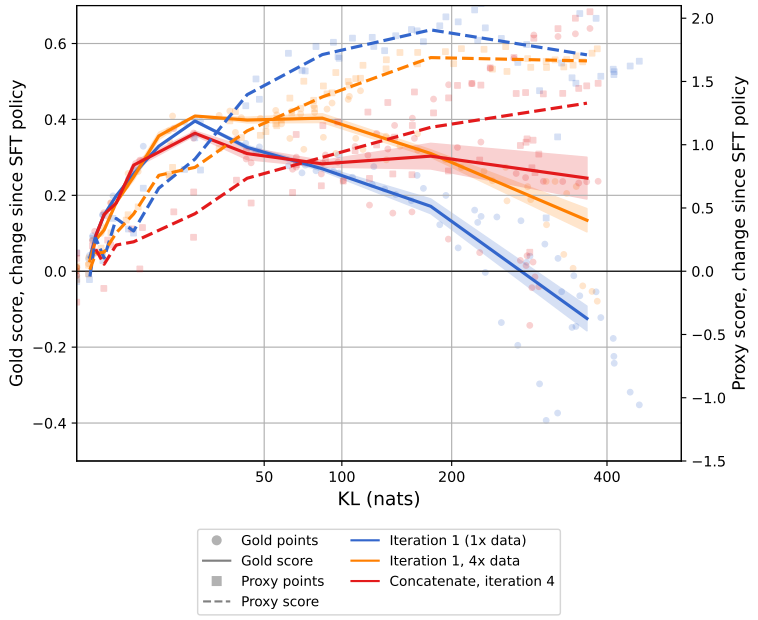


Figure 16: Data volume versus iteration (AlpacaFarm, three seeds). A single iteration trained on $4\times$ the preference data from π^{sft} reaches the maximum gold reward of the fourth *Concatenate* iteration but overoptimises at high KL, where its gold reward decreases while the proxy reward stays high. The gold reward of the fourth *Concatenate* iteration declines much less.

F.4 Additional results for reward model transfer

Here we provide additional results for methods addressing reward model transfer. Figures 17 and 18 show the individual training seeds of the methods across iterations.

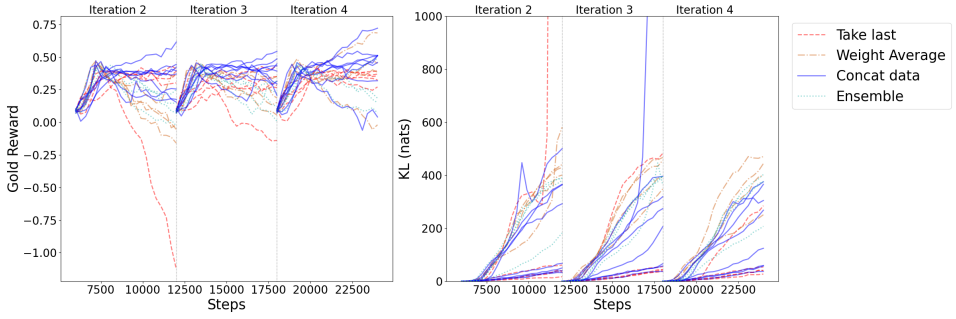


Figure 17: Gold score and KL of individual seeds across iterations comparing reward function choices.

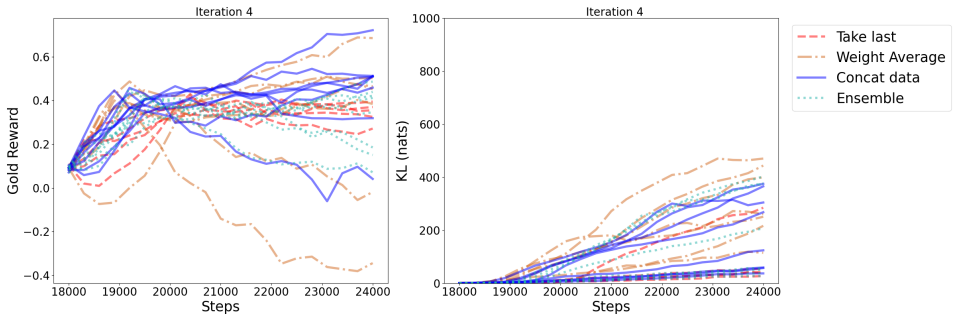


Figure 18: Gold score and KL of individual seeds in the fourth iteration comparing reward function choices.

F.5 Additional results for policy initialisation

Figure 19 shows every seed of the three policy-initialisation arms with *Concatenate* data at 410M. With *From SFT* every iteration starts from π^{sft} and a collapse in one iteration does not carry into the next. With *Continue* the median KL to π^{sft} grows in every iteration, and three of the eight seeds end the fourth iteration at or below the level of π^{sft}. With *LITI* every seed restarts within 67 nats of π^{sft} with a gold score between 0.13 and 0.43. Five seeds keep that gain, four of them ending the fourth iteration between 120 and 262 nats with gold scores of 0.37 to 0.52, while three seeds return to the high-KL region within a few hundred steps, lose the gain in every iteration, and one of them diverges to a KL above 9000 nats.

Whether a restart holds depends on the previous iteration. Of the fourteen restarts that follow an iteration ending with a gold score above 0.2, thirteen end above 0.2 as well, and of the ten that follow an iteration ending below it, eight end below it again. The interpolated policy also moves an order of magnitude faster than a policy reset from π^{sft}, gaining 34 nats in the first 900 steps at the median against 2 nats. In the failing restarts the proxy of the iteration's own reward model rises for the first few hundred steps and then falls together with the gold score, whereas under *Continue* the proxy rises by about one unit in each of the first two iterations and by about half a unit afterwards, while the gold stays close to the level of π^{sft}. The spread of gold scores across the evaluation prompts falls from 0.74 at the restart to 0.41 at the end of the failing runs, against 0.61 in the runs that hold and 0.72 for π^{sft}, so the responses become uniform across prompts. Such a policy yields near-zero policy gradients under a negative reward, since the score function scales with one minus the probability of the sampled tokens and identical samples leave the whitened advantages as noise, so even a reward model that scores these responses low cannot pull these runs back.

F.6 Continuing the policy with a larger reward model

Figure 20 shows the runs behind the *Continue* lines of Figure 5 against PPO steps. With the 70M reward model the first iteration loses 0.51 of its maximum gold score and ends at -0.10, and the continued policy starts iterations 2 to 4 at -0.10, 0.00 and 0.10, at or below the level of π^{sft}. With the 160M reward model the first iteration loses 0.13 and ends at 0.30, and the continued policy starts iterations 2 to 4 at 0.29, 0.35 and 0.50. The larger reward model also moves the policy less in every later iteration, with a median KL gain per iteration of 157, 112 and 59 nats against 178, 155 and 166 nats with the 70M reward model.

F.7 Larger policies

The policy-initialisation experiments of Section 5.4 were repeated with the 1.4B SFT policy of Coste et al. (2024), with the same 70M reward models, prompts, step budget and protocol as at 410M, on three of the eight seeds. The *Continue* arm at 1.4B uses *Take last* preference data. Figure 21 shows the proxy–gold progression of *Concatenate* with *From SFT* at both sizes.

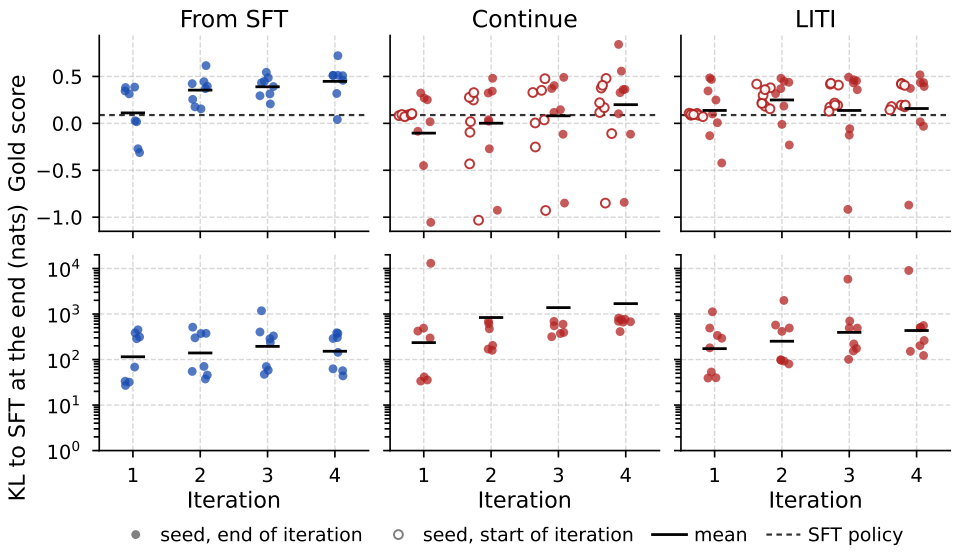


Figure 19: Gold score (top) and KL to π^{sft} (bottom) of every seed at the end of each iteration for the three policy-initialisation arms with *Concatenate* data at 410M, with the restart value of the warm-started arms as hollow markers and the mean as a bar. *LITI* restarts every seed near π^{sft} with a restored gold score, and the failing restarts follow collapsed iterations.

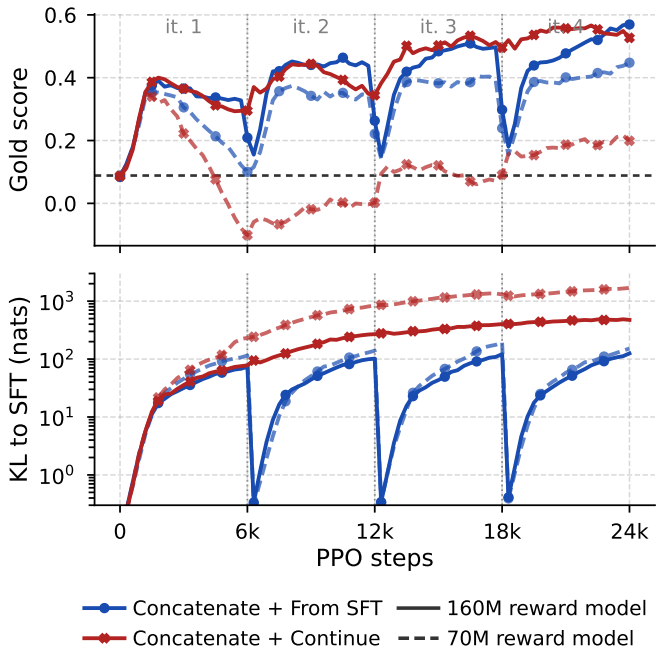


Figure 20: Continuing the policy with 70M (dashed) and 160M (solid) reward models on AlpacaFarm with the 410M policy. Gold score (top) and KL to π^{sft} (bottom) over four iterations for *Concatenate* with *From SFT* and with *Continue*, seed means. With the 160M reward model the first iteration keeps most of its gain and the continued policy drifts less in every later iteration.

F.8 TL;DR SUMMARISATION

To test whether reward models also become more robust across iterations on a second task, we repeat the iterated protocol on TL;DR summarisation (Stiennon et al., 2020) with a `pythia-410m` policy fine-tuned on the TL;DR SFT split, 70M reward models, 1000 prompts and 6000 PPO steps

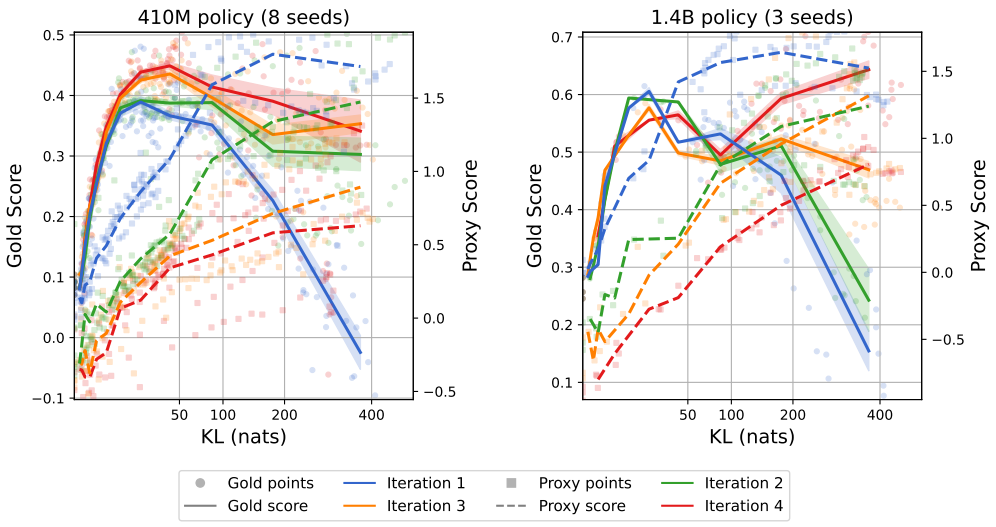


Figure 21: Proxy–gold progression of *Concatenate* with *From SFT* at 410M (8 seeds) and 1.4B (3 seeds), in the style of Figure 4a. At 1.4B the first two iterations collapse at high KL and the gap closes from the third.

per iteration, and 8 seeds. Preference labels and gold scores are provided by the 6.9B reward model `cleanrl/EleutherAI_pythia-6.9b-deduped__reward__tldr`. Compared to AlpacaFarm, the PPO learning rate is halved to 5×10^{-7} and policy optimisation is stopped once the KL to π^{sft} exceeds 300 nats, since at the AlpacaFarm learning rate both proxy and gold reward collapse within the first iteration. Figure 22 shows gold and proxy reward over four iterations for *Concatenate* and *Take last* with *From SFT* initialisation, and for *Take last* with *Continue*. We observe that, as on AlpacaFarm, the gap between the proxy and the gold reward decreases across iterations. The proxy reward rises much less after the first iteration, and for *Concatenate* both the rise of the proxy reward and the KL reached before the 300-nat cap decrease with every iteration. The gold reward, however, decreases within every iteration irrespective of the reward model, which we attribute to the optimisation setup on this task rather than to the reward models and leave to future work. Our focus here is the overoptimisation of the proxy, which is reduced across iterations on TL;DR as on AlpacaFarm.

F.9 Investigating entropy collapse

In Figure 23 we show the policy entropy throughout training for the *Continue* policy initialisation method. The performance collapse extends beyond a simple entropy collapse, suggesting that the policy is exploiting weaknesses in the proxy reward model that cannot be corrected in subsequent iterations.

F.10 On training stability across seeds and iterations

As is common with RL fine-tuning, we observed variance across random seeds. To mitigate this, we have performed training with 8 random seeds (significantly more than what is standard in the literature) and report the average performance and standard errors. While we focused on the effect of different methods on overoptimisation, we also observed that the design choices we evaluate, particularly those that reduce overoptimisation, tend to lead to more stable training. For instance, with *Concatenate* data, *From SFT* initialisation showed lower variance across seeds than *Continue* and *LITI*, suggesting improved stability. Please find a summary of these results in Table 7.

F.11 Overoptimisation analysis

Here we perform an additional analysis of response length and repetitiveness, two specific modes of overoptimisation that have been observed in the literature. In Table 8, we observe that the two

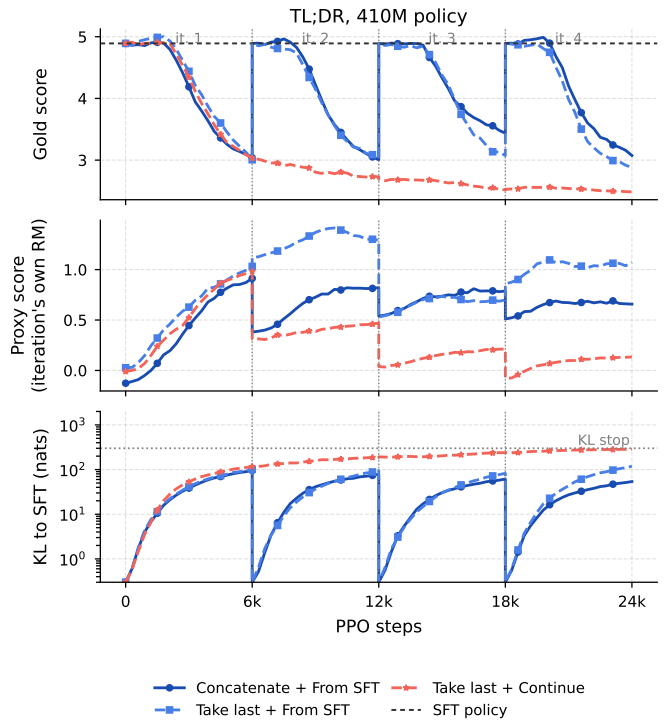
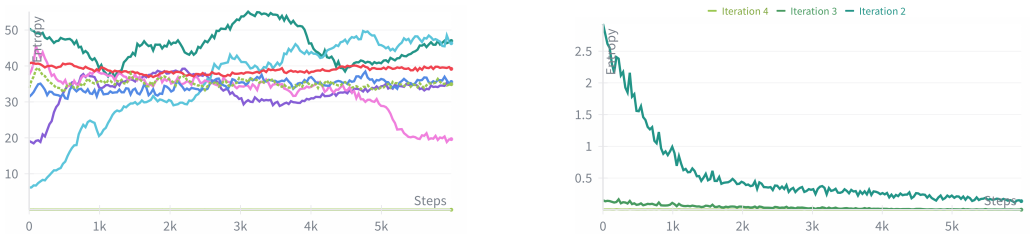


Figure 22: TL;DR summarisation, 410M policy, 8 seeds. Gold reward (top), the proxy reward of each iteration's own reward model (middle) and KL to π^{sft} (bottom) over four iterations. The gap between the proxy and the gold reward decreases across iterations, and for *Concatenate* so does the KL reached before the cap, while the gold reward falls to the same level in each iteration. Seeds stopped by the KL cap hold their last value.



(a) All seeds final iteration.

(b) Entropy collapse across iterations.

Figure 23: We plot the entropy of the policy against training steps for the *Continue* policy initialisation strategy. In the final iteration only one out of eight seeds has collapsed.

methods which continue PPO from the previous policy, with *Sample* and with *Concatenate*, and which exhibit the greatest degree of overoptimisation, produce policies with substantially higher trigram repetition, a lower proportion of unique words, and reduced vocabulary entropy. These patterns are well-known indicators of overoptimisation. The *Take last* row of Table 8 covers seven of its eight seeds. Figure 24 further shows that these indicators of overoptimisation consistently provide a stronger signal as iterations proceed.

G AN EXAMPLE OF IDIOSYNCRATIC BEHAVIOUR

Below we compare example responses from the *Continue* and *From SFT* policy initialisation methods at the start of training and at the end of the fourth iteration. The response at the end of training

Table 7: Mean and standard error across seeds at the end of the fourth iteration.

Method	Mean	Standard Error
Take last	0.3539	0.0381
Sample	0.2761	0.0408
Concatenate	0.4477	0.0698
Ensemble	0.3136	0.0550
Worst-Case Optimisation	0.2942	0.0486
Weight Average	0.2546	0.1265
Concatenate + Continue	0.1991	0.1793
Concatenate + LITI	0.1581	0.1635
Sample + Continue	−0.0632	0.1128

Table 8: Length and diversity metrics of policies in the final iteration.

Method	Gold score	Length (Words)	Trigram Repetition	Unique Ratio	Entropy
Concatenate + Continue	0.199	202.3	0.5613	0.2607	3.199
Sample + Continue	−0.063	180.6	0.5852	0.2660	2.827
Weight Average	0.255	204.9	0.3638	0.3027	4.629
Ensemble	0.314	165.7	0.3236	0.3704	4.638
Concatenate	0.448	187.1	0.4285	0.3098	4.231
Sample	0.276	162.5	0.3124	0.3835	4.612
Take last	0.357	160.0	0.2914	0.3939	4.614

obtained with *Continue* initialisation is notably more repetitive indicating severe overoptimisation. On the other hand, while still slightly repetitive, the *From SFT* initialisation results in a more sensible and less idiosyncratic response.

Continue Initialisation

Instruction:
"Given the following input, construct a creative story.
A magic bow and arrow"

Response from π^{sft}:
"The Arrow of time was a powerful magical weapon that could grant the user special abilities. It could grant them the ability to change shape, fly, and even beamed down with sound. The Arrow of time could also be used to travel to distant lands, granting the user access to magical items and secrets."

Response at the end of iteration 4:
"The story demonstrates the power and ability the ability to use the creativity and the practice in the practice and the practice in the creativity. This example demonstrates the ability to practice the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates the practice in the practice in the practice in the practice. This demonstrates"

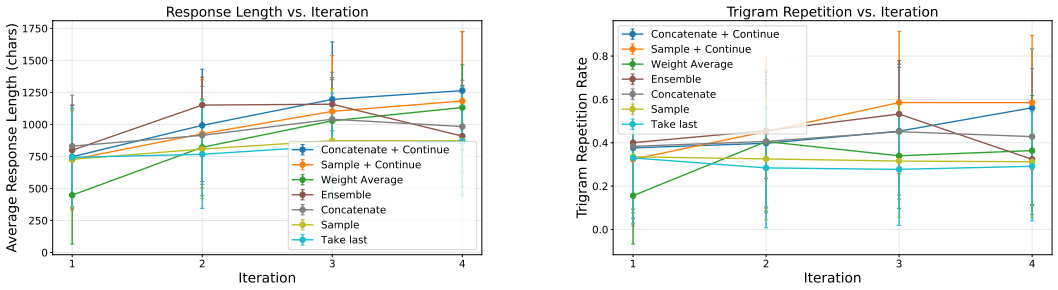


Figure 24: Response length and trigram repetition across iterations.

Concatenate and *From SFT* Initialisation

Instruction:
"Given the following input, construct a creative story.
A magic bow and arrow"

Response from π^{sft}:
"The Arrow of time was a powerful magical weapon that could grant the user special abilities. It could grant them the ability to change shape, fly, and even beamed down with sound. The Arrow of time could also be used to travel to distant lands, granting the user access to magical items and secrets."

Response at the end of iteration 4:
"A young man named Tom had always wanted to explore the world, but he was scared of the unknown. He was determined to learn to use the skills and knowledge he had acquired in the outdoors and to create his own special abilities and abilities. He set off on an adventurous journey and soon found himself on a magical journey to the far away and mystical world of the Grimm. With the help of a mysterious young woman and a powerful and powerful magic, Tom learned to use the bow and arrow and to create his own unique and powerful abilities. He learned to use the skill and knowledge he had gained in the outdoors and to use it to help protect and help others. This story will inspire and engage readers and will help them to explore and develop their creativity and knowledge in the world of the imagination."