Reward Model Overoptimisation in Iterated RLHF

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

Robert Kirk
UK AI Security Institute

Mirco Musolesi
UCL Centre for Artificial Intelligence
Department of Computer Science
University College London
Department of Computer Science and Engineering
University of Bologna

Abstract

Reinforcement learning from human feedback (RLHF) aligns large language models with human preferences but often suffers from reward model overoptimisation, where models exploit quirks of the reward function rather than generalising. A common mitigation is *iterated RLHF*, which repeatedly retrains reward models with new feedback and re-optimises policies. Despite its growing use, the dynamics of overoptimisation in this setting remain unclear. We present the first systematic study of iterated RLHF, analysing how data transfer, reward choice, and policy initialisation affect outcomes. Using the AlpacaFarm benchmark, we find that overoptimisation decreases across iterations as reward models better approximate preferences, but performance gains plateau. Reinitialising from the base policy is robust yet constrains optimisation, while alternative strategies struggle to recover from early overoptimisation. These results provide practical guidance for more stable and generalisable RLHF pipelines.

1 Introduction

Reinforcement learning from human feedback (RLHF) is the standard method for aligning large language models with human preferences [19, 11, 1]. Yet RLHF suffers from reward model overoptimisation [4], where fine-tuned models exploit the reward function—scoring highly without truly reflecting human intent. This produces brittle policies that appear aligned in training but fail in deployment. Iterated RLHF seeks to address this by repeatedly collecting preferences on policy outputs, retraining the reward model, and fine-tuning the policy [1, 18]. Despite widespread industry adoption [19, 11, 1], it remains unclear whether iterated RLHF resolves overoptimisation, merely delays inevitable exploitation [5], or perpetuates new cycles of overoptimisation [14].

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

Our key contributions can be summarised as:

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: Evaluating the Evolving LLM Lifecycle.

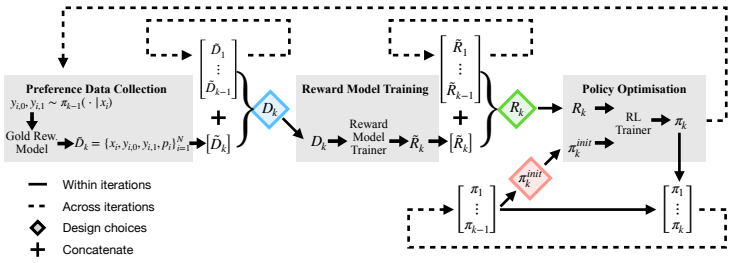


Figure 1: The Iterated RLHF framework performs multiple rounds of preference data collection, reward modelling, and policy optimisation. Our research reveals three design choices that dramatically impact performance: (1) how preference data is managed across iterations, (2) which reward function formulation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

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

Using a gold-standard reward model to simulate human labellers [2, 4] on the AlpacaFarm dataset [16] and working exclusively with open-source models, our experiments yield several key insights: Reward models become increasingly robust across iterations, leading to higher gold reward scores (Figure 2). Performance gains diminish after three iterations for most methods. Concatenating preference data across iterations dramatically outperforms other approaches. Small but persistent overoptimisation remains after four iterations regardless of design choices.

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 [8] by increasingly capable models.

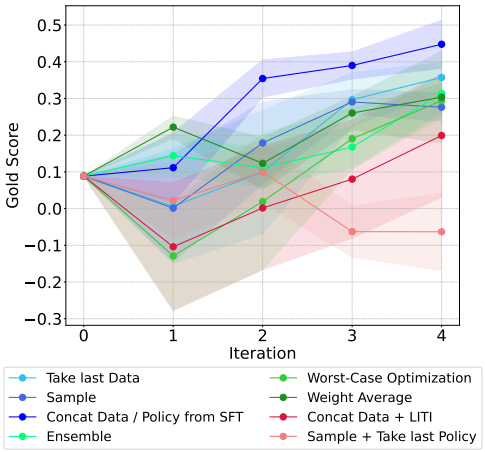


Figure 2: Our design choices for preference data management, reward function formulation, and policy initialisation significantly impact performance across iterations.

2 Iterated Reinforcement Learning from Human Feedback

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

2.1 Single-iteration RLHF

Preference data collection. We start with a supervised fine-tuned policy π^{sft} (a checkpoint) and use it to collect preference data. The dataset $\mathcal{D} = x_i, y_{i,0}, y_{i,1}, p_i i = 1^N$ contains prompts $x_i \in \mathcal{X}$, paired responses $yi, j \sim \pi^{sft}(\cdot|x_i)$ for $j \in 0, 1$, and preference labels p_i indicating whether $y_{i,0}$ is preferred over $y_{i,1}$. Following [2, 4], labels p_i are simulated using a gold reward model $R^\star$, larger than the proxy reward models, as a stand-in for human annotators.

Algorithm 1 Iterated RLHF (design choices highlighted)

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

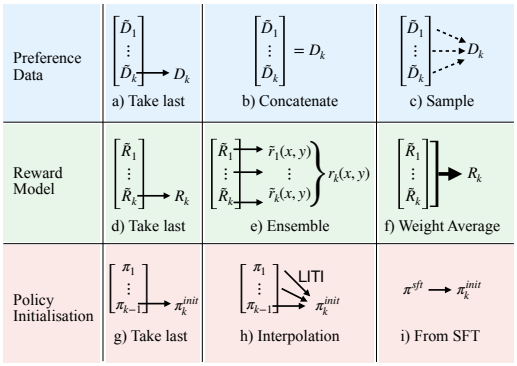


Figure 3: Design choices for Iterated RLHF (Algorithm 1). Options include how to combine preference data (latest only, concat, or sample), transfer reward models (last, ensemble, or weight averaged), and initialize policies (last, interpolate, or from SFT). These choices determine how learning signals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

Reward model training. The proxy reward model R_ϕ is initialised from model checkpoint R^{init}, with a randomly initialised prediction head, and subsequently trained by minimizing the cross-entropy loss on the preference dataset $\mathcal{D}$.

Policy optimisation. After initializing the policy π_θ from π^{sft} and training the proxy reward model R_ϕ, the policy is fine-tuned to optimize the reward model using Proximal Policy Optimization (PPO). A Kullback-Leibler (KL) divergence penalty is incorporated to prevent the policy from over-optimizing the proxy reward and diverging too much from its initial state, π^{sft}. A key limitation of collecting preferences only once is that reward models become poorly calibrated in high-reward regimes, which can lead to policy instability due to over-optimization. Furthermore, this optimization process causes π_θ to diverge from π^{sft}, generating responses outside the training data $\mathcal{D}$. This forces the reward model R_ϕ, which was trained on $\mathcal{D}$, to evaluate out-of-distribution inputs.

2.2 Iterated RLHF and design choices

Iterated Reinforcement Learning from Human Feedback (RLHF) addresses the divergence between $\pi_\theta(y|k)$ and $\pi^{sft}(y|x)$ by repeating the RLHF process. At each iteration k, the policy π_{k-1} generates new response pairs for a new preference dataset, $\tilde{\mathcal{D}}_k$, which is then combined with all previous datasets $[\tilde{\mathcal{D}}_1, ..., \tilde{\mathcal{D}}_k]$. This iterative process, which begins with π^{sft} and an initial reward model R^{init} with a random prediction head, helps align the reward model with in-distribution outputs and mitigate over-optimization by using only the latest policy to generate new data.

Combining preference data. Given k preference datasets from policies $\pi_1, \ldots, \pi_{k-1}$, we consider three ways to form the training set $\mathcal{D}_k$. In the simplest case, $\mathcal{D}_k = \tilde{\mathcal{D}}_k$ (Fig.3.a). At the other extreme, all datasets are concatenated, $\mathcal{D}k = \bigcup i = 1^k \tilde{\mathcal{D}}_i$ (Fig.3.b). A compromise keeps dataset size fixed by sampling subsets $\tilde{\mathcal{D}}_i$ and concatenating them (Fig. 3.c). The proxy reward model $\tilde{R}_k$ is then trained on $\mathcal{D}_k$, initialised from the same base model each iteration. Having trained the reward model, we now arrive at the second critical design choice of defining the reward function.

Combining reward models. The reward model is central to stable RLHF, its role compounding over iterations. Given proxy reward models $[\tilde{R}_1, \ldots, \tilde{R}_k]$, we must define a robust R_k for optimization, paralleling preference dataset combination. Options: use only the latest model $R_k = \tilde{R}_k$ (Fig. 3.d); ensemble all proxy models $R_k(x, y) = \frac{1}{k} \sum_{i=1}^k \tilde{R}_i(x, y)$ (Fig. 3.e) [2]; apply worst-case optimization $R_k(x, y) = \min_{i=1,\ldots,k} \tilde{R}_i(x, y)$. To reduce ensemble cost, consider weight-averaged reward models via task arithmetic [7], where $\tilde{R}_i$ has parameters $\tilde{\phi}_i$ and averaged model R_k is parameterized by

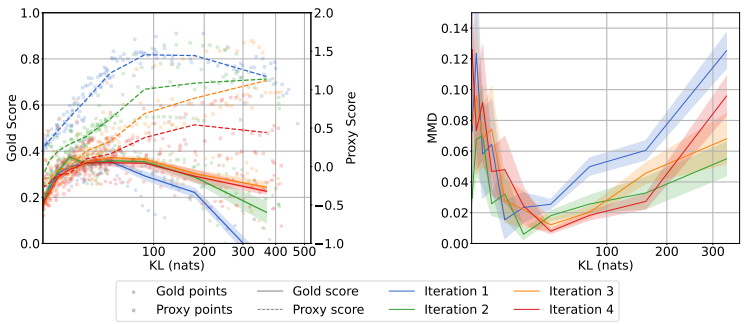
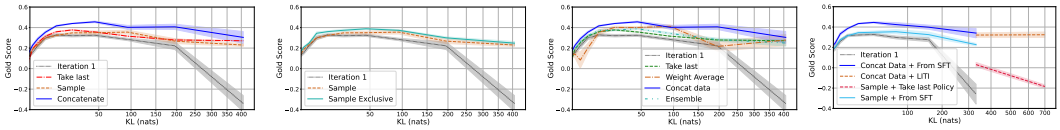


Figure 4: Progression of proxy and gold reward alignment across RLHF iterations with policy reinitialization from π^{sft} and concatenated preference data.



(a) Combining Data (b) Prompt Coverage (c) Combining RMs (d) Policy Initialisation

Figure 5: Iterated RLHF benefits most from scaling reward model training data.

$\phi_k = \frac{1}{k}\sum_{i=1}^{k}\tilde{\phi}_i$ (Fig. 3.f). Once R_k is defined, the final iteration step is policy optimization, raising the final design question on policy initialization.

Policy initialisation. The final design choice is policy initialisation, i.e., how $\pi^{init}k$ is chosen. A simple option is From SFT, where each iteration restarts from π^{sft} [1] (Fig. 3.i). As an alternative, we use linear interpolation towards initialisation (LITI) [13], inspired by WiSE-FT [17], defined as $\pi^{init}k = (1-\eta)\pi^{init}k - 1 + \eta\pi k - 1$ (Fig. 3.h), where η balances reliance on the previous policy. Setting $\eta = 1$ reduces to $\pi^{init}k = \pi k - 1$, continuing directly from the fine-tuned model. While this reuses prior computation, it risks entropy collapse and entrenchment of undesirable behaviours, which may be difficult to unlearn. Under LITI, optimisation is regularised by the KL divergence between π_k and π_k^{init}.

3 Experimental results

Our evaluation was performed on the AlpacaFarm dataset [3] with the Pythia model family. To measure overoptimisation we use both the mean proxy and gold reward, as well as the MMD metric to measure discrepancies between the distributions. For details on our training and evaulation setup we refer the reader to Appendix A.

Iterated RLHF can close the gap between proxy and gold reward functions. Across multiple iterations of RLHF, the proxy reward model becomes more robust and aligns better with the gold reward function, as evidenced by a decreasing gap between them. The KL-reward Pareto front also advances with each iteration, showing improved policy performance, though the gains diminish over time (Figure 4). The rate at which this gap closes varies significantly between different methods, underscoring the importance of specific design choices in the iterative process.

Combining preference data. The *Concatenate* strategy, which combines all preference data from each iteration, is the most effective way to improve reward model performance and prevent overoptimization (Figure 3.b). This method consistently outperforms other approaches like *Take last* and *Sample* by leveraging a larger training dataset, proving that scaling data is crucial for robust reward models (Figure 5a).

No free lunch by merging reward models. Figure 5c shows that approaches for combining reward models, like *Weight Average* and *Ensemble*, achieve performance comparable to the data concatenation method, especially in early KL regions. While these methods offer computational efficiency, they do not show significant performance differences, suggesting that their selection should be based on computational cost and scalability. We also evaluate how scaling the reward model size affects these methods in Appendix E.4.

Policy initialisation from SFT is the most robust. Comparing policy initialisation methods we find that initialisating from the sft checkpoint at each iteration is consistently robust (Figure 5d). Additionally, we observe that policies are hard to recover once they have overoptimised (Figure 6). Finally, we have observed that policy interpolation tends to work better with increasing parameter size of the reward model.

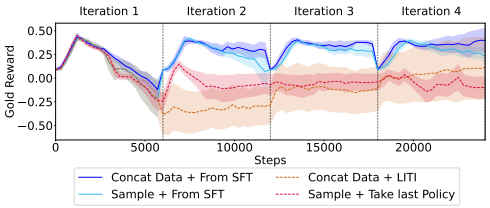


Figure 6: Policy Initialisation Across Iterations

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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 Evaluating overoptimisation in Iterated RLHF

Here we detail our evaluation setup, emphasizing the quantification of overoptimisation and examining how its progression over iterations is influenced by different design choices.

A.1 Training setup

Our evaluation setup follows extensive prior works that study overoptimisation in the single iteration RLHF in a controlled and simulated manner [2, 4]. Similarly to [2] we use instructions from the *AlpacaFarm* dataset [3] for reward model training and policy optimisation. The preference data $\tilde{\mathcal{D}}_k$ collected at each iteration contains preferences for a subset of 1000 instructions in the preference split of AlpacaFarm. Preference labels p_i are simulated with the 7 billion parameter `Reward-Model-AlpacaFarm-Human` [3], which is also used by [2]. It is worth noting again the significant difference in parameter sizes between the proxy reward models and the gold reward model, justifying the use of the gold reward model as a proxy for human labellers. Similarly to [2], to obtain π^{sft}, we performed supervised fine-tuning on the `pythia-410m` model[1] on the AlpacaFarm SFT split. We chose `pythia-410m` as it achieves an appropriate balance between computational cost and experimental rigour for our investigation. The authors in [4] also found that policy size did not affect the shape of the overoptimisation curve in their setting, further justifying this choice of policy. We initialise proxy reward models $\tilde{R}_k$ from the huggingface checkpoint `tlc4418/pythia_70m_sft` provided by [2][2], as well as the larger `pythia-160m`, with a randomly initialised prediction head [2]. We train reward models for 5 epochs with a learning rate of 1×10^{-5} [2]. For policy optimisation, we perform 6000 steps of PPO on the unlabelled split of AlpacaFarm. The learning rate is set to 1×10^{-6} and a constant KL penalty of 1×10^{-4} is used. The full specifications of the hyperparameters for reward model training and policy optimisation, and the prompt format are given in Appendix C.

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

A.2 Measuring reward model overoptimisation with the Maximum Mean Discrepancy

The standard methodology for investigating reward model overoptimisation is to compare the achieved mean reward on the proxy and gold reward functions on a hold-out set [2, 10, 4]. However, this analysis does not capture discrepancies in the long-tail, i.e., in the high-reward regime, that have a larger impact on the policy optimisation. In this work we propose to compare two reward models based on their distributions of rewards. To this end, we evaluate the policy and reward models on the 2000 unseen instructions contained in the validation split of AlpacaFarm at every 300 steps during policy optimisation. Our approach to measuring differences between reward functions consists of two steps, the first of which is a standardisation that ensures reward functions that lead to the same ordering of policies when optimised are treated as equal (see Apendix B). In the second step, we use the maximum mean discrepancy (MMD) [6] 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 $R^\star$. We now justify the validity of this method.

Formally, our goal is to compare any two reward functions R_{ϕ_1} and R_{ϕ_2}. As the first step, we scale both reward functions to have mean zero and variance one. This ensures that reward functions, which differ only by an affine transformation, are treated as equal to one another after scaling. For details about this result, please refer to Appendix B. This is desirable since affine transformations do not affect the ordering over policies induced by the original and transformed reward functions when they are optimised [15].

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.

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

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

Given that prompts are identically and independently distributed $x_i \overset{i.i.d.}{\sim} \rho$ and $y_i \sim \pi_\theta(\cdot|x_i)$, we obtain that the observed rewards $r_i = R_\phi(x_i, y_i)$ are i.i.d samples (details in Appendix B). As a consequence, we can rely on the Maximum Mean Discrepancy (MMD) to measure the discrepancy between distributions of observed rewards from R_{ϕ_1} and R_{ϕ_2}. The MMD compares two distributions based on their distances in the feature space determined by the chosen kernel. It is known for its strong theoretical guarantees, and it is commonly used in the two sample testing literature [6]. We use the popular squared exponential kernel.

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

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

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

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

B Proofs

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

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

$$\begin{aligned}R_2' &= \frac{R_2 - \mu_2}{\sigma_2} && (1)\\ &= \frac{R_2 - b'}{a'} && (2)\\ &= \frac{a' R_1' + b' - b'}{a'} && (3)\\ &= R_1'. && (4)\end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

$$P(Y_i, Y_j) = \left(\int P(Y_i \mid X_i) P(X_i) \, dX_i \right) \times \left(\int P(Y_j \mid X_j) P(X_j) \, dX_j \right). \tag{5}$$

$$\tag{6}$$

This shows that

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

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

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

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

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

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

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

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

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

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

C Additional experimental details

C.1 Hyperparameters

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

Table 1: SFT hyperparameters.

PARAMETER	VALUE
LEARNING RATE	$8e-6$
EPOCHS	3
BATCH SIZE	4

Table 2: RM hyperparameters.

PARAMETER	VALUE
LEARNING RATE	$1e-5$
EPOCHS	5
BATCH SIZE	32

C.2 Dataset

We use the instructions and inputs contained in the popular alpacafarm dataset [3, 16]. The entire dataset contains $52,000$ samples split into "sft" (10k), "preference" (20k), "unlabled" (20k), and "val" (2k). We use the "val" split strictly only for validation. The instructions for the reward model training are sampled from the "preference" split and the instructions for PPO are sampled from the "unlabled" split.

C.3 Prompt format

We follow the prompt format used in [2, 9], which is that of the v2 format used in Open Assitant. It uses special tokens `<|prompter|>` and `<|assistant|>`, and is consistent with the `GPTNeoXTokenizer` class.

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

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

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

C.4 Computational setup and cost

All experiments were run on a single Nvidia A100. Running the full pipeline consisting of all 3 RLHF steps for 4 iterations takes approximately 35 hours per seed and configuration. Subsequently labelling the results with the 7B gold reward model takes approximately 18h when using an evaluation set of size 2000 and evaluating every 300 steps.

D Iterated RLHF and Performative Prediction

We note that the framework of performative prediction applies to our setting. In fact, when performing iterated RLHF we are simulating performative prediction or more specifically a version of strategic

Table 3: PPO hyperparameters.

PARAMETER	VALUE
LEARNING RATE	$1e-6$
COSINE ANNEALING SCHEDULER	$1e-7$
PPO STEPS	6000
BATCH SIZE	32
NUMBER OF ROLLOUTS	256
CHUNK SIZE	32
CLIPPING RANGE & VALUE	0.2
GAE LAMBDA	0.95

Table 4: Generation hyperparameters.

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

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

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

classification. We have that a reward model R_ϕ induces a potentially different distribution $\mathcal{D}(\phi)$ over instances (x, y) where continuations y are obtained from the policy π_θ optimised for R_ϕ, which yields that a reward model $R_{\phi_{PO}}$ is performatively optimal if $\phi_{PO} = \arg\min_\phi \mathbb{E}_{(x,y)\sim\mathcal{D}(\phi)} \ell((x, y, \phi))$.

Furthermore, [12] call a model $R_{\phi_{PS}}$ performatively stable if

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

Intuitively, retraining a performatively stable reward model after optimizing against it will yield the same reward model. As such the reward model would not be over-optimised and still perform optimally on its induced distribution. In their Theorem 3.5 the authors of [12] provide 3 conditions under which the reward model obtained from repeated iterations of RLHF converges to a unique performatively stable reward model at a linear rate. We require the loss to be β-jointly smooth and γ-strongly convex, and the map $\mathcal{D}(\cdot)$ from reward model parameters to the distribution of prompt continuation pairs to be ϵ-sensitive [12]. Since as part of the map $\mathcal{D}(\cdot)$ the policy is optimised with PPO, where small changes in the reward model can lead to significant changes in the optimal policy, this mapping is generally not ϵ-sensitive. As a consequence, linear convergence is not guaranteed. Note, that we may still aim for close to linear convergence by making adjustments to satisfy the stated conditions.

E Additional results

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

Here we provide additional experimental results for taking the last preference dataset and sampling the preference datasets with equal proportion. In terms of the rate at which the gap between proxy and gold reward functions is reduced over iterations, the sampling strategy (see Figure 7) falls in between concatenating all preference data and taking only the last dataset (see Figure 8).

E.2 Additional results for combining preference data

In Figure 9 we provide the individual seeds for methods combining preference data across all iterations and in Figures 10 and 11 we provide the results for the sampling strategies. Figure 12 shows the MMD across iterations when only using the most recent preference dataset.

E.3 Additional results for reward model transfer

Here we provide additional results for methods addressing reward model transfer. Figure 13 and 14 show the individual training seeds of the methods across iterations.

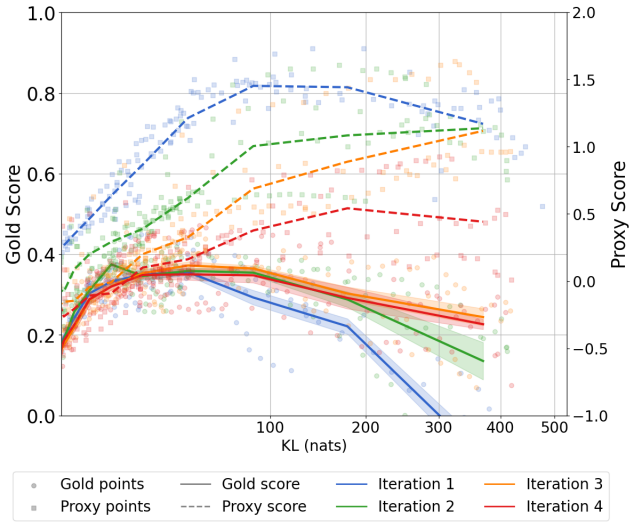


Figure 7: The gap between gold and proxy reward function when sampling from all preferences dataset equally to form the reward model training data.

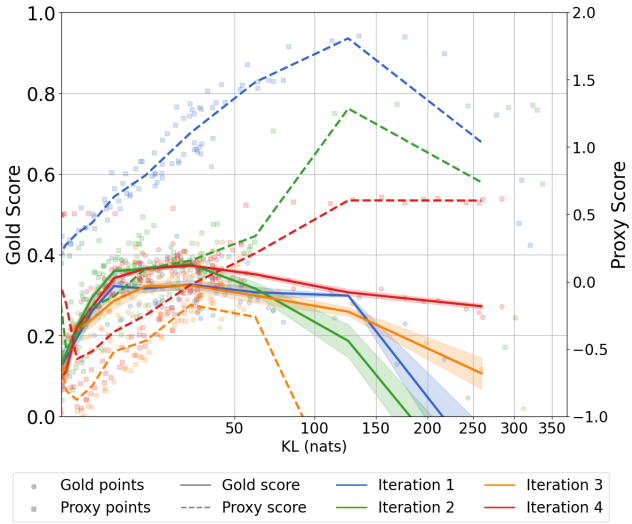


Figure 8: The gap between gold and proxy reward function when only taking the last preferences dataset for reward model training.

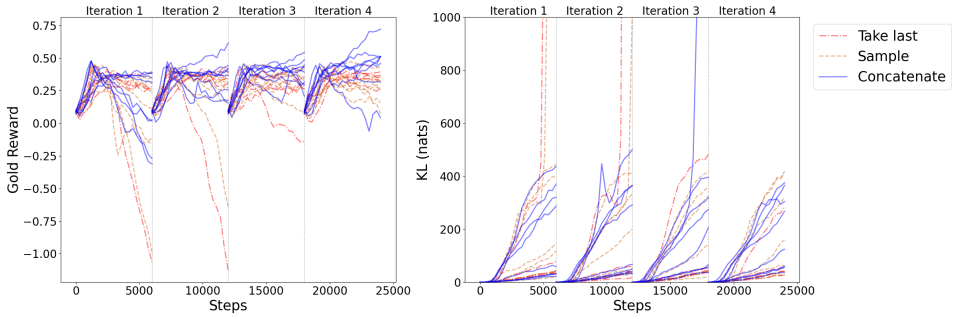


Figure 9: Gold score and KL of individual seeds across iterations for varying preference data combination methods.

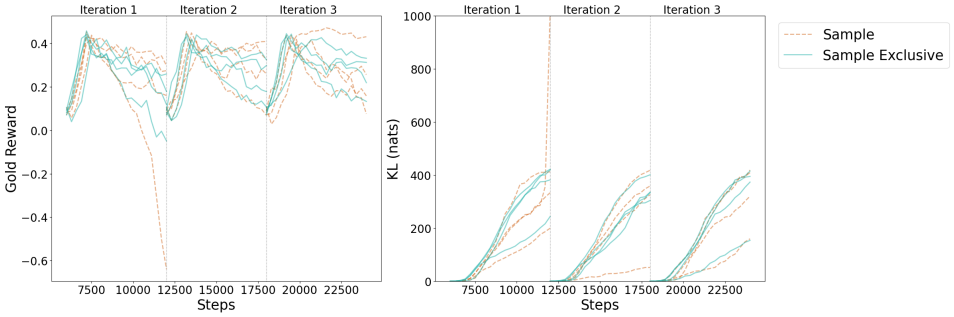


Figure 10: Gold score and KL of individual seeds across iterations comparing sampling with full coverage of the prompts vs random sampling.

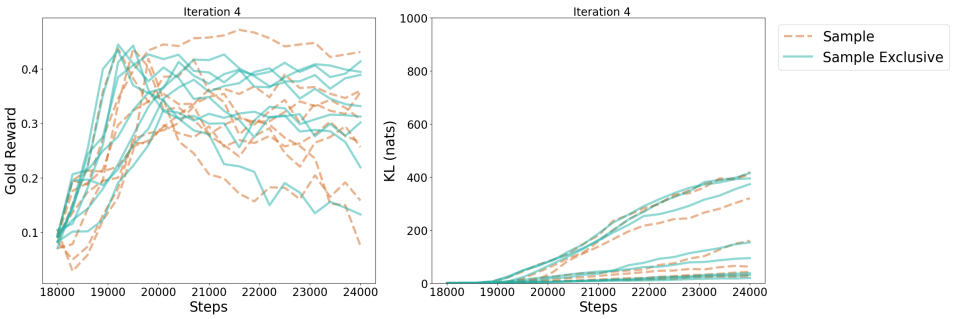


Figure 11: Gold score and KL of individual seeds in the fourth iteration comparing sampling with full coverage of the prompts vs random sampling.

E.4 Results for reward model scaling.

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 15, performance differences between the 70M and 160M reward models are most pronounced for *Ensemble* and *Worst-Case Optimisation*, with both methods substantially improving at the larger scale and approaching the performance of the data concatenation baseline by the fourth iteration. This suggests that while reward model combination methods did not match the effectiveness of preference data concatenation at smaller scales, their potential is unlocked with more expressive reward models. These results highlight that design choices affecting reward model size not only influence individual model accuracy but can significantly enhance the utility of design

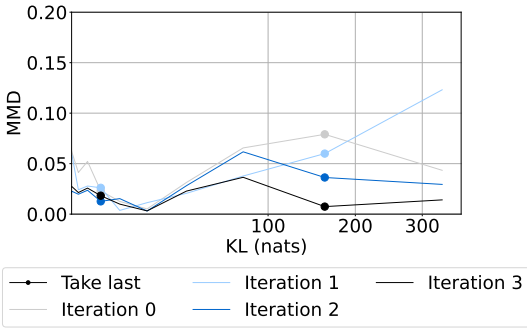


Figure 12: Taking the last preference dataset results in consistently low MMD, in the final iteration.

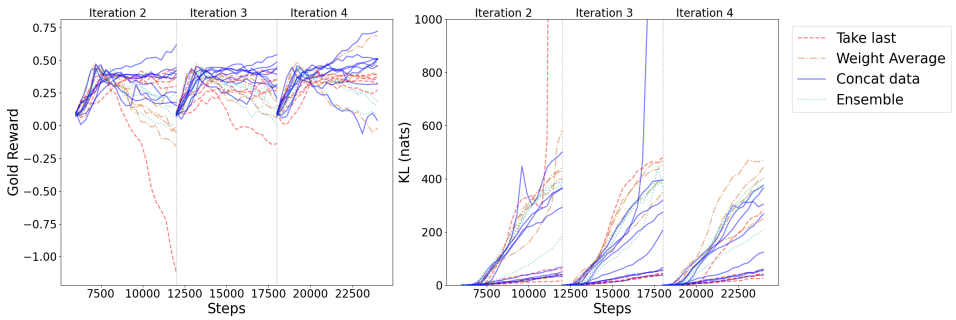


Figure 13: Gold score and KL of individual seeds across iterations comparing reward function choices.

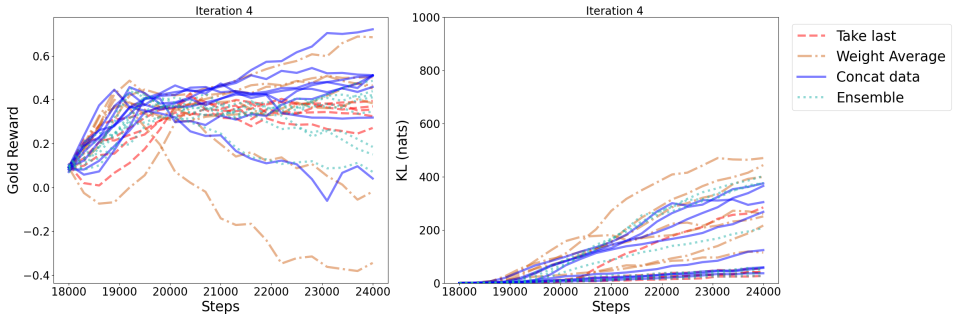


Figure 14: Gold score and KL of individual seeds in the fourth iteration comparing reward function choices.

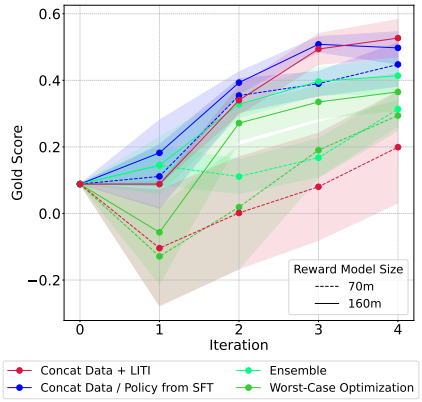


Figure 15: Impact of reward model size and design choices on gold score across iterations. Larger reward models (160M, solid lines) consistently outperform smaller ones (70M, dashed), with the biggest gains seen in *Ensemble* and *Worst-Case Optimisation*—suggesting that these strategies benefit most from increased reward model capacity. While *From SFT* remains the most stable across scales, *LITI* shows steady improvement, especially with the larger model.

choices combining reward models in iterated RLHF settings. We next examine if policy initialization strategies can complement reward modelling and preference aggregation to prevent overoptimisation.

E.5 Additional results for policy initialisation

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

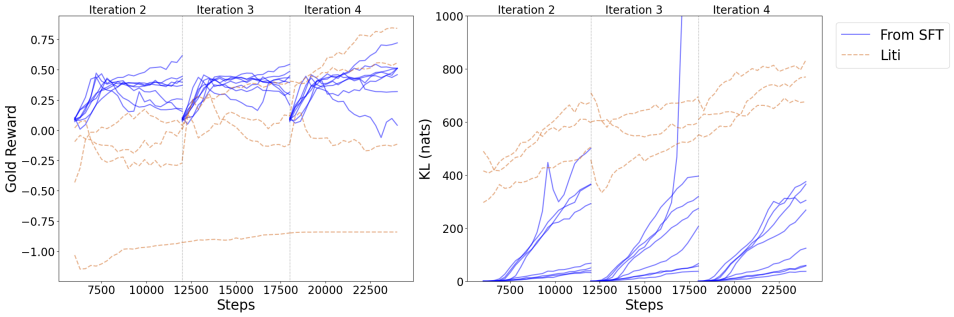


Figure 16: Gold score and KL of individual seeds across iterations comparing policy initialisation methods.

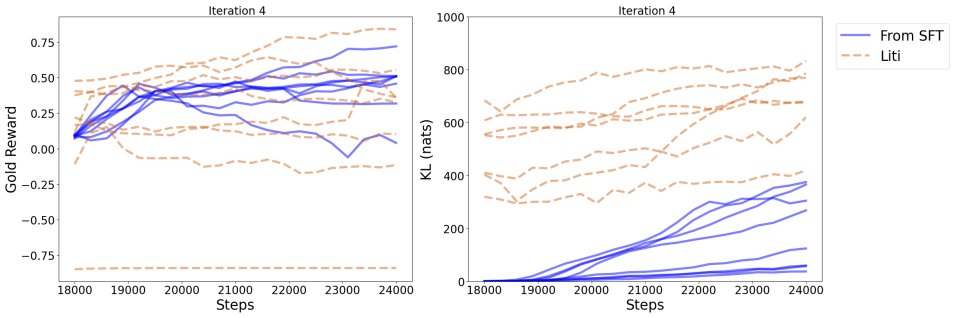


Figure 17: Gold score and KL of individual seeds in the final iteration comparing policy initialisation choices.