Preference Fine-Tuning of LLMs Should Leverage Suboptimal, On-Policy Data

Fahim Tajwar[1*], **Anikait Singh**[2*], **Archit Sharma**[2], **Rafael Rafailov**[2], **Jeff Schneider**[1], **Tengyang Xie**[4], **Stefano Ermon**[2], **Chelsea Finn**[2] **and Aviral Kumar**[3]

[*]Equal contributions (ordered via coin-flip), [1]Carnegie Mellon University, [2]Stanford University, [3]Google DeepMind, [4]UW-Madison

Learning from preference labels plays a crucial role in fine-tuning large language models. There are several distinct approaches for preference fine-tuning, including supervised learning, on-policy reinforcement learning (RL), and contrastive learning. Different methods come with different implementation tradeoffs and performance differences, and existing empirical findings present different conclusions, for instance, some results show that online RL is quite important to attain good fine-tuning results, while others find (offline) contrastive or even purely supervised methods sufficient. This raises a natural question: ***what kind of approaches are important for fine-tuning with preference data and why?*** In this paper, we answer this question by performing a rigorous analysis of a number of fine-tuning techniques on didactic and full-scale LLM problems. Our main finding is that, in general, approaches that use on-policy sampling or attempt to minimize the likelihood on certain responses (i.e., employ a negative gradient) outperform offline and maximum likelihood objectives. We conceptualize our insights and unify methods that use on-policy sampling or negative gradient under the notion of mode-seeking objectives, extended to categorical distributions. Mode-seeking objectives are able to alter probability mass on specific bins of a categorical distribution at a fast rate compared to maximum likelihood, allowing them to relocate masses across bins more effectively. Our analysis prescribes actionable insights for preference fine-tuning of LLMs and informs how data should be collected for maximal improvement.

1. Introduction

Pre-training endows a large language model (LLM) with knowledge about the world. Yet, it does not provide a lever to control responses from these models, especially when we want these solutions to optimize some task-dependent success criteria (e.g., align with human preferences, optimize correctness or compactness). To align LLMs with downstream success criteria, they are then fine-tuned with downstream objectives after pre-training. In this paper, we focus on fine-tuning problems that aim to optimize for binary preferences (from humans or other AI models). A plethora of methods have been proposed for this sort of fine-tuning, including supervised learning on filtered responses (Gulcehre et al., 2023), contrastive training (Rafailov et al., 2023), and on-policy reinforcement learning (RL) (Ouyang et al., 2022) on a reward function extracted from human preferences.

In theory, while all of these methods aim to discover identical optimal policies, achieving this in practice would require full data coverage and infinite computation. These requirements are not met in practice, and hence, the choice of the loss function and the optimization procedure affects performance. However, a lack of a clear understanding of different approaches, coupled with different tradeoffs in implementation, has resulted in substantial confusion: practitioners are unsure as to: **(1)** whether RL (Ouyang et al., 2022) is required at all, or contrastive approaches (Rafailov et al., 2023; Gheshlaghi Azar et al., 2023) or supervised fine-tuning are good enough; and **(2)** whether preference data should be collected with models in the loop (i.e., in an "on-policy" fashion) or not.

Our goal is to provide clarity on these questions by performing a rigorous study to understand the behavior of existing methods when optimizing for preferences. Our study operates under assumptions typical in preference fine-tuning, including the existence of an underlying ground truth reward function that explains the preference data. We study methods that train an LLM policy to optimize a surrogate loss given by the expected reward under a model of the reward function (learned from preference data) penalized by the KL-divergence between the policy and a reference policy.

Corresponding author(s): anikait@stanford.edu, ftajwar@andrew.cmu.edu

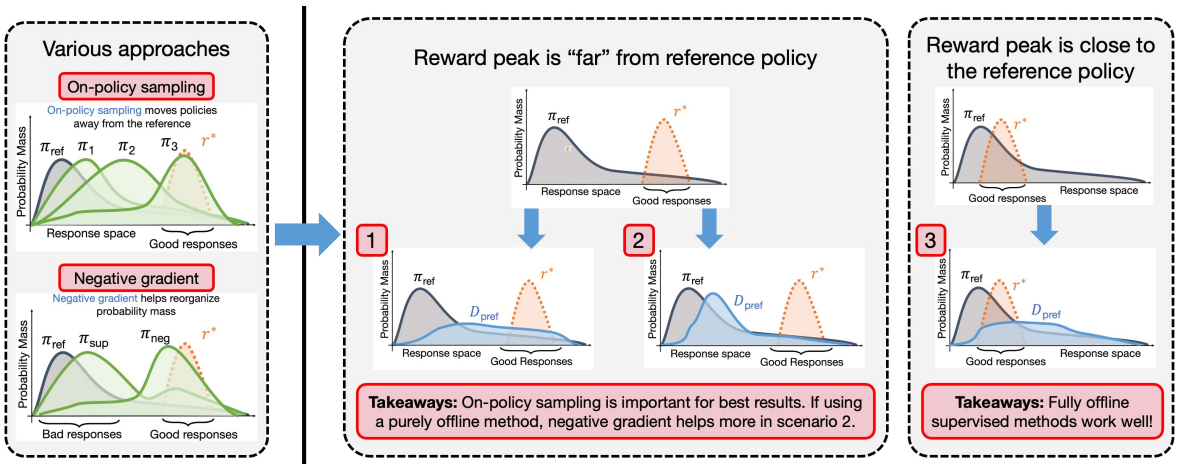

Figure 1: Left: an illustration of various fine-tuning techniques. On-policy sampling gradually shifts policy mass from π_{ref} to π_i, moving it towards the peak in the reward function indicated by r^*. Offline methods that employ a negative gradient push down the likelihood of bad responses under the learned policy, resulting in farther deviation of π_{neg} compared to methods that only maximize some sort of likelihood, π_{sup}. **Right: our key takeaways for practitioners:** when the peak of the reward function lies in the less likely regions of π_{ref}, on-policy sampling is generally beneficial. In conjunction, a negative gradient may be beneficial in some cases above (case 2). When r^* already lies in the high likelihood regions of π_{ref}, offline supervised methods can also work well. On-policy sampling or negative gradients may not be needed.

To answer the above questions, we develop an analysis framework consisting of didactic bandit problems, synthetic LLM problems, and full-scale LLM problems, constructed out of AlpacaFarm (Dubois et al., 2024) and UltraFeedback (Cui et al., 2023) benchmarks. We then study behaviors of different methods given coverage conditions and geometric relationships in the problem. ***Our main empirical observation*** is that algorithms that use on-policy RL or attempt to push-down likelihood on certain responses, i.e., utilize a negative gradient term as in contrastive objectives tend to outperform offline supervised objectives with no online sampling. We find that **using on-policy sampling and negative gradients are especially important when high-reward responses appear in less-likely regions of the reference policy distribution, and provide benefits complementary to each other.** In particular, we find that supervised objectives such as Pref-FT and Binary Feed-ME (Dubois et al., 2024) are not able to effectively move probability mass from low reward responses to high reward responses. On the other hand, sampling on-policy responses for training, contrastive learning, or employing both on-policy sampling and contrastive training can produce reward-maximizing policies.

We theoretically show that approaches that use on-policy RL or certain variants of contrastive training exhibit "mode-seeking" behavior, resulting in faster accumulation of probability mass on a subset of high-reward responses. This behavior is in contrast to typical "mode-covering" supervised objectives that attempt to increase likelihood on all high-reward responses, and as a result are unable to quickly increase probability mass enough on one subset of high-reward responses. We then compare the behavior of a representative mode-seeking objective, the reverse KL-divergence, with the mode-covering forward KL-divergence to formalize this behavior for categorical distributions. Conceptually, this ability to commit to a certain subset of high-reward responses enables algorithms with on-policy sampling or a negative gradient to perform better than maximum likelihood.

Our work presents several actionable takeaways for practitioners. First, we tie the performance of various methods to geometric conditions on the problem, that can inform practitioners which approach to use. Second, we observe a tradeoff between drawing more on-policy samples and performing more number of gradient steps with a different policy training objective. Understanding this tradeoff is useful for practitioners since on-policy sampling and training present different computational tradeoffs. Finally, since the performance of fine-tuning is intricately tied to the data composition, we study the effect of conditions on relative coverage of the preference data and the reference policy on performance, which could inform future preference data collection schemes.

2. Related Work

A dominant recipe for fine-tuning LLMs is to run supervised next token prediction ("supervised fine-tuning") on a dataset of high-quality responses to obtain a good policy initialization. This is followed by fine-tuning on a dataset of human preferences (Casper et al., 2023; Ouyang et al., 2022). This fine-tuning can use on-policy RL methods such as REINFORCE (Sutton et al., 1999) or PPO (Schulman et al., 2017) to maximize the predictions of a reward model obtained from the preference data, regularized with a KL constraint. Another approach (Dubois et al., 2024) performs supervised fine-tuning on the filtered set of preferred completions in the preference dataset. A different family of methods runs supervised learning on preferred responses iteratively such as ReST (Gulcehre et al., 2023), RWR (Hu et al., 2023), and SuperHF (Mukobi et al., 2023). Alternatively, methods such as DPO (Rafailov et al., 2023), IPO (Gheshlaghi Azar et al., 2023), SLiC-HF (Zhao et al., 2023), and KTO (ContextualAI, 2024) learn directly from human preferences, with no explicit reward model. Concurrent work also runs DPO iteratively (Yuan et al., 2024; Chen et al., 2024). These methods come with different tradeoffs necessitating a study to understand their behaviors.

Prior analysis work. To understand the effect of preference fine-tuning, prior work attempts to uncover its effect on network parameters for a certain set of tasks (Jain et al., 2023; Lee et al., 2024). Our analysis is complementary in that it studies conditions when different algorithms perform well, and is applicable to any downstream task. Kirk et al. (2023) study the contribution of RL fine-tuning on generalization to out-of-distribution prompts but this is complementary to our approach. Gao et al. (2022); Coste et al. (2023); Eisenstein et al. (2023) study reward over-optimization to better build reward models, which is complementary to the behavior of the policy optimization approach. Agarwal et al. (2023) develop a recipe that uses the mode-seeking KL divergence for knowledge distillation: this prior work is largely centered in the problem setting of distillation and does not study the optimization behavior of RL, contrastive, or supervised objectives. Perhaps closely related to our work is Singhal et al. (2023), which investigates the interplay between PPO and the composition of preference data, but this analysis is largely concentrated on studying the length bias of RL fine-tuning rather than developing insights into the behavior of fine-tuning algorithms. We do design didactic examples that use rewards dependent on length, but this is solely for analysis.

Concurrently, Ahmadian et al. (2024) show that REINFORCE may simply be enough for preference fine-tuning of LLMs and complex policy optimization methods such as PPO may not be needed. Our conclusions are mostly complementary, though we do observe that PPO is more robust to sample reuse than REINFORCE. Concurrently, Sharma et al. (2024) compares contrastive and supervised fine-tuning on LLM-generated data, but this work does not study the role of coverage or geometric conditions. Nevertheless their conclusions that various approaches perform similarly when the peak in the reward function (i.e., oracle AI preferences) aligns with the likely regions in the data (i.e., responses generated from the same AI model), thus providing evidence to support our findings.

3. Characterizing And Unifying Preference Fine-Tuning Methods

Typical preference fine-tuning methods use a variety of objectives including RL, maximum likelihood, and contrastive learning. While the huge number of fine-tuning methods inhibits us from empirically analyzing each of them, in this section we characterize a number of existing methods into different families based and subsequently study a representative member from each family.

3.1. Preliminaries and Notation

Typically, before training on preference data, a pre-trained model is fine-tuned on high-quality data from the task of interest via supervised fine-tuning (SFT), to obtain a "reference" model π_{ref}. Then, to fine-tune π_{ref} with human preferences, usually a preference dataset $\mathcal{D}_{\text{pref}} = \{\boldsymbol{x}^{(i)}, \boldsymbol{y}_w^{(i)}, \boldsymbol{y}_l^{(i)}\}$ is

collected, where $\boldsymbol{x}^{(i)}$ denotes a prompt and $\boldsymbol{y}_w^{(i)}, \boldsymbol{y}_l^{(i)}$ denote preferred and dispreferred responses. Given a preference dataset, most fine-tuning pipelines assume the existence of an underlying reward function $r^*(\boldsymbol{x}, \cdot)$. One popular framework for this is the Bradley-Terry (BT) model (Bradley and Terry, 1952), assuming that human preferences can be written as:

$$p^*(\boldsymbol{y}_1 \succ \boldsymbol{y}_2|\boldsymbol{x}) = \frac{e^{r^*(\boldsymbol{x},\boldsymbol{y}_1)}}{e^{r^*(\boldsymbol{x},\boldsymbol{y}_1)}+e^{r^*(\boldsymbol{x},\boldsymbol{y}_2)}} \tag{3.1}$$

Given this reward function r^*, preference fine-tuning aims to find the optimum of the reward r^*. While the ultimate goal of preference fine-tuning is to find the *unconstrained* optimum of the reward function, in practice, we often replace the reward function with a reward model. Since the reward model is erroneous, we apply KL-constraint to prevent exploitation in the reward model. To align our results with typical preference fine-tuning procedures, we will consider such a KL-constrained reward optimization as our fine-tuning goal:

$$\max_{\pi_\theta} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}},\boldsymbol{y}\sim\pi_\theta(\cdot|x)}[r^*(\boldsymbol{x},\boldsymbol{y})] - \beta\mathbb{D}_{\text{KL}}[\pi_\theta(\cdot|\boldsymbol{x})||\pi_{\text{ref}}(\cdot|\boldsymbol{x})] \quad \textbf{(Surogate fine-tuning goal)} \tag{3.2}$$

The regularizer, weighted by β, controls the deviation of π from π_{ref} under the reverse KL divergence.

Reward model training. In order to fine-tune an LLM policy $\pi_\theta(\boldsymbol{y}|\boldsymbol{x})$, Equation 3.1 provides a convenient way to learn a reward model either explicitly (i.e., by fitting a parametric reward model $r_\phi(\boldsymbol{x}, \boldsymbol{y})$) or implicitly (i.e., via direct preference optimization (DPO) Rafailov et al. (2023) or IPO (Gheshlaghi Azar et al., 2023), that re-purposes the log-likelihood $\log \pi_\theta(\boldsymbol{y}|\boldsymbol{x})$ of the policy to implicitly represent the reward $r_\phi(\boldsymbol{x}, \boldsymbol{y})$). Explicit reward models are trained using the following classification objective:

$$\max_{\phi} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\sim\mathcal{D}_{\text{pref}}} \left[\log \sigma\left(r_\phi(\boldsymbol{x},\boldsymbol{y}_w) - r_\phi(\boldsymbol{x},\boldsymbol{y}_l)\right)\right], \tag{3.3}$$

where σ is the logistic function. Contrastive learning objectives (Rafailov et al., 2023; Gheshlaghi Azar et al., 2023) repurpose Equation 3.3 by repurposing $r_\phi(\boldsymbol{x}, \boldsymbol{y})$ as:

$$r_\phi(\boldsymbol{x}, \boldsymbol{y}) = \log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) - \log \pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x}). \tag{3.4}$$

3.2. Characterizing Fine-Tuning Methods

With a reward model $r_\phi(\boldsymbol{x}, \boldsymbol{y})$, most fine-tuning approaches attempt to discover the policy $\pi_\theta(\boldsymbol{y}|\boldsymbol{x})$ which optimizes Equation 3.2 by using r_ϕ as a surrogate for r^*. Since we cannot empirically investigate all of these methods, we group them into different categories (summary shown in Table 1). In particular, we are interested in whether these methods employ:

1. ***on-policy sampling***: explicit sampling of new responses from the policy (e.g., PPO, REINFORCE) or purely learning from offline data (e.g., RWR, DPO, IPO)
2. ***on-policy sample reuse***: for only those approaches that perform on-policy sampling, whether the approach makes more than one gradient update on a given prompt-response $(\boldsymbol{x}, \boldsymbol{y})$ pair (e.g., exactly 1 update for REINFORCE, ≥ 1 for PPO, online RWR)
3. ***negative gradient***: whether the approach explicitly minimizes a loss that attempts to "push-down" likelihood on certain responses by multiplying the gradient of their likelihood with a negative coefficient (e.g., contrastive methods such as DPO; RL methods REINFORCE, PPO)

On-policy RL approaches such as PPO (Schulman et al., 2017) and REINFORCE (Williams, 1992) explicitly sample new responses from the current snapshot of the learned policy, $\boldsymbol{y}_i \sim \pi_\theta(\cdot|\boldsymbol{x}_i)$, score them under the reward model, and perform a policy gradient update on parameters θ, for example:

$$\theta' \leftarrow \theta - \eta\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}},\boldsymbol{y}\sim\pi_\theta(\cdot|\boldsymbol{x})}\left[\nabla_\theta \log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) \cdot \bar{r}_\phi(\boldsymbol{x}, \boldsymbol{y})\right] \qquad \text{(REINFORCE)}, \tag{3.5}$$

is the gradient update employed by REINFORCE, where $\bar{r}_\phi(\boldsymbol{x}, \boldsymbol{y})$ corresponds to a normalized estimate of the reward model's predictions over a batch of samples drawn from the policy. As we discuss in more detail in Appendix D.1), using a normalized reward estimate instead of directly the raw reward value helps reduce variance of the policy gradient estimate. High variance gradients slow down convergence and even sometimes lead to sub-optimal solutions in deep RL (Mei et al., 2022).

Due to the use of normalized reward estimates, policy gradient approaches behave distinctly from maximum likelihood supervised learning: a policy gradient update also updates the parameters θ in a direction that attempts to push down likelihood $\log \pi_\theta(\boldsymbol{y}'|\boldsymbol{x})$ for samples $\boldsymbol{y}'$ on which normalized reward $\bar{r}_\phi(\boldsymbol{x}, \boldsymbol{y}') < 0$. This means that on-policy RL approaches also have a **"negative gradient"**.

PPO differs from REINFORCE because it employs *sample reuse* in addition to on-policy sampling: unlike REINFORCE which only performs a single gradient update on a response sampled from the current policy, PPO utilizes a response for several policy updates. To prevent making updates on overly off-policy responses, responses are filtered by magnitude of the importance ratio between the current policy $\pi_\theta(\boldsymbol{y}|\boldsymbol{x})$ and the data collection policy, $\pi_{\theta_{\text{old}}}(\boldsymbol{y}|\boldsymbol{x})$.

Finally, we also remark that while on-policy methods do generate new rollouts from the policy, these responses are still scored by a reward model (and not the groundtruth reward function, i.e., humans). Since reward labels come from a reward model, on-policy preference fine-tuning approaches are instances of **offline model-based RL** (Yu et al., 2021, 2020; Kidambi et al., 2020) methods that run on-policy rollouts against a learned dynamics and reward model (due to the single step nature of preference fine-tuning, there is no dynamics model).

Fine-Tuning Approach	On-Policy Sampling	Sample Reuse	Negative Gradient
PPO	✓	✓	✓
REINFORCE	✓	×	✓
DPO, IPO, and variants	×	N/A	✓
Pref-FT, Binary FeedMe	×	N/A	×
offline RWR, offline Best-of-N	×	N/A	×
ReST, RWR, online Best-of-N	✓	✓	×

Table 1: **Grouping various fine-tuning methods** along the axes on-policy sampling, sample reuse, and negative gradient. Since offline methods do not collect on-policy data, the question of discarding or reusing on-policy samples is not applicable.

On-policy supervised approaches such as ReST (Gulcehre et al., 2023) and SuperHF (Mukobi et al., 2023) iteratively minimize a weighted maximum likelihood loss inspired Peters and Schaal (2007); Korbak et al. (2022). These methods sample responses from the model for a given prompt $\boldsymbol{x}_i$: $\boldsymbol{y}_i^1, \cdots, \boldsymbol{y}_i^N \sim \pi_\theta(\cdots|\boldsymbol{x}_i)$, then weight these responses by the exponentiated reward, $\exp(r_\phi(\boldsymbol{x}_i, \boldsymbol{y}_i^j)/\beta)$ as in the case of reward-weighted regression (RWR) or obtain the subset of K highest rewarding responses as in the case of ReST (Gulcehre et al., 2023) or Best-of-N. Finally these methods train via supervised next token prediction on these filtered or weighted responses. Given a weighting function, $F(\boldsymbol{x}_i, \boldsymbol{y}_i^j|\boldsymbol{y}_i^{0\cdots N})$ that maps a response $\boldsymbol{y}_i^j$ for a given prompt $\boldsymbol{x}_i$ to a scalar value conditioned on other responses $\boldsymbol{y}_i^k$ sampled from the model for the same prompt $\boldsymbol{x}$, these methods maximize:

$$\max_\theta \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}, \boldsymbol{y}^{0\cdots N}\sim\pi_{\theta^{\text{old}}}} \left[\log \pi_\theta(\boldsymbol{y}^i|\boldsymbol{x}) \cdot F(\boldsymbol{x}, \boldsymbol{y}^i|\boldsymbol{y}^{0\cdots N})\right].$$

These algorithms employ sample reuse because they operate in a **"batched" online fashion**: instead of performing ***exactly one*** gradient step on a given model sample; RWR, ReST, and SuperHF run more gradient updates, after which new samples are drawn. However, **since these methods only maximize likelihood (i.e., only positive multipliers), there is no negative gradient effect**.

Fully offline methods like DPO and IPO run contrastive training on the preference dataset $\mathcal{D}_{\text{pref}}$ without any on-policy sampling . These methods train using variants of Equation 3.3 (objective for IPO is shown Appendix D.2) combined with Equation 3.4 on responses $\boldsymbol{y}_w$ and $\boldsymbol{y}_l$ from the preference dataset $\mathcal{D}_{\text{pref}}$. Despite the absence of on-policy sampling, the contrastive loss between winning and losing responses explicitly attempts to reduce the likelihood ratio $\log \frac{\pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})}{\pi_{\text{ref}}(\boldsymbol{y}_l|\boldsymbol{x})}$ for $\boldsymbol{y}_l$. Another offline method is Pref-FT (Dubois et al., 2024) that runs supervised fine-tuning on preferred responses.

4. Research Questions and Analysis Setup

Our goal is to understand the behaviors of various procedures for fine-tuning language models. As discussed in the previous section, typically these methods differ along the use of on-policy sampling (with additional differences pertaining to sample reuse) and the presence of a negative gradient term, that penalizes likelihoods that the model assigns to certain responses. We build a setup to understand these differences empirically by answering the following questions:

Question 1: How and when does on-policy sampling improve over offline fine-tuning, even though on-policy samples are annotated by a reward model, which itself is learned from offline data? Is sample reuse important for on-policy methods?

Question 2: When does an explicit negative gradient term help the discovery of effective policies compared to maximum likelihood approaches such as distilling the Best-of-N policy?

Question 3: Does on-policy sampling offer complementary benefits to negative gradient, resulting in better performance with effective contrastive approaches (e.g., DPO)?

To gain practically useful and actionable insights, we must answer these questions in the context of coverage and geometric relations between the training data, reference policy, and the reward function. These relations affect the shape of the optimally fine-tuned policy and dictate the dynamics of various objectives under consideration. We consider specific conditions and relations that we discuss next.

4.1. Coverage Conditions and Geometric Relationships

The dynamics of the KL-constrained surrogate optimization problem (Equation 3.2) depends on the geometric alignment between the ground-truth reward function r^* and the reference policy initialization π_{ref}. In addition, when the surrogate reward model r_ϕ is learned from the preference data, the coverage of the preference data used to train this reward model relative to the reference policy π_{ref} also dictates the correctness of reward estimates, and hence controls the efficacy of the surrogate fine-tuning optimization. The performance of purely offline methods (e.g., offline best-of-N or contrastive methods such as offline DPO) that do not use a reward model also depends on the relative geometric alignment between r^* and π_{ref} (i.e., lower alignment would necessitate more deviation from the reference policy) and also on the relative coverage of preference data (i.e., the lower the coverage, the harder it is to discover high reward responses). To understand the efficacy of various methods, we consider multiple scenarios that differ along these two factors:

- **[C1]**: the geometric alignment between the ground-truth reward function r^* and the reference π_{ref}, that can be measured in terms of any probabilistic divergence $D(\pi_{\text{ref}}, \exp(r^*))$. This concept is analogous to that of a **"concentrability coefficient"** (Munos and Szepesvári, 2008).
- **[C2]**: the coverage of the preference data used to train the surrogate reward model r_ϕ relative to the reference policy π_{ref}, that can be measured in terms of the average density of the responses in the preference dataset under the reference policy initialization, π_{ref}.

Understanding the behavior of various approaches as a function of these factors will allow us to better understand the performance of various approaches on downstream fine-tuning in terms of problem geometry **[C1]** and statistical learning considerations **[C2]**.

4.2. Tasks and Datasets

We construct a variety of didactic and LLM tasks that allow us to gain intuition for performance of different methods under various scenarios grouped along relationships [C1] and [C2].

Didactic N-d bandit problems. Per Equation 3.2, we pose preference fine-tuning as a KL-regularized contextual bandit problem over contexts $\boldsymbol{x}$. Therefore, we develop a didactic N-dimensional contextual bandit problem. We use a set of tokens of size V of size 100. The context, $\boldsymbol{x}$, is a single discrete token from V. A response $\boldsymbol{a}$ is a sequence of $N = 10$ discrete tokens from V. We primarily study the effect of geometric relationship [C1] and assume that the reward function is known exactly, therefore not accounting for data coverage and reward model training. We consider two reward functions that differ in their relative geometry relative to the reference policy, as shown in Figure 2. Specifically the difference lies in how perfectly the optimum of the reward function aligns with the high density regions of the reference distribution. The optimum of the reward function $\boldsymbol{R}_1$ is located in low likelihood regions of the reference, whereas the optimum of $\boldsymbol{R}_2$ is roughly aligned with the mode of the reference. We hypothesize that on-policy sampling will be crucial to optimize reward function $\boldsymbol{R}_1$, whereas offline or maximum likelihood methods could be sufficient for $\boldsymbol{R}_2$.

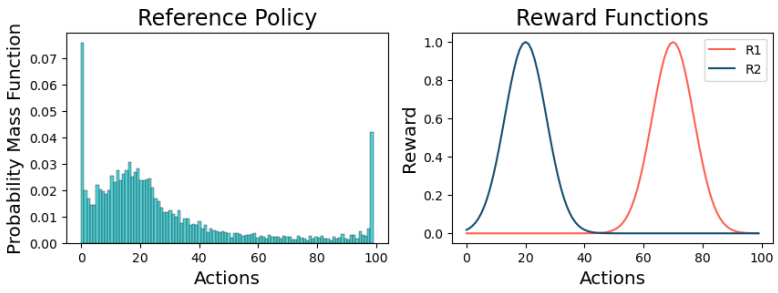

Figure 2: The didactic bandit problem which we use for our analysis in this paper. Reference policy initialization and reward slice for each token (the total reward is a mean of token-level rewards). The optima of reward functions $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ occur in low density and high density regions respectively.

Synthetic LLM fine-tuning problems. Next, we will generalize our intuitions from bandit problems to the LLM setting. Instead of directly experimenting with human preferences, we first study two synthetic problems that utilize hand-crafted reward functions, which can be approximated via reward models. Access to functional forms of these hand-crafted reward functions will enable us to track the ground-truth objective throughout training to see if our insights about various approaches under condition [C1] will hold even when learning against a reward model. Subsequently, we run this experiment with an altered skewed preference data distribution (see Figure 3) to understand the effect of coverage conditions [C2]. We consider two reward functions: **(1)** one that minimizes the response length ("**Min Length**"), analogous to $\boldsymbol{R}_1$ in the bandit problem, and **(2)** that attempts to anchor the response length to a pre-specified target value ("**Avg Length**"), which lies in the mode of the target distribution. This second condition exhibits similar characteristics to $\boldsymbol{R}_2$. The **Skew Length** scenario skews the preference data in the **Min Length** problem scenario.

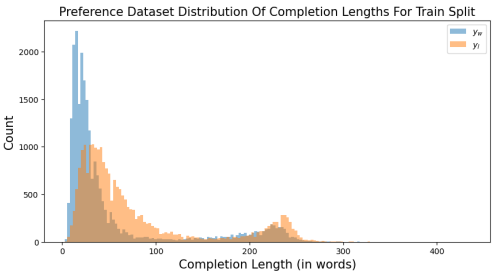

Figure 3: Skewed preference word length distribution. Above, we show the word length distribution for the **Left:** preferred (y_w) and **Right:** dispreferred (y_l) completions.

Full-scale LLM fine-tuning. Finally, we scale up our study to full-scale LLMs, with real preference data. Recent work (Singhal et al., 2023) shows that preference labels are usually biased towards much longer responses, indicating that preference fine-tuning usually admits a geometric relationship where the mode of the reward function is distinct from the mode of human data (and hence, any reference policy). For the majority of our experiments, we use preference datasets from the AlpacaFarm benchmark (Dubois et al., 2024). We also scale up our experiments to UltraChat (Ding et al., 2023), a ~ 10 times larger dataset with responses from many strong LLMs such as GPT 4 and GPT-3.5.

4.3. A Generic Fine-Tuning Algorithm Encapsulating All Axes

To systematically analyze the behavior of fine-tuning methods that differ along the axes discussed in Section 3.2, in this section, we introduce a generic algorithm with different hyperparameters associated with each axes. With a generic algorithm of this sort, we will be able to answer our research questions by varying each hyperparameter. Our unified practical algorithm is shown Algorithm 1. While on-policy algorithms perform steps 1 and 2 of on-policy data collection with a reward model, purely offline methods (e.g., DPO and RWR) utilize preference data directly.

Algorithm 1 A Unified Fine-Tuning Algorithm

for training iterations **do**
  (1) Sample B/C prompts $[\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_B]$.
  (2) Generate dataset $\mathcal{D}$ with C responses for B/C prompts, from the policy for online ($\boldsymbol{y}_i^1, \boldsymbol{y}_i^2, \cdots, \boldsymbol{y}_i^C \sim \pi_\theta(\cdot|\boldsymbol{x}_i)$) or from an offline dataset ($\boldsymbol{y}_i^1, \boldsymbol{y}_i^2, \cdots, \boldsymbol{y}_i^C \sim \mathcal{D}_{\text{pref}}$).
  (3) If applicable, label the responses $\boldsymbol{y}_i^1, \boldsymbol{y}_i^2, \cdots, \boldsymbol{y}_i^C$, with rewards drawn from your learnt reward model $\widehat{r}_\phi(\boldsymbol{y}|\boldsymbol{x})$
  for T inner iteration steps **do**
    (a) Divide $\mathcal{D}$ into minibatches $\mathcal{D}_1, \ldots, \mathcal{D}_N$, each with M prompts-response pairs
    for $i = 1, \ldots, N$ **do**
      (i) Apply the gradient of the objective $\mathcal{L}(\theta; \mathcal{D}_i; \widehat{r}_\phi)$ prescribed by the fine-tuning method.
    end for
  end for
end for

To study the impact of on-policy sampling, we vary the extent to which updates are made on data from the current policy. We can control this by two means in Algorithm 1: **(1)** by varying the total number of samples $|\mathcal{D}| = \frac{B}{C} \times C = B$ used for a given training iteration but keeping minibatch size M fixed, assuming the algorithm performs exactly one pass over all this sampled data, and **(2)** by varying the number T of gradient steps performed before refreshing the dataset $\mathcal{D}$ of samples (i.e., a larger T leads to more off-policy updates). In other words, approach **(1)** will perform more updates using stale data for large values of $|\mathcal{D}|$; and for small values of $|\mathcal{D}|$, approach **(2)** will make more off-policy updates if T is larger. While both approaches enable us to control how on-policy an algorithm is, approach **(1)** does not reuse samples (since $\mathcal{D}$ is large), but approach **(2)** reuses samples for different number of gradient updates, controlled directly by T. By studying both approaches for inducing off-policyness, we are able to isolate the effect of sample reuse on on-policy methods. We also study offline methods with no on-policy sampling, such as DPO and filtered supervised learning on the preferred response $\boldsymbol{y}_w$ in the dataset to understand the role of the negative gradient.

5. Empirical Analysis Results

In this section, we will present the results of our empirical study to answer our research questions. To answer each question, we will begin by studying the didactic bandit problem with the ground-truth reward function, followed by synthetic and then full-scale LLM fine-tuning problems.

5.1. Question 1: The Role of On-Policy Sampling

To understand the role of on-policy sampling, we will investigate if on-policy sampling can improve performance for several approaches followed by making conclusions regarding sample reuse.

5.1.1. Takeaway 1: On-Policy Sampling Generally Improves Performance

We first study on-policy sampling as a function of the geometric relationship [C1] in our bandit setting (see Figure 2), with no sampling error. Then, we will extend our conclusions to the LLM setting.

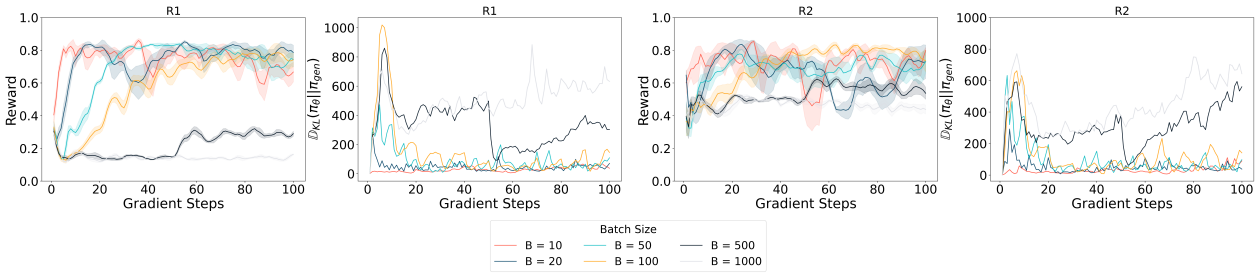

Figure 4: On-policy sampling on bandit problems. Performance of on-policy best-of-N as a function of the data sampled in each iteration. Larger batch sizes result in more off-policy updates. **Left**: (i) reward vs update step for $\boldsymbol{R}_1$, (ii) divergence between the policy parameters and data collection policy during training; **Right:** (i) reward vs update step for $\boldsymbol{R}_2$, (ii) KL divergence for $\boldsymbol{R}_2$. Observe the slow learning speed of more off-policy updates in $\boldsymbol{R}_1$, but less severe degradation for $\boldsymbol{R}_2$, where peaks in the reference policy and reward function are more aligned.

Didactic bandit problems. Figure 4 shows that given a fixed amount of total data budget, *sampling data more frequently from more recent policies*, but in smaller batches, results in better performance with both $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$. Doing so, naturally makes the algorithm more on-policy since each gradient update uses a mini-batch sampled from a more recent policy. This is also reflected in larger values of divergences between the sampling policy π_{gen} and the policy π_θ, $\mathrm{D}_{\mathrm{KL}}(\pi_\theta, \pi_{\text{gen}})$, in Figure 4. Concretely, larger B results in higher peak values of this divergence during training indicating farther deviation from the data at intermediate times during training. This means that being more on-policy corresponds to better performance and faster convergence for best-of-N. Similar conclusions can be made for on-policy RL methods such as REINFORCE and PPO as shown in **??**.

That said, we also note in Figure 4 that the performance degradation with more off-policy updates is substantially more mild for $\boldsymbol{R}_2$, indicating that when the peak in the reward function lies in the high likely regions of the reference policy, a higher degree of off-policy updates are tolerable.

[C1] ↓ || [C2] →	high $\mathcal{D}_{\text{pref}}$ and π_{ref} overlap	low $\mathcal{D}_{\text{pref}}$ and π_{ref} overlap
peaks of r^* and π_{ref} overlap	✓ **Mode Length**	×
peaks of r^* and π_{ref} disjoint	✓ **Min Length**	✓ **Skew Length**

Table 2: **Coverage conditions and geometric relations** that we study with synthetic LLM fine-tuning data. The three settings we study differ in terms of overlap between π_{ref}, reward r^*, and the preference dataset, $\mathcal{D}_{\text{pref}}$.

Synthetic LLM problems. In this problem setting, we wish to incorporate the effects of sampling error when learning with a proxy RM. Per Section 4.2, to do so we construct three scenarios that differ along geometric ([C1]) and coverage ([C2]) conditions as depicted in Table 2. To recap, the peak of the reward function in the **Min Length** scenario appears in the less likely regions of π_{ref}, whereas the peak of the reward function in the **Mode Length** scenario appears in highly likely regions under π_{ref}.

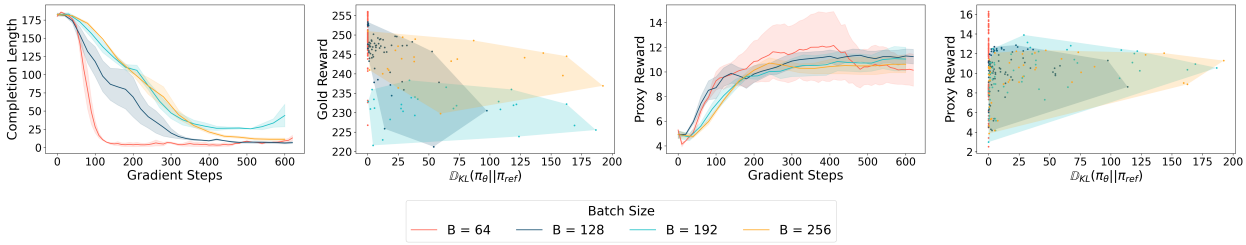

Figure 5: On-policy sampling for PPO in the Min Length scenario. This plot keeps the minibatch size M fixed to 64, but samples more stale data when B is large. Increasing B results in more off-policy updates and consequently slower convergence to ground-truth reward (i.e., a completion length of 0). (From left) **First**: average completion length (lower the better), **Second**: gold reward vs KL tradeoff (note that the smallest B achieves the best tradeoff)s, **Third**: proxy reward, **Fourth**: proxy reward vs KL tradeoff. Being more on-policy results in better performance.

We present our results for one algorithm in detail (in this case, PPO) (Figures 5 to 7) and then

present a summary bar chart showing that our conclusions also transfer to other algorithms (such as REINFORCE and RWR) (Figure 8). Corroborating insights from the bandit problem, in the **Min Length** scenario, we find that being more on-policy (i.e., a smaller batch size) leads to a lower completion length and hence a higher golden reward, despite potential inaccuracies in the proxy reward model that PPO is actually optimizing (Figure 5). Akin to our bandit experiments, we also observe that smaller batch sizes ($B = 64$ and $B = 128$) optimize the proxy reward at a faster rate compared to $B = 192$ and $B = 256$. This indicates that with a significant overlap between the preference data and the reference policy, on-policy sampling still leads to better performance with fewer updates. Finally, also note that more on-policy updates lead to a more favorable gold reward vs KL tradeoff (Figure 5). We also find similar trends across on-policy variants of RWR and REINFORCE, where modulo training instabilities, being more on-policy results in better performance (Figure 8; **Min Length**).

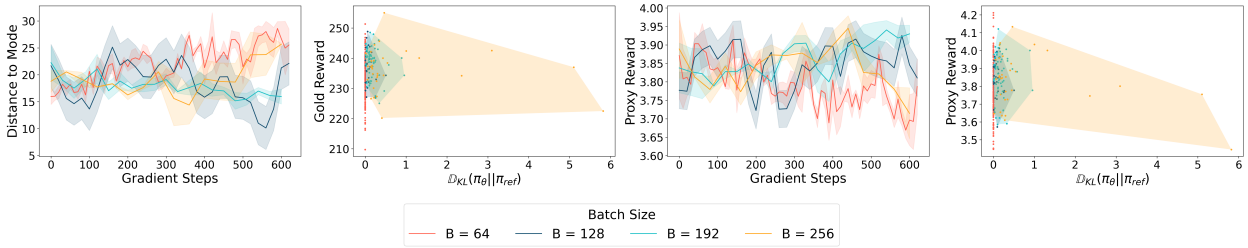

Figure 6: On-policy sampling for PPO in the Mode Length scenario. In this case, since the peak in the reward function and the highly likely regions of the reference policy are close, we find that the degree of on-policyness does not significantly affect performance. (From left) **First**: average completion length, **Second**: gold reward vs KL tradeoff, **Third**: proxy reward, **Fourth**: proxy reward vs KL tradeoff.

In the **Mode Length** scenario, where the preferred response for each preference pair are those that are closest to the average length in the dataset (203), varying the degree of on-policy sampling by adjusting the sampling frequency largely does not affect either the proxy or gold reward for PPO (Figure 6). We make similar observations for other algorithms:(Figure 8; **Mode Length**): different degrees of on-policyness perform similarly, except the more on-policy runs sometimes exhibit instability. This is in agreement with the results from the bandit setting above: when the peak in the reward function lies in highly likely regions under the reference policy, on-policy sampling has minor effect and more off-policy configurations can work well too.

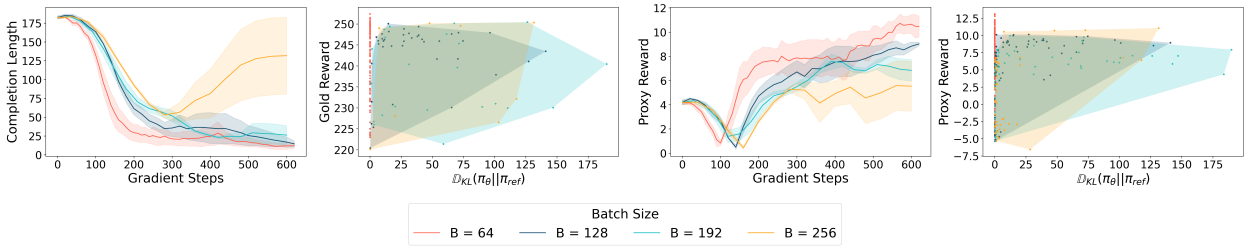

Figure 7: On-policy sampling for PPO on the Skew Length scenario. Being more on-policy results in faster convergence and better performance. (From Left) **First**: average completion length, **Second**: gold reward (= max length - completion length) vs KL tradeoff, **Third**: proxy reward, **Fourth**: proxy reward vs KL tradeoff.

Finally, to evaluate the robustness of these findings under more challenging coverage conditions, we deliberately skew the length distribution in the preference dataset to make it distinct from the reference policy (called **Skew Length**). Concretely, with a 95% probability, we truncate the length of the response by sampling a length from an exponential distribution, which naturally leads to a shorter completion length. The remaining 5% of samples are drawn from the standard SFT policy to simulate the broader coverage for the preference data. Overall, the resulting data admits a significantly skewed distribution over response lengths, as visualized in Figure 3. Not only does the peak in the reward function now appear in less likely regions of the reference policy, but to succeed, an optimization algorithm must now do the required heavy lifting to shift the probability mass to the low-density regions of the response space that maximize reward.

Our detailed results of running PPO in this setting are shown in Figure 7. In this setting, we still find that more on-policy updates lead to a higher gold reward with PPO. In addition, we also observe much larger gaps in proxy reward values attained at any given gradient step compared to the **Min Length** scenario, in favor of on-policy sampling. For other algorithms, we also observe strong and clear trends supporting that on-policy sampling with a smaller but frequently sampled batch results in better performance as shown in the summary plot (see Figure 8; **Skew Length**).

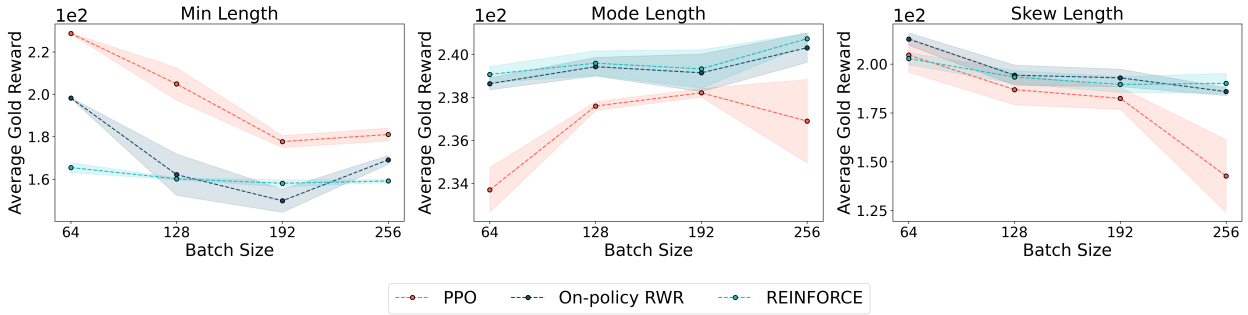

Figure 8: Summary: effect of on-policy sampling on synthetic LLM problems. Average gold reward over the course of training for RWR, and REINFORCE with different B. For **Min Length** and **Skew Length**, generally being more on-policy (i.e., smaller batch size) leads to higher gold reward. For **Mode Length**, all batch sizes perform close to each other, with performance differences largely due to instability.

Full-scale LLM problems. Finally, we evaluate if our insights transfer to the full-scale AlpacaFarm setup. We use a Pythia-1.4B model as our reference policy and generate two responses per prompt. We label the preferred and dispreferred responses with a gold reward model of human preferences from AlpacaFarm to construct a preference dataset. Figure 9 shows that our intuitions from the simple bandit and synthetic LLM experiments transfer to this real preference learning task, as making updates on only on-policy samples leads to higher gold reward for both on-policy RWR and REINFORCE. We observe that for the case of REINFORCE, the most on-policy run ($B = 64$) also provides a more favorable gold reward vs KL tradeoff (as indicated by the red polygon for REINFORCE in Figure 9).

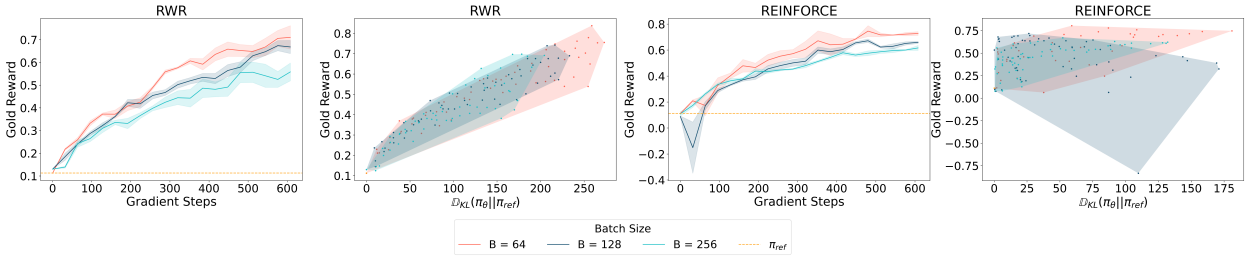

Figure 9: Effect of on-policy sampling on AlpacaFarm with a fixed mini-batch $M = 64$, but varying batch size B. As discussed, increasing B makes the algorithm more off-policy and this results in lower performance. (From left) **First**: gold reward vs gradient steps for RWR, **Second**: gold reward vs KL tradeoff for RWR, **Third**: gold reward vs gradient steps for REINFORCE **Fourth**: gold reward vs KL tradeoff for REINFORCE.

5.1.2. Takeaway 2: On-Policy Sample Reuse Can Enable Leveraging Off-Policy Data

In the previous section, exactly one gradient step was taken on a given sample and we found that making updates on stale data was not helpful due to off-policy updates. **Is there any scenario under which we can still attain good policy performance despite employing off-policy updates?** In this section, we will answer this question, and show that it might be possible to learn with off-policy updates for some algorithms if we are allowed to make more than one update on a given sample. Of course, a substantial amount of sample reuse is detrimental since it would lead to more off-policy updates, thus leading to statistical or propensity overfiting (Swaminathan and Joachims, 2015), but it is reasonable to surmise that some amount of sample reuse can help. To study the effect of sample reuse, we compare methods when $T > 1$ gradient steps can be made on a given sample.

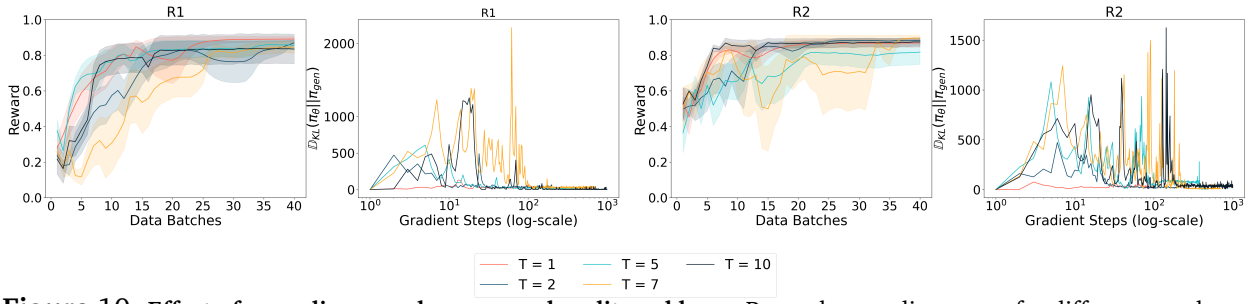

Figure 10: Effect of on-policy sample reuse on bandit problems. Reward vs gradient steps for different number of inner iteration steps, T, on the same data batch for RWR. Increasing T controls the number of gradient steps taken before collecting the new batch of on-policy samples.

We study sample reuse for on-policy RWR in the bandit setting in Figure 10. While increasing T can slow down convergence in general, we note that using a larger value of T may be better (e.g., $T = 5$ learns faster than $T = 2$; $T = 10$ learns faster than $T = 7$).

Synthetic LLM problems. We also evaluate the effect of sample reuse on synthetic LLM problems. In this case, we study two algorithms PPO and on-policy best-of-N to be able to understand the effect of sample reuse on multiple algorithms. In contrast to the performance degradation with off-policy updates induced due to stale samples in PPO, we find that off-policy updates induced due to sample reuse do not hurt performance (Figure 11; PPO), with even $T = 8$ performing similarly to $T = 1$. On the other hand increasing T from 1 to 2, i.e., performing two gradient updates on each sample improves the golden reward for best-of-N (Figure 11; Best-of-N) within a given data sampling budget.

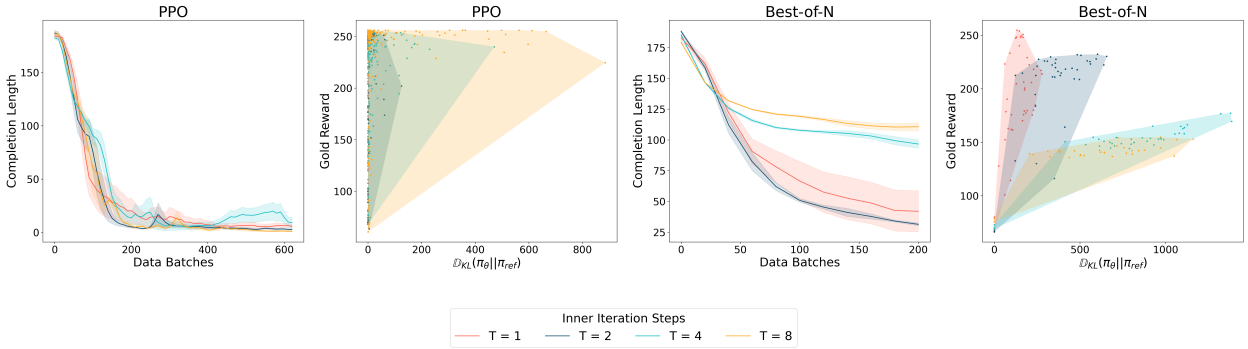

Figure 11: Effect of on-policy sample reuse in the Min Length scenario. Average completion length (i.e., the lower the better) vs gradient steps for different number of inner iteration steps, T, on the same data batch. A larger value of T implies that the algorithm is more off-policy. Observe that some sample reuse can improve sample efficiency (T = 2 outperforms T = 1), but excessive sample reuse can hurt performance. Also note that algorithms with mechanisms to control off-policy updates such as PPO with importance-weight clipping are suited to perform better in the off-policy sample reuse setting.

Why do PPO and best-of-N respond differently to sample reuse? We believe that this is because PPO employs an off-policy correction, and hence, significantly off-policy samples do not contribute to the gradient, addressing the well-known challenge of propensity overfitting (Swaminathan and Joachims, 2015). This is not the case with best-of-N, where excessive sample reuse can hurt exploration, because training on old samples with a log-likelihood loss push the current policy to be close to the stale data-generating policy. That said, more than one gradient step can still be useful when presented with a fixed data budget, unless it bottlenecks exploration of high reward regions.

Takeaways for on-policy sampling

To summarize, these results imply that on-policy sampling generally improves performance and efficiency, especially in cases when the peak of reward appears farther from the reference policy. In some cases, sample reuse can reduce the dependency on on-policy sampling of data, but it presents a tradeoff by reducing exploration of the response space.

5.2. Question 2: The Role of Negative Gradient

To understand the role of negative gradient, we will compare contrastive algorithms such as DPO and IPO with maximum likelihood methods such as RWR (or Pref-FT, which attempts to increase the likelihood of the preferred response only) and best-of-N in a fully offline setting, where no new on-policy samples are collected. We compare various algorithms in terms of performance and then analyze other metrics to understand the mechanisms behind this performance increase.

5.2.1. Takeaway 1: Negative Gradient Enables Faster Convergence

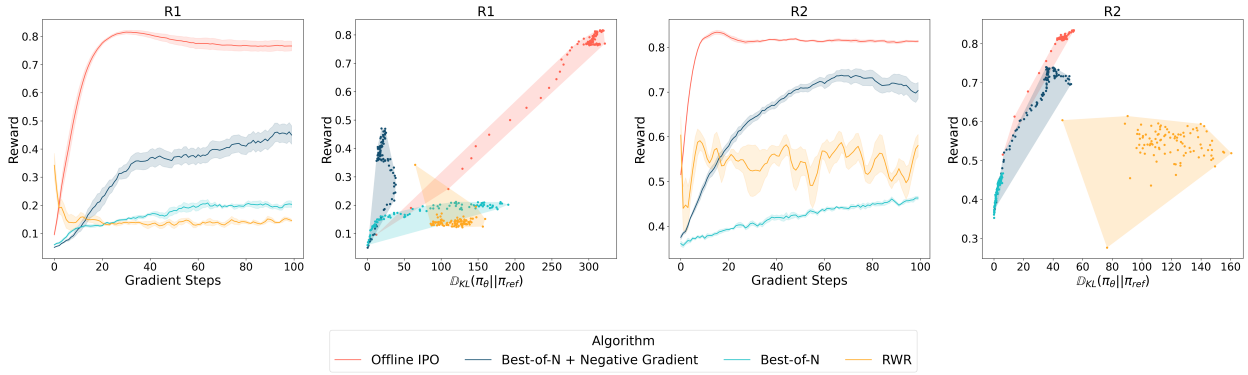

Figure 12: Negative gradients on the bandit. Average reward during training and the KL-reward trade-off for four algorithms in the fully offline setting: best-of-N (no negative gradient), RWR (no negative gradient), best-of-N + an explicit negative gradient on less preferred actions, and IPO (with negative gradient). Negative gradient help find a better policy by aggressively pushing down likelihoods of less preferred actions.

We begin by comparing a representative set of algorithms on the didactic bandit problem. These methods include those that do not use a contrastive update on the didactic bandit problem, namely offline supervised approaches, Best-of-N and offline RWR, and offline IPO (Gheshlaghi Azar et al., 2023), a representative offline fine-tuning method which uses a contrastive negative gradient term. We also consider a variant of best-of-N where we explicitly add a term to the loss function that attempts to minimize likelihood of the dispreferred response akin to unlikelihood (Welleck et al., 2020) (see Appendix G.2 for more details). In Figure 12, we find that IPO and best-of-N + negative gradient learn a better policy from an offline dataset collected from sub-optimal π_{ref}, compared to best-of-N and RWR. IPO achieves a better KL-reward trade-off in $\boldsymbol{R}_1$ (where high likelihood regions of π_{ref} and the peak in r^* are far away from each other). While best-of-N attains a higher reward when the reward function is given by $\boldsymbol{R}_2$ (where the peaks in π_{ref} and r^* overlap) compared to $\boldsymbol{R}_1$, it still underperforms IPO. We suspect that this is because maximizing likelihood on some responses alone is not enough to steer the learned policy away meaningfully away from π_{ref} towards the peak in the reward function, especially when this peak is far away from π_{ref}. Best-of-N + negative gradient significantly outperforms Best-of-N in both scenarios and closes the performance gap to IPO, which shows that explicit adding a loss term to minimize probability on less preferred responses can provide substantial performance improvement. That said, for reward function $\boldsymbol{R}_2$, we also observe a smaller gap between the best algorithm without a negative gradient (i.e., RWR) and offline IPO, indicating that when the peak in π_{ref} and r^* exhibit more overlap, the performance benefits of contrastive training are smaller. We also investigated a simpler 1-token bandit problem where we found best-of-N to be better than IPO for $\boldsymbol{R}_2$. This is possibly due to the much smaller space of possible tokens and responses, where maximum likelihood methods perform well enough.

Synthetic LLM problems. Our experiments in the synthetic LLM setting corroborate this finding. Here we compare Pref-FT with DPO (with negative gradients). In the **Min Length** setting, we find in Figure 13 that DPO significantly outperforms Pref-FT. On the other hand, when the peak in the ground-truth reward appears in high-likely regions of the reference policy and the preference data

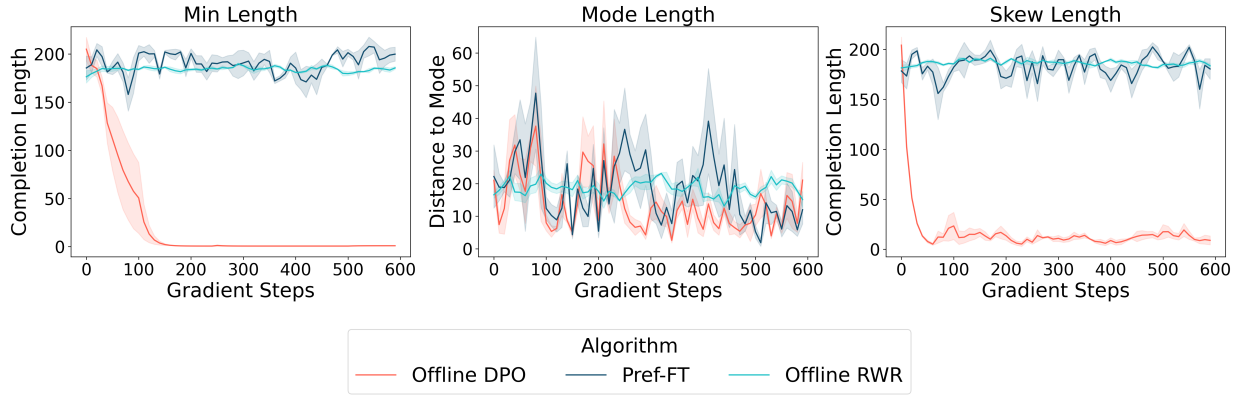

Figure 13: Negative gradients in synthetic LLM problems. Completion length (inverse of the true reward) for three offline algorithms. DPO outperforms Pref-FT and offline RWR in **Min Length** and the **Skew Length** settings, where the peak in r^* and π_{ref} are misaligned. For the **Mode Length** setting, all of the algorithms perform similarly.

$\mathcal{D}_{\text{pref}}$ covers this region (**Mode Length**), we find both approaches to perform similarly. Finally, in the **Skew Length** scenario when π_{ref} and $\mathcal{D}_{\text{pref}}$ do not overlap significantly, but the peak in r^* is covered by the preference dataset $\mathcal{D}_{\text{pref}}$, we also find that DPO is much more effective in driving the policy further from the reference initialization and outperforms Pref-FT.

Full-scale LLM fine-tuning. Finally, we compare supervised Pref-FT and contrastive DPO when fine-tuning on actual preference data. In addition to AlpacaFarm, we also run experiments using the Ultra-Feedback (Ding et al., 2023) dataset. For the Ultra-Feedback dataset, we use different models (GPT-3.5, GPT-4) to generate responses to various prompts. The resulting dataset has a broader preference dataset distribution than π_{ref}.

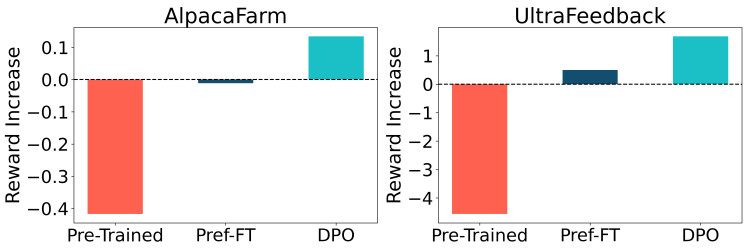

Figure 14: Negative gradients in AlpacaFarm (left) and UltraFeedback (right). For these domains, we consider the increase in average gold reward **compared to the SFT model** for different offline approaches. Algorithms with a negative gradient such as DPO outperform approaches such as Pref-FT not utilizing any negative gradient term.

In this setting, we utilize a checkpoint of the Mistral7B model obtained by running supervised next-token prediction on a subset of UltraChat (comprising of GPT-3.5 responses) as the reference initialization. We use the UltraRM model with a LLaMA2-13B base architecture as our gold reward model. As shown in Figure 14, DPO which utilizes a negative gradient shows a much larger improvement over the reference policy π_{ref} (in this case the SFT model) compared to methods that do not utilize a negative gradient (e.g., Pref-FT).

5.2.2. Takeaway 2: Mechanisms Explaining the Behavior of the Negative Gradient Term

Having seen that using a negative gradient leads to much better performance, we next attempt to understand the mechanism behind this better performance. To do so, we visualize the evolution of the log likelihoods of the preferred response and the dis-preferred response in a held-out dataset as multiple gradient steps are taken on an offline preference optimization loss.

Contrastive training increases the gap between the likelihoods of preferred and less preferred responses. Perhaps as expected, we find that contrastive training is more effective at increasing the gap between the likelihoods of preferred and less-preferred responses compared to offline Pref-FT in several LLM settings: the synthetic LLM settings with **Min Length** and **Skew Length**, and full-scale AlpacaFarm and UltraFeedback settings (Figure 15). More concretely, note that the margin for Pref-FT largely converges to 0, whereas offline DPO can enable a more positive margin.

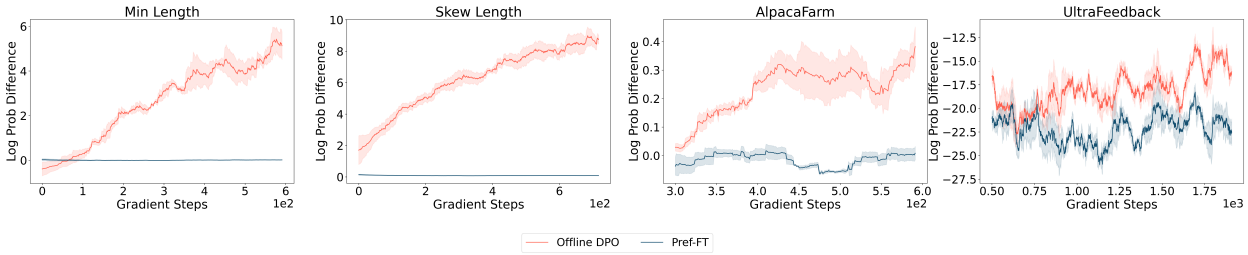

Figure 15: Difference in likelihoods of preferred and dispreferred responses. DPO increases the margin $\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x}) - \log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})$ more compared to non-contrastive methods such as Pref-FT.

Changes in log likelihoods depend on model capacity, reference initialization, and data composition. The next question to ask is if the probability mass recovered by increasing the margin between the positive and negative response goes into increasing likelihood of the positive response. We track the likelihoods $\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})$ and $\log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})$ on the bandit problem, while varying the size of the preference dataset. Observe in Figure 16 that when the dataset size is small relative to the model capacity, contrastive training via IPO is able to increase likelihood of $\boldsymbol{y}_w$ while reducing likelihood of $\boldsymbol{y}_l$. However, as the number of prompts increases, contrastive training counter-intuitively results in a decreasing value of $\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})$, although the margin is still increasing. The recovered probability mass is instead used to increase likelihood of already likely responses under the current policy $\pi_\theta(\neq p(\boldsymbol{y}_w|\boldsymbol{x}))$. This means that in certain problems, depending upon the initialization, contrastive training might extrapolate to producing good or bad responses.

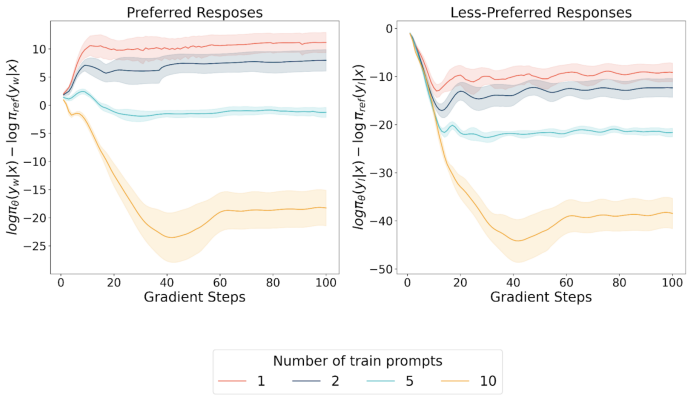

Figure 16: Log likelihood margins. We find that with fewer prompts, contrastive methods can increase the log-likelihood of the positive response while reducing the likelihood of the negative response, however as the number of data points grows, this may not be possible and the likelihood of both positives and negatives might reduce.

We also observe a similar trend in full-scale LLM experiments in Figure 17: we observe a decrease in the log likelihoods of both the preferred and dis-preferred responses over the course of training on AlpacaFarm with small 1.4B Pythia policies. However, using a Mistral7B model to train a policy on the Ultra Feedback dataset results in an increasing value of log likelihood of $\pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})$ and a decreasing value of $\pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})$, when starting from a SFT on Ultrachat-200K initialization (same setup as Zephyr (Tunstall et al., 2023)). We believe that these opposite trends are a consequence of the responses for a given prompt in the UltraFeedback dataset appearing more semantically distinct from each other, as different responses come from models with different capabilities (e.g., a GPT-4 response is paired with a GPT-3.5 response) such that given enough model capacity, contrastive training is able to push up likelihoods of $\pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})$ while pushing down $\pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})$. In contrast, running Pref-FT increases likelihoods of both $\boldsymbol{y}_w$ and $\boldsymbol{y}_l$ (Figure 17).

Takeaways for negative gradients

A negative gradient is useful when the peak in the reward function appears in less likely regions of the reference policy. It can increase the likelihood of y_w when y_l is semantically different and the model capacity is large enough. If not, we will still observe an increase in the likelihood margin $\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x}) - \log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})$, but the resulting probability mass will increase likelihood of the highly likely regions under the current policy π_θ (and not the positives $p(\boldsymbol{y}_w|\boldsymbol{x})$).

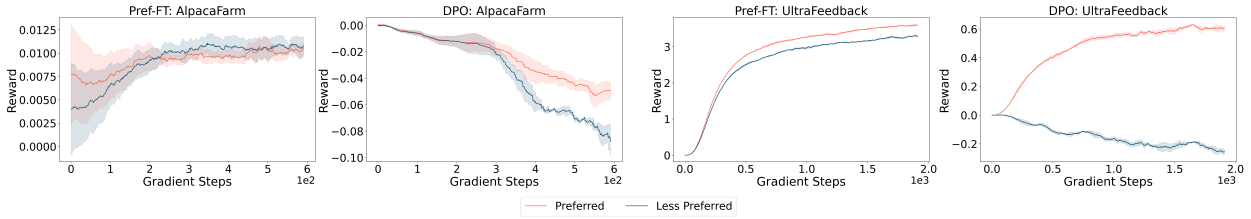

Figure 17: DPO reward estimates for Pref-FT and DPO on AlpacaFarm and UltraFeedback. For a Pythia-1.4B model trained on AlpacaFarm, DPO decreases the implicit reward, $r_\theta(\boldsymbol{x}, \boldsymbol{y}) = \log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) - \log \pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x})$, for both $\boldsymbol{y}_w$ and $\boldsymbol{y}_l$, whereas Pref-FT increases both. For a Mistral-7B model trained on UltraFeedback, DPO is able to increase reward for $\boldsymbol{y}_w$ and decreases reward for $\boldsymbol{y}_l$, whereas Pref-FT increases both. In both cases, DPO leads to a higher margin than Pref-FT.

5.3. Question 3: Complementarity of On-Policy Sampling and the Negative Gradient

Based on our findings that both on-policy sampling and negative gradients are independently effective, we now study if combining them would provide any additional benefits. To understand this, we develop a straightforward on-policy variant of DPO/IPO: instead of utilizing the PPO or Best-of-N objective on on-policy samples, for each prompt $\boldsymbol{x}$, we sample N responses from the policy $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n \sim \pi_\theta(.|\boldsymbol{x})$, rank them according to a reward model r_ϕ, and construct preference pairs by taking the higher reward completion as the preferred one and lower reward completion as the less-preferred one. This recipe is similar to concurrent works such as Rosset et al. (2024). Then we use calculate the DPO/IPO loss on this preference dataset and update our model accordingly.

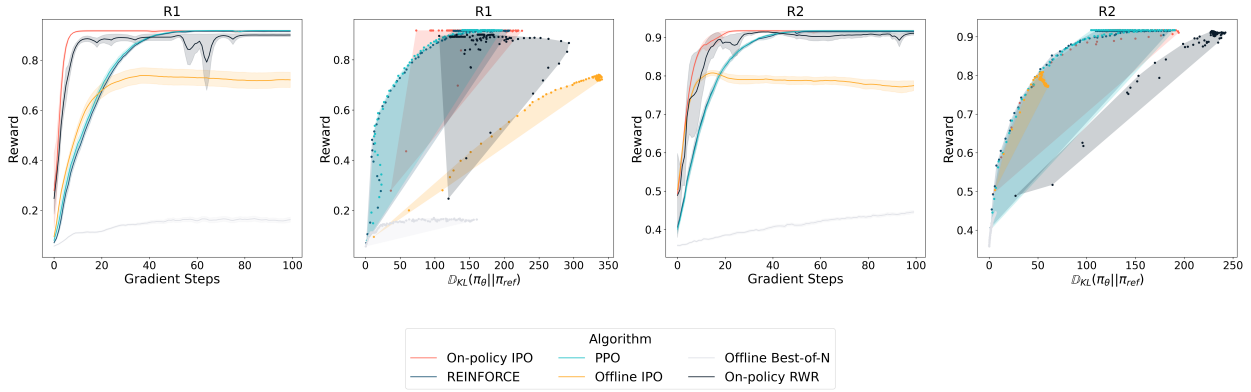

Figure 18: On-policy sampling + negative gradients in bandit setup. Complimentary benefit of on-policy sampling and negative gradients. Online IPO (using both on-policy sampling and negative gradients) performs better than offline IPO (negative gradients but no on-policy sampling) and RWR (on-policy sampling but no negative gradients).

Performance on bandit and synthetic LLM problems. Figure 18 shows that the on-policy version of IPO achieves both faster convergence and better performance compared to the offline version, for both $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ in the didactic bandit problem. We also ran on-policy DPO in the synthetic LLM settings and found it to converge significantly faster and to a better solution than offline DPO, on-policy RL, and on-policy variants of supervised learning approaches as shown in Figure 19. We also find that on-policy versions of contrastive approaches exhibit favorable computational vs wall-clock time tradeoffs compared to purely on-policy RL methods and even offline contrastive methods that may not find as good solutions as their on-policy counterparts (see Appendix B).

Why can on-policy versions of contrastive methods perform better than on-policy RL? We saw in Section 5.2.1 that offline contrastive training with a negative gradient was effective at quickly reorganizing probability mass to high-reward responses covered by the preference data. When combined with on-policy sampling, this behavior results in faster convergence: for any given batch of on-policy data, contrastive training with a negative gradient is able to quickly reconfigure the policy distribution within the support of the on-policy data obtained thus far. Similarly to how best-of-N + negative gradient outperformed vanilla best-of-N but underperformed DPO in Figure 12, PPO also

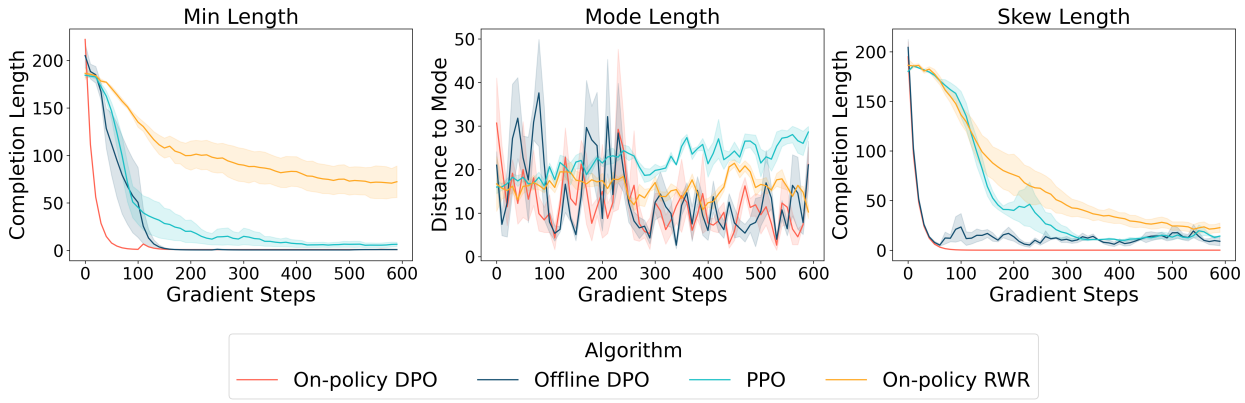

Figure 19: On-policy sampling + negative gradients in LLM length experiments. Complimentary benefit of on-policy sampling and negative gradients on the synthetic LLM length experiments. Online DPO performs the best where optimal policy and reference policy lies far from each other (min length and skew length), and all algorithms perform similarly when these two policies are close (mode length).

improves over RWR that does not have a negative gradient term, but is unable to match on-policy DPO in Figure 19. Note that this does not mean that, in general, on-policy DPO would always outperform PPO, but that it might be a good choice to try on-policy versions of contrastive methods.

Takeaways for on-policy sampling + negative gradient

On-policy sampling and offline negative gradients present complementary benefits, in that the best offline loss function with negative gradients can be used to train on on-policy data, improving over on-policy RL or supervised learning. Conceptually, while sampling responses on-policy provides coverage of the response space, an effective negative gradient loss provides a stronger learning signal given a sampled dataset. This can also result in computational benefits.

6. Conceptual Unification and Theoretical Analysis

With empirical results showing the benefits of on-policy sampling and negative gradient for preference fine-tuning of LLMs, in this section, we attempt to conceptually understand the benefits by building a mental model. In this section, we will first unify these seemingly distinct notions of on-policy sampling and negative gradient into a unified notion of mode-seeking objectives, in contrast to mode-covering maximum likelihood objectives. Then, we will contrast the learning dynamics of the reverse KL-divergence, a representative mode-seeking objective against the mode-seeking forward KL-divergence (i.e., the supervised learning loss) to corroborate some of our findings.

6.1. Seeking Modes Unifies On-Policy Sampling and Negative Gradients

In this section, we will show that the notion of mode-seeking divergences unifies on-policy sampling and negative gradients for the various objectives we investigated in the paper. Specifically, we show below that several on-policy RL methods that we studied optimize the reverse KL-divergence (with additional regularization in PPO) and are hence mode-seeking, offline contrastive methods that employ a negative gradient are also mode-seeking (due to the negative gradient), and finally, supervised maximum likelihood approaches (e.g., offline Best-of-N, Pref-FT, Binary FeedMe) are mode-covering.

To show that on-policy RL methods are mode-seeking, we show that the optimization objective of these methods is a reverse KL-divergence, which is known to be mode-seeking.

Lemma 6.1. *On-policy RL methods optimize regularized version of a reverse KL-divergence with respect to the optimal policy and are hence mode seeking.*

A proof for Lemma 6.1 is shown in Appendix C.1.1. Next, we show that offline contrastive methods that employ a negative gradient are also mode-seeking. While these approaches do not optimize the reverse KL-divergence, we can still show that the probability mass obtained by minimizing density on negative responses $\boldsymbol{y}_l$ gets disproportionately utilized, far more for increasing the probability mass on the "mode" (i.e., highest probability categories under the current policy π_θ) compared to other categories. When the offline dataset consists of multiple high-reward categories, this preference to put more probability mass on the mode of the current policy results in mode-seeking behavior, compared to increasing probability mass on all high-reward categories.

Lemma 6.2. *Let θ_t denote the parameters of the model at a given iteration t. Consider contrastive approaches that include a negative gradient under a functional form shown below:*

$$\theta_{t+1} \leftarrow \theta_t + \eta\, \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\nabla_\theta \log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x}) \cdot c_1(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l) - \nabla_\theta \log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x}) \cdot c_2(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\right]\Big|_{\theta_t}, \quad (6.1)$$

where c_1 and c_2 are non-negative functions that depend on the reward value and the associated samples, $\boldsymbol{y}_w$ and $\boldsymbol{y}_l$. In contrast, an update based on weighted maximum likelihood, without the negative gradient sets $c_2 = 0$. Define $\omega_t := \log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x}) - \log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})$. Then, for all possible models θ and for all t, there always exists an appropriate dataset of positive and negative samples $\mathcal{D}$, such that:

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\omega_{t+1}\right]\Big|_{c_2>0} \geq \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\omega_{t+1}\right]\Big|_{c_2=0}. \quad (6.2)$$

Furthermore if the model class representing the policy π_θ is expressive enough, such that

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\nabla_\theta \log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})^\top \nabla_\theta \log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})\right] \leq 0,$$

then, we additionally have that:

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})\right]\Big|_{c_2>0} \geq \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})\right]\Big|_{c_2=0} \quad (6.3)$$

$$\mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})\right]\Big|_{c_2>0} \leq \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l\sim\mathcal{D}}\left[\log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})\right]\Big|_{c_2=0} \quad (6.4)$$

A proof for Lemma 6.2 is in Appendix ??. This Lemma indicates that if the negative responses are chosen appropriately, then the contrastive update accelerates the rate of increase of probability mass on $\boldsymbol{y}_w$, for any model class π_θ and reference initialization θ_0, compared to only utilizing the positive gradient term (i.e., when $c_2 = 0$) that resembles weighted maximum likelihood. This corresponds to mode-seeking behavior. The update induced by DPO admits a similar form (see the discussion after Equation 7 in Rafailov et al. (2023)), and hence DPO is mode-seeking. This theoretical result also corroborates our findings in the experiments in Section 5.2.1 regarding the negative gradient term. The gradient of IPO also admits a similar form (Appendix C.1.2), and is hence mode-seeking.

Next, we note that purely offline versions of supervised methods such as RWR, ReST, and BoN, that only maximize likelihood are mode-covering because these objectives can be shown to maximize the forward KL-divergence against the optimal fine-tuning policy (proof in Appendix C.1.3).

Lemma 6.3. *Consider offline supervised methods such as ReST, BoN, and RWR, that maximize weighted log likelihood:*

$$\mathcal{L}_{\textit{off-sup}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\textit{pref}}}\left[\mathbb{E}_{\boldsymbol{y}\sim\pi_{\text{ref}}(.|\boldsymbol{x})}[\log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) \cdot F(\boldsymbol{x},\boldsymbol{y})]\right] \quad (6.5)$$

where $F(\boldsymbol{x},\boldsymbol{y}) \geq 0$ is the weight for $(\boldsymbol{x},\boldsymbol{y})$. Furthermore, $\sum_{\boldsymbol{y}} F(\boldsymbol{x},\boldsymbol{y}) > 0$ (i.e., for every $\boldsymbol{x}$, there exists a response $\boldsymbol{y}$ with non-zero $F(\boldsymbol{x},\boldsymbol{y})$). Then these methods optimize a forward KL-divergence like objective.

Mode-seeking objectives unify on-policy sampling and negative gradient

Utilizing either on-policy sampling or negative gradients induces mode-seeking behavior.

6.2. Case Study: Mode-Seeking Reverse KL vs. Mode-Covering Forward KL

Having seen that the notion of mode-seeking and mode-covering divergences can unify on-policy sampling and negative gradients, in this section, we perform a theoretical analysis to quantify the behavior of the two representative mode-seeking and mode-covering objectives: reverse KL (mode-seeking) and forward KL (mode-covering) objectives on categorical distributions, parameterized via a Gibbs distribution. Our goal is to formalize some of the intuitions pertaining to sharpening the probability mass on only certain categories and on-policy sampling.

Notation and setup. For this result, we will study training a categorical distribution $p(\boldsymbol{x})$ to match the theoretically optimal fine-tuned policy, $q(\boldsymbol{x})$. We assume that $p(\boldsymbol{x}) \propto \exp(f(\boldsymbol{x}))$, where each logit $f(\boldsymbol{x})$ is an independent parameter. We train $p(\boldsymbol{x})$ by performing gradient descent, starting from an initial reference distribution p_0 on a fine-tuning loss with gradient descent and a learning rate η. We denote the distribution at step t of this gradient descent as p_t. For this analysis it would be helpful to explicitly write out the parameter updates at any iteration t, induced by forward and reverse KL.

Lemma 6.4. *For any given distribution p_t, with $p_t(\boldsymbol{x}) = \exp(f_t(\boldsymbol{x}))$, the updates induced by the forward and reverse KL-divergences within one step of gradient descent with a learning rate η are given by:*

$$\textit{Forward KL:} \quad \log \frac{p^f_{t+1}(\boldsymbol{x})}{p_t(\boldsymbol{x})} = \eta \left(q(\boldsymbol{x}) - p_t(\boldsymbol{x})\right) + \mathbb{Z}. \tag{6.6}$$

$$\textit{Reverse KL:} \quad \log \frac{p^r_{t+1}(\boldsymbol{x})}{p_t(\boldsymbol{x})} = \eta \left(p_t(\boldsymbol{x}) \left[\log \frac{q(\boldsymbol{x})}{p_t(\boldsymbol{x})} + \mathbb{D}_{KL}(p_t(\cdot)||q(\cdot)) \right] \right) + \mathbb{Z}', \tag{6.7}$$

where $\mathbb{Z}$ and $\mathbb{Z}'$ denote constant normalization factors.

For a proof of Lemma 6.4, see Appendix C.2.

In principle, upon convergence, both the reverse and forward KL-divergences should find the optimally fine-tuned distribution, $q(\boldsymbol{x})$ in this simple setting. But to understand their behavior in relevant practical situations, we are particularly interested in understanding their behavior at intermediate points during training, when either divergence is not minimized to exactly 0. Insights about intermediate points in training are representative of practical problems when early stopping is used to prevent overfitting and the loss is rarely 0. Thus, our theoretical result below attempts to characterize the learning dynamics of these objectives at any given iteration t during training:

Theorem 6.5 (Informal)**.** *Let $p^f_{t+1}(\boldsymbol{x})$ be the distribution obtained after one gradient step, starting from p_t using the forward KL divergence. Likewise, let $p^r_{t+1}(\boldsymbol{x})$ be the distribution obtained using the reverse KL divergence, from p_t. Define Δ^f_t and Δ^r_t as the difference of log probability ratios across two categories $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$, obtained from the forward and reverse divergences respectively:*

$$\Delta^f_t(\boldsymbol{x}_1, \boldsymbol{x}_2) := \log \frac{p^f_{t+1}(\boldsymbol{x}_1)}{p_t(\boldsymbol{x}_1)} - \log \frac{p^f_{t+1}(\boldsymbol{x}_2)}{p_t(\boldsymbol{x}_2)}, \tag{6.8}$$

and Δ^r_t is similarly defined. Then we have the following (for appropriate positive constants β, δ_1, δ_2):

1. *If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are such that, $\delta_1 \leq p(\boldsymbol{x}_1) = p(\boldsymbol{x}_2) \leq 1 - \delta_2$ (where δ_1 and δ_2 are not too small), but $q(\boldsymbol{x}_1) \geq q(\boldsymbol{x}_2) + \beta$, then, $\Delta^r_t(\boldsymbol{x}_1, \boldsymbol{x}_2) > \Delta^f_t(\boldsymbol{x}_1, \boldsymbol{x}_2)$.*
2. *If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are such that, $p(\boldsymbol{x}_2) + \beta \leq p(\boldsymbol{x}_1) \leq 1 - \delta_2$, and $q(\boldsymbol{x}_1) = q(\boldsymbol{x}_2) > \boldsymbol{c}_0 \cdot p(\boldsymbol{x}_1)$, where $\boldsymbol{c}_0$ is a positive constant ≥ 1, then, $\Delta^r_t(\boldsymbol{x}_1, \boldsymbol{x}_2) > \Delta^f_t(\boldsymbol{x}_1, \boldsymbol{x}_2)$.*
3. *If $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ are such that, $p(\boldsymbol{x}_2) + \beta \leq p(\boldsymbol{x}_1) \leq 1 - \delta_2$, and $q(\boldsymbol{x}_1) = q(\boldsymbol{x}_2) < \boldsymbol{c}_1 \cdot p(\boldsymbol{x}_2)$, where $\boldsymbol{c}_1$ is a positive constant < 1, then, $\Delta^r_t(\boldsymbol{x}_1, \boldsymbol{x}_2) < \Delta^f_t(\boldsymbol{x}_1, \boldsymbol{x}_2)$.*

A proof of Theorem 6.5 is shown in Appendix C.4. This theorem enlists several cases where the forward KL modifies probability mass equally on all categories where $q(\boldsymbol{x})$ and $p_t(\boldsymbol{x})$ differ by a given amount, but the reverse KL acts disproportionately. In particular, case 1 shows that the reverse KL is particularly effective in removing probability mass from bins where $p_t(\boldsymbol{x})$ is substantially higher than $q(\boldsymbol{x})$ due to the logarithmic dependency on the probability mass $q(\boldsymbol{x})$ (compared to the linear dependency for the forward KL). Case 2 shows that when the target for two bins is much larger than the probability mass currently associated with those bins, then the reverse KL attempts to preferentially increase probability mass more in the bin with a larger likelihood $p_t(\boldsymbol{x})$. Finally, case 3 shows that when the likelihood of a bin is significantly larger than the target $q(\boldsymbol{x})$, the reverse KL is more effective at reducing this probability mass within one gradient step.

Finally, consider another special case (which is a direct consequence of case 2 in Theorem 6.5), where the difference $q(\boldsymbol{x}) - p_t(\boldsymbol{x})$ is identical for two categories $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. In this case, while the forward KL will increase log probability ratios for both $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ equally, i.e., $\Delta^f(\boldsymbol{x}_1, \boldsymbol{x}_2) = 0$, the reverse KL will prioritize the category with a higher $p_t(\boldsymbol{x})$ value (modulo some edge cases).

Mode-seeking vs. mode-covering objectives for categorical distributions

Prior work typically shows that the benefits of mode-seeking behavior of this sort are more apparent when the model $p(\boldsymbol{x})$ is unable to realize the target distribution $q(\boldsymbol{x})$, such that minimizing either KL would give rise to difference solutions. Unlike this prior argument, our argument above shows that even when the $p(\boldsymbol{x})$ can fully represent the target distribution $q(\boldsymbol{x})$, when training with gradient descent, reverse KL is able to quickly re-distribute probability mass to only a subset of the required categories likely in target distribution. We also illustrate this idea using a numerical toy experiment in Figure 20.

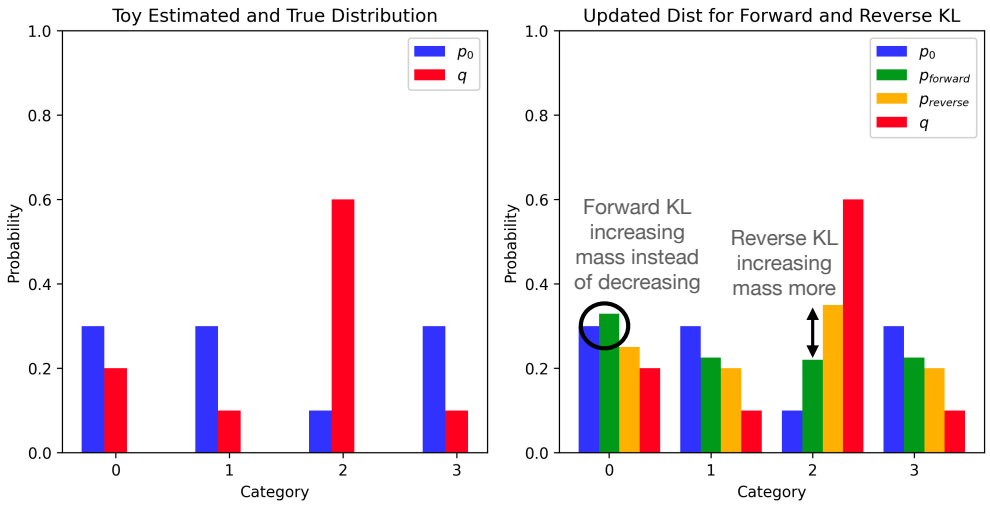

Figure 20: Numerical Experiment on Categorical Distribution. In this setting, the reverse KL strictly outperforms the forward KL in terms of distance to π_{ref}. We see for Category 1, the Forward KL increases in mass instead of decreasing. For Category 2 we see that the reverse KL, increases mass more than the forward KL.

7. Discussion, Conclusion, and Limitations

We attempted to understand which components are particularly important for fine-tuning language models with preference data. Through extensive experiments on different fine-tuning problems in both didactic and LLM settings, we established that on-policy sampling is crucial for good performance especially when the peak in the ground-truth reward lies in less-likely regions of the reference policy initialization. That said, in practice, doing so requires preference datasets with broader coverage than

the reference policy. We also showed that negative gradients can enable faster convergence and that objectives that induce a negative gradient are complementary to using on-policy sampling. While we conceptualize these observations, a limitation is that we don't derive rigorous statistical guarantees and this is an avenue for future work. It would also be interesting to study more recent approaches based on minimax formulations (Munos et al., 2023; Yuan et al., 2024; Swamy et al., 2024; Chen et al., 2024) in our framework. Next, while we consider the coverage of preference data relative to that of the reference policy in our study, this is a simplification that does not account for the coverage of the pre-training distribution which future work can incorporate. Finally, we remark that our study does not benchmark the quality of the reward model, which plays a crucial role in LLM fine-tuning. It would be importance to understand how our conclusions transfer to arbitrarily reward models.

Acknowledgements

We would like to thank Rishabh Agarwal, Yi Su, Young Geng, Abitha Thankaraj, Yuxiao Qu, So Yeon Min, Yutong He, Sukjun Hwang, Khurram Yamin, Charlie Snell, Zhang-Wei Hong, Kaylee Burns, Eric Mitchell, and other members of the CMU AI and Decision-Making Lab, CMU Russ Lab, CMU Auton Lab, Stanford IRIS Lab, and Stanford Ermon Group for discussions and feedback on an early version of this paper. AK thanks George Tucker and Sergey Levine for informative discussions. This research is supported by computational resources from Google TPU Research Cloud (TRC) and the National Science Foundation. FT thanks Ruslan Salakhutdinov for helpful suggestions during this project. AS gratefully acknowledges the support of the NSF Graduate Research Fellowship Program.

References

Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. Gkd: Generalized knowledge distillation for auto-regressive sequence models. *arXiv preprint arXiv:2306.13649*, 2023.

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

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952. ISSN 00063444. URL http://www.jstor.org/stable/2334029.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, Tony Tong Wang, Samuel Marks, Charbel-Raphael Segerie, Micah Carroll, Andi Peng, Phillip Christoffersen, Mehul Damani, Stewart Slocum, Usman Anwar, Anand Siththaranjan, Max Nadeau, Eric J Michaud, Jacob Pfau, Dmitrii Krasheninnikov, Xin Chen, Lauro Langosco, Peter Hase, Erdem Biyik, Anca Dragan, David Krueger, Dorsa Sadigh, and Dylan Hadfield-Menell. Open problems and fundamental limitations of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=bx24KpJ4Eb. Survey Certification.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-Play Fine-Tuning Converts Weak Language Models to Strong Language Models. *arXiv e-prints*, art. arXiv:2401.01335, January 2024. doi: 10.48550/arXiv.2401.01335.

ContextualAI. Human-centered loss functions (halos), 2024. URL https://github.com/ContextualAI/HALOs.

Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. Reward Model Ensembles Help Mitigate Overoptimization. *arXiv e-prints*, art. arXiv:2310.02743, October 2023. doi: 10.48550/arXiv.2310.02743.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. UltraFeedback: Boosting Language Models with High-quality Feedback. *arXiv e-prints*, art. arXiv:2310.01377, October 2023. doi: 10.48550/arXiv.2310.01377.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. Enhancing chat language models by scaling high-quality instructional conversations. *arXiv preprint arXiv:2305.14233*, 2023.

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, 2024.

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. *arXiv e-prints*, art. arXiv:2312.09244, December 2023. doi: 10.48550/arXiv.2312.09244.

Kawin Ethayarajh, Winnie Xu, Dan Jurafsky, and Douwe Kiela. Human-aware loss functions (halos). Technical report, Contextual AI, 2023. https://github.com/ContextualAI/HALOs/blob/main/assets/report.pdf.

Leo Gao, John Schulman, and Jacob Hilton. Scaling Laws for Reward Model Overoptimization. *arXiv e-prints*, art. arXiv:2210.10760, October 2022. doi: 10.48550/arXiv.2210.10760.

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A General Theoretical Paradigm to Understand Learning from Human Preferences. *arXiv e-prints*, art. arXiv:2310.12036, October 2023. doi: 10.48550/arXiv.2310.12036.

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

Jian Hu, Li Tao, June Yang, and Chandler Zhou. Aligning Language Models with Offline Learning from Human Feedback. *arXiv e-prints*, art. arXiv:2308.12050, August 2023. doi: 10.48550/arXiv.2308.12050.

Samyak Jain, Robert Kirk, Ekdeep Singh Lubana, Robert P. Dick, Hidenori Tanaka, Edward Grefenstette, Tim Rocktäschel, and David Scott Krueger. Mechanistically analyzing the effects of fine-tuning on procedurally defined tasks. *arXiv e-prints*, art. arXiv:2311.12786, November 2023. doi: 10.48550/arXiv.2311.12786.

Rahul Kidambi, Aravind Rajeswaran, Praneeth Netrapalli, and Thorsten Joachims. Morel: Model-based offline reinforcement learning. *arXiv preprint arXiv:2005.05951*, 2020.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

Robert Kirk, Ishita Mediratta, Christoforos Nalmpantis, Jelena Luketina, Eric Hambro, Edward Grefenstette, and Roberta Raileanu. Understanding the Effects of RLHF on LLM Generalisation and Diversity. *arXiv e-prints*, art. arXiv:2310.06452, October 2023. doi: 10.48550/arXiv.2310.06452.

Tomasz Korbak, Hady Elsahar, Germán Kruszewski, and Marc Dymetman. On reinforcement learning and distribution matching for fine-tuning language models with no catastrophic forgetting. *Advances in Neural Information Processing Systems*, 35:16203–16220, 2022.

Andrew Lee, Xiaoyan Bai, Itamar Pres, Martin Wattenberg, Jonathan K. Kummerfeld, and Rada Mihalcea. A Mechanistic Understanding of Alignment Algorithms: A Case Study on DPO and Toxicity. *arXiv e-prints*, art. arXiv:2401.01967, January 2024. doi: 10.48550/arXiv.2401.01967.

Ximing Lu, Sean Welleck, Jack Hessel, Liwei Jiang, Lianhui Qin, Peter West, Prithviraj Ammanabrolu, and Yejin Choi. Quark: Controllable text generation with reinforced unlearning. *Advances in neural information processing systems*, 35:27591–27609, 2022.

Jincheng Mei, Wesley Chung, Valentin Thomas, Bo Dai, Csaba Szepesvari, and Dale Schuurmans. The role of baselines in policy gradient optimization. *Advances in Neural Information Processing Systems*, 35:17818–17830, 2022.

Gabriel Mukobi, Peter Chatain, Su Fong, Robert Windesheim, Gitta Kutyniok, Kush Bhatia, and Silas Alberti. SuperHF: Supervised Iterative Learning from Human Feedback. *arXiv e-prints*, art. arXiv:2310.16763, October 2023. doi: 10.48550/arXiv.2310.16763.

Rémi Munos and Csaba Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9(5), 2008.

Rémi Munos, Michal Valko, Daniele Calandriello, Mohammad Gheshlaghi Azar, Mark Rowland, Zhaohan Daniel Guo, Yunhao Tang, Matthieu Geist, Thomas Mesnard, Andrea Michi, Marco Selvi, Sertan Girgin, Nikola Momchev, Olivier Bachem, Daniel J. Mankowitz, Doina Precup, and Bilal Piot. Nash Learning from Human Feedback. *arXiv e-prints*, art. arXiv:2312.00886, December 2023. doi: 10.48550/arXiv.2312.00886.

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, 2022.

Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th International Conference on Machine Learning*, pages 745–750. ACM, 2007.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. 2018.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*, 2023.

Corby Rosset, Ching-An Cheng, Arindam Mitra, Michael Santacroce, Ahmed Awadallah, and Tengyang Xie. Direct nash optimization: Teaching language models to self-improve with general preferences. *arXiv preprint arXiv:2404.03715*, 2024.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms. *arXiv e-prints*, art. arXiv:1707.06347, July 2017. doi: 10.48550/arXiv.1707.06347.

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

Archit Sharma, Sedrick Keh, Eric Mitchell, Chelsea Finn, Kushal Arora, and Thomas Kollar. A critical evaluation of ai feedback for aligning large language models. *arXiv preprint arXiv:2402.12366*, 2024.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in rlhf. *arXiv preprint arXiv:2310.03716*, 2023.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Adith Swaminathan and Thorsten Joachims. The self-normalized estimator for counterfactual learning. In *advances in neural information processing systems*, pages 3231–3239, 2015.

Gokul Swamy, Christoph Dann, Rahul Kidambi, Zhiwei Steven Wu, and Alekh Agarwal. A minimaximalist approach to reinforcement learning from human feedback. *arXiv preprint arXiv:2401.04056*, 2024.

Lewis Tunstall, Edward Beeching, Nathan Lambert, Nazneen Rajani, Kashif Rasul, Younes Belkada, Shengyi Huang, Leandro von Werra, Clémentine Fourrier, Nathan Habib, Nathan Sarrazin, Omar Sanseviero, Alexander M. Rush, and Thomas Wolf. Zephyr: Direct distillation of lm alignment, 2023.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, and Shengyi Huang. Trl: Transformer reinforcement learning. https://github.com/huggingface/trl, 2020.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SJeYe0NtvH.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, May 1992.

Annie Xie, Fahim Tajwar, Archit Sharma, and Chelsea Finn. When to ask for help: Proactive interventions in autonomous reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 35, 2022.

Denis Yarats, Rob Fergus, Alessandro Lazaric, and Lerrel Pinto. Mastering visual continuous control: Improved data-augmented reinforcement learning. *arXiv preprint arXiv:2107.09645*, 2021a.

Denis Yarats, Ilya Kostrikov, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. In *International Conference on Learning Representations*, 2021b. URL https://openreview.net/forum?id=GY6-6sTvGaf.

Tianhe Yu, Garrett Thomas, Lantao Yu, Stefano Ermon, James Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *arXiv preprint arXiv:2005.13239*, 2020.

Tianhe Yu, Aviral Kumar, Rafael Rafailov, Aravind Rajeswaran, Sergey Levine, and Chelsea Finn. Combo: Conservative offline model-based policy optimization. *Advances in neural information processing systems*, 34:28954–28967, 2021.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-Rewarding Language Models. *arXiv e-prints*, art. arXiv:2401.10020, January 2024. doi: 10.48550/arXiv.2401.10020.

Yao Zhao, Rishabh Joshi, Tianqi Liu, Misha Khalman, Mohammad Saleh, and Peter J. Liu. SLiC-HF: Sequence Likelihood Calibration with Human Feedback. *arXiv e-prints*, art. arXiv:2305.10425, May 2023. doi: 10.48550/arXiv.2305.10425.

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, 2020.

Appendices

A. Interpreting Existing Fine-Tuning Results

Our proposed framework also allows us to explain comparative evaluations in a number of existing LLM fine-tuning results, and as a result, implies several practical guidelines for RLHF practitioners. On the AlpacaFarm benchmark (Dubois et al., 2024), our results corroborate the gap between conditional supervised fine-tuning objectives such as binary FeedME and reward conditioning, and RL or contrastive training methods such as PPO and DPO: these results are perhaps even more extreme in that these conditional and weighted supervised fine-tuning objectives are not even able to outperform regular SFT. Methods that utilize on-policy sampling such as ReST (Gulcehre et al., 2023) and Quark (Lu et al., 2022) do outperform SFT, but still underperform on-policy RL or contrastive training. While the Best-of-N baseline is competitive with the top methods, it is prohibitively expensive. The top performing methods on the benchmark are DPO, which uses a negative-gradient objective, through maximizing reward with a reverse KL penalty, and PPO, which leverages on-policy sampling.

B. Computational vs Wall-Clock Time Tradeoffs for On-Policy Negative Gradient

A natural takeaway extending the empirical results from Section 5.3 is that on-policy variants of contrastive approaches might provide for an better tradeoff between computation and wall-clock time. We perform a comparison of wall-clock time needed to run our experiments in Table 3. in particular, we found that on-policy DPO only requires 0.4 hours to converge, while offline DPO requires a wall-clock time of 1.3 hours to converge to the same solution in the **Min Length** scenario. In the **Skew Length** scenario, where the learned policy must deviate from the initial reference policy substantially, we find that while offline DPO can converge a bit quickly (0.12 hours), it flatlines at a sub-optimal solution (completion length of 11.8) as compared to on-policy DPO which takes merely 0.4 hours to reach a more optimal solution. This is far more time-efficient compared to other on-policy methods such as PPO and RWR that present a sampling bottleneck.

	Bandit (R1)		**Min Length**		**Skew Length**	
	Reward (↑)	**Time**	**Completion Length (↓)**	**Time**	**Completion Length (↓)**	**Time**
Offline DPO / IPO	0.82 (0.04)	1.7 hours	1.0 (0.0)	1.3 hours	11.8 (14.0)	0.12 hours
On-policy PPO	0.92 (0.01)	0.93 hours	20.5 (25.4)	4.84 hours	15.8 (11.1)	7.26 hours
On-policy RWR	0.88 (0.01)	0.12 hours	65.5 (36.7)	15.5 hours	15.8 (9.3)	15.5 hours
On-policy DPO / IPO	0.92 (0.01)	0.12 hours	1.0 (0.0)	0.4 hours	0.0 (0.0)	0.4 hours

Table 3: **Wall-clock time comparisons.** Comparison between on-policy and offline variants of contrastive objectives (DPO/IPO) in terms of reward and wall-clock time required till convergence of the run. Generally, on-policy contrastive approaches achieve both superior reward and wall-clock time as opposed to offline contrastive approaches (offline DPO/IPO) and on-policy RL (PPO, RWR). Synthetic LLM experiments use a single A40 GPU. Bandit experiments use a Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz CPU, with 4 threads.

C. Theoretical Analysis

C.1. Unifying On-Policy Sampling and Negative Gradients via Mode Seeking

Here we provide proofs for the claims in 6.1. Our main goal is to show that the two main classes of algorithms we consider in this paper, namely on-policy RL methods and offline constrastive methods, both are mode-seeking as opposed to supervised maximum likelihood approaches, which are mode-covering. This conceptually explains the differences in their behaviors that we observe in our experiments.

C.1.1. On-policy RL Methods Are Mode-Seeking

First, we prove Lemma 6.1, i.e., we want to show that on-policy RL methods optimize regularized version of a reverse KL-divergence.

Proof. On-policy RL algorithms optimize some version of the following loss function:

$$\mathcal{L}_{\text{RL}}(\mathcal{D}_{\text{pref}}, \pi_\theta) = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}}[\mathbb{E}_{\boldsymbol{y}\sim\pi_\theta(.|\boldsymbol{x})}[r(\boldsymbol{x},\boldsymbol{y})] - \beta\mathbb{D}_{\text{KL}}[\pi_\theta(.|\boldsymbol{x})||\pi_{\text{ref}}(.|\boldsymbol{x})]] \tag{C.1}$$

Following Appendix A.1 of Rafailov et al. (2023), we have:

$$r(\boldsymbol{x},\boldsymbol{y}) = \beta \log Z(\boldsymbol{x}) + \beta \log \left(\frac{\pi^*(\boldsymbol{y}|\boldsymbol{x})}{\pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x})}\right) \tag{C.2}$$

where $Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x}) \exp\left(\frac{r(\boldsymbol{x},\boldsymbol{y})}{\beta}\right)$ is the partition function. Combining these two, we get:

$$\mathcal{L}_{\text{RL}}(\mathcal{D}_{\text{pref}}, \pi_\theta) = -\beta\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}}\left[\mathbb{E}_{\boldsymbol{y}\sim\pi_\theta(.|\boldsymbol{x})}\left[\log Z(\boldsymbol{x}) + \log\left(\frac{\pi^*(\boldsymbol{y}|\boldsymbol{x})}{\pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x})}\right)\right] - \mathbb{D}_{\text{KL}}[\pi_\theta(.|\boldsymbol{x})||\pi_{\text{ref}}(.|\boldsymbol{x})]\right] \tag{C.3}$$

$$= -\beta\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}}\left[\mathbb{E}_{\boldsymbol{y}\sim\pi_\theta(.|\boldsymbol{x})}\left[\log Z(\boldsymbol{x}) + \log\left(\frac{\pi^*(\boldsymbol{y}|\boldsymbol{x})}{\pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x})}\right)\right] - \mathbb{E}_{\boldsymbol{y}\sim\pi_\theta(.|\boldsymbol{x})}\left[\log\left(\frac{\pi_\theta(\boldsymbol{y}|\boldsymbol{x})}{\pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x})}\right)\right]\right] \tag{C.4}$$

$$= -\beta\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}}\left[\mathbb{E}_{\boldsymbol{y}\sim\pi_\theta(.|\boldsymbol{x})}\left[\log Z(\boldsymbol{x}) - \log\left(\frac{\pi_\theta(\boldsymbol{y}|\boldsymbol{x})}{\pi^*(\boldsymbol{y}|\boldsymbol{x})}\right)\right]\right] \tag{C.5}$$

$$= -\beta\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}}[\log Z(\boldsymbol{x})] + \beta\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}}[\mathbb{D}_{\text{KL}}[\pi_\theta(.|\boldsymbol{x})||\pi^*(.|\boldsymbol{x})]] \tag{C.6}$$

Note that $Z(\boldsymbol{x})$ does not depend on π_θ. Therefore, minimizing $\mathcal{L}_{\text{RL}}$ with respect to π_θ is equivalent to optimizing the reverse KL-divergence. □

Since optimizing the reverse KL-divergence is mode-seeking, we see that on-policy RL algorithms have mode-seeking behavior.

C.1.2. Contrastive Approaches (e.g., DPO/IPO) are Mode-Seeking

Next, we show that this is also the case for contrastive approaches as we prove Lemma 6.2.

Proof. □

Given Lemma 6.2, we also want to show that the gradient of the popular negative gradient based methods — DPO and IPO — takes the form from Eq. (6.1). From Rafailov et al. (2023), the gradient of the DPO loss is:

$$\nabla_\theta\mathcal{L}_{\text{DPO}}(\pi_\theta;\pi_{\text{ref}}) = -\beta\mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\sim\mathcal{D}_{\text{pref}}}\left[c^{\text{DPO}}(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\cdot[\nabla_\theta\log\pi_\theta(\boldsymbol{y}_w|\boldsymbol{x}) - \nabla_\theta\log\pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})]\right] \tag{C.7}$$

where $c^{\text{DPO}}(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l) = \sigma\left(\beta\log\frac{\pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})}{\pi_{\text{ref}}\boldsymbol{y}_l|\boldsymbol{x})} - \beta\log\frac{\pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})}{\pi_{\text{ref}}\boldsymbol{y}_w|\boldsymbol{x})}\right)$.

Now we derive the gradient of the IPO loss. Define

$$c^{\text{IPO}}(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l) = 2\cdot\left(\log\left(\frac{\pi_\theta(\boldsymbol{y}_w|x)}{\pi_{\text{ref}}(\boldsymbol{y}_w|x)}\right) - \log\left(\frac{\pi_\theta(\boldsymbol{y}_l|x)}{\pi_{\text{ref}}(\boldsymbol{y}_l|x)}\right) - \frac{\tau^{-1}}{2}\right) \tag{C.8}$$

Then the gradient of the IPO loss becomes:

$$\nabla_\theta \mathcal{L}_{\text{IPO}}(\pi_\theta; \pi_{\text{ref}}) = \nabla_\theta \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\sim\mathcal{D}_{\text{pref}}} \left[\left(\log \left(\frac{\pi_\theta(\boldsymbol{y}_w|x)\pi_{\text{ref}}(\boldsymbol{y}_l|x)}{\pi_{\text{ref}}(\boldsymbol{y}_w|x)\pi_\theta(\boldsymbol{y}_l|x)} \right) - \frac{\tau^{-1}}{2} \right)^2 \right] \tag{C.9}$$

$$= \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y}_w,\boldsymbol{y}_l)\sim\mathcal{D}_{\text{pref}}} \left[c^{\text{IPO}}(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l) \cdot [\nabla_\theta \log \pi_\theta(\boldsymbol{y}_w|x) - \nabla_\theta \log \pi_\theta(\boldsymbol{y}_l|x)] \right] \tag{C.10}$$

The IPO loss tries to regress the margin $\log \frac{\pi_\theta(\boldsymbol{y}_l|\boldsymbol{x})}{\pi_{\text{ref}}\boldsymbol{y}_l|\boldsymbol{x})} - \log \frac{\pi_\theta(\boldsymbol{y}_w|\boldsymbol{x})}{\pi_{\text{ref}}\boldsymbol{y}_w|\boldsymbol{x})}$ to $\frac{\tau^{-1}}{2}$. If the margin is larger than $\frac{\tau^{-1}}{2}$, then $c^{\text{IPO}}(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l) > 0$, and minimizing the loss pushes down $\log \pi_\theta(\boldsymbol{y}_w|x)$ and pushes up $\log \pi_\theta(\boldsymbol{y}_l)$. If the margin is smaller than $\frac{\tau^{-1}}{2}$, then the opposite thing happens.

C.1.3. Supervised Offline Algorithms are Mode-Covering

Now we prove Lemma 6.3, which shows that supervised offline methods that optimize a maximum likelihood loss exhibit mode-covering behavior.

Proof. Offline supervised methods optimize the following loss function:

$$\mathcal{L}_{\text{off-sup}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} \left[\sum_{\boldsymbol{y}} \pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x}) \log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) \cdot F(\boldsymbol{x}, \boldsymbol{y}) \right] \tag{C.11}$$

Define a new distribution

$$\tilde{\pi}(\boldsymbol{y}|\boldsymbol{x}) = \frac{\pi_{\text{ref}}(\boldsymbol{y}|\boldsymbol{x}) \cdot F(\boldsymbol{x}, \boldsymbol{y})}{Z(\boldsymbol{x})}$$

Here $Z(\boldsymbol{x}) = \sum_{\boldsymbol{z}} \pi_{\text{ref}}(\boldsymbol{z}|\boldsymbol{x}) \cdot F(\boldsymbol{x}, \boldsymbol{z})$ is the normalization constant. It is easy to check that this a valid conditional distribution.

This gives us:

$$\mathcal{L}_{\text{off-sup}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} \left[Z(\boldsymbol{x}) \sum_{\boldsymbol{y}} \tilde{\pi}(\boldsymbol{y}|\boldsymbol{x}) \log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) \right] \tag{C.12}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} \left[Z(\boldsymbol{x}) \cdot \mathbb{E}_{\boldsymbol{y}\sim\tilde{\pi}(\boldsymbol{y}|\boldsymbol{x})} \left[\log \left(\frac{\tilde{\pi}(\boldsymbol{y}|\boldsymbol{x})}{\pi_\theta(\boldsymbol{y}|\boldsymbol{x})} \right) \right] \right] - \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} \left[Z(\boldsymbol{x}) \cdot \mathbb{E}_{\boldsymbol{y}\sim\tilde{\pi}(\boldsymbol{y}|\boldsymbol{x})} [\log \tilde{\pi}(\boldsymbol{y}|\boldsymbol{x})] \right] \tag{C.13}$$

$$= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} [Z(\boldsymbol{x}) \cdot \mathbb{D}_{\text{KL}}(\tilde{\pi}(.|\boldsymbol{x})||\pi_\theta(.|\boldsymbol{x}))] + \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} [Z(\boldsymbol{x}) \cdot H(\tilde{\pi}(.|\boldsymbol{x}))] \tag{C.14}$$

Hence offline supervised methods minimize the weighted expectation of the forward KL-divergence $\mathbb{D}_{\text{KL}}(\tilde{\pi}(.|\boldsymbol{x})$ over $\boldsymbol{x} \sim \mathcal{D}_{\text{pref}}$. □

C.1.4. Supervised On-policy Objectives Are Mode-Seeking

Finally we prove **??**, which shows that KL regularized supervised algorithms such as RWR, ReST, BoN are mode-seeking if they calculate the loss function on samples collected from the current policy.

Proof. Supervised on-policy algorithms optimize the following objective:

$$\mathcal{L}_{\text{off-sup}}(\pi_\theta) = -\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pref}}} \left[\mathbb{E}_{\boldsymbol{y}\sim\pi_\theta(.|\boldsymbol{x})} [F(\boldsymbol{x}, \boldsymbol{y})] - \beta \mathbb{D}_{\text{KL}}[\pi_\theta(\cdot|\boldsymbol{x})||\pi_{\text{ref}}(.||\boldsymbol{x})] \right] \tag{C.15}$$

Note that this is exactly the same objective as Eq. (C.1), with $F(\boldsymbol{x}, \boldsymbol{y})$ as the reward function. Hence the proof of Lemma 6.1 works, and we are done.

□

C.2. Characterization of Gradients of Forward and Reverse-KL

For simplicity, let $\mathcal{X}$ be our input space, and $\mathcal{Y} = \{1, \ldots, V\}$ be the output space consisting of $V > 1$ discrete tokens. Let $f : \mathcal{X} \to \mathbb{R}^V$ be our network that outputs V real values logits for inputs $x \in \mathcal{X}$. With respect to the logits of f, we can define a probability distribution p over discrete tokens $1, \ldots, V$ using the softmax function, namely:

$$p_i(x) = \frac{\exp(f_i(x))}{\sum_{k=1}^{V} \exp(f_k(x))} \tag{C.16}$$

for any $x \in \mathcal{X}$.

Now assume that for some given input x, $q(x)$ is the true probability distribution over tokens that we want to learn. One way to do this would be to find p that minimizes $\mathbb{D}_{\text{KL}}(q||p)$ via SGD:

$$p_0(x) \leftarrow p_{\text{ref}}(x)$$

$$p_{t+1}(x) \leftarrow p_t(x) - \eta \nabla_{f(x)} \mathbb{D}_{\text{KL}}(q(x)||p(x))\big|_{p=p_t}$$

where p_{ref} is reference distribution we start with, and η is the learning rate. $\mathbb{D}_{\text{KL}}(q||p)$ is called the **forward-KL**, since q is the true distribution of interest. In contrast, $\mathbb{D}_{\text{KL}}(p||q)$ is called the **reverse-KL**, and one can try to optimize p by minimizing $\mathbb{D}_{\text{KL}}(p||q)$ in a similar fashion.

We shall now prove Lemma 6.4. We break this lemma in two parts. First, let us investigate how the gradients $\nabla_{f(x)} \mathbb{D}_{\text{KL}}(q(x)||p(x))\big|_{p=p_t}$ and $\nabla_{f(x)} \mathbb{D}_{\text{KL}}(p(x)||q(x))\big|_{p=p_t}$ look like:

Lemma C.1. *The gradients of forward and reverse KL looks like the following respectively:*

$$\frac{\partial}{\partial f_j} \mathbb{D}_{KL}(q(x)||p(x)) = p_j(x) - q_j(x) \tag{C.17}$$

$$\frac{\partial}{\partial f_j} \mathbb{D}_{KL}(p(x)||q(x)) = p_j(x) \left[\log \frac{p_j(x)}{q_j(x)} - \mathbb{D}_{KL}(p(x)||q(x)) \right] \tag{C.18}$$

Proof. We start with the definition of KL-divergence:

$$\begin{aligned}
\mathbb{D}_{\text{KL}}(q(x)||p(x)) &= \sum_i q_i(x) \log q_i(x) - \sum_i q_i(x) \log p_i(x) \\
&= -H(q) - \sum_i q_i(x) \log \left(\frac{e^{f_i(x)}}{\sum_k e^{f_k(x)}} \right) \\
&= -H(q) - \sum_i q_i(x) f_i(x) + \sum_i q_i(x) \log \left(\sum_k e^{f_k(x)} \right)
\end{aligned}$$

Therefore, we have:

$$\begin{aligned}\frac{\partial}{\partial f_j}\mathbb{D}_{\text{KL}}(q(x)||p(x)) &= -q_j(x) + \sum_i q_i(x)\left(\frac{e^{f_j(x)}}{\sum_k e^{f_k(x)}}\right) \\ &= -q_j(x) + \sum_i q_i(x)p_j(x) \\ &= p_j(x) - q_j(x)\end{aligned}$$

This proves Eq. (C.17). Similarly, we can write:

$$\begin{aligned}\mathbb{D}_{\text{KL}}(p(x)||q(x)) &= \sum_i p_i(x)\log p_i(x) - \sum_i p_i(x)\log q_i(x) \\ &= \sum_i \left(\frac{e^{f_i(x)}}{\sum_k e^{f_k(x)}}\right)\left[f_i(x) - \log\left(\sum_k e^{f_k(x)}\right)\right] - \sum_i \log q_i(x)\left(\frac{e^{f_i(x)}}{\sum_k e^{f_k(x)}}\right) \\ &= \frac{\sum_i f_i(x)e^{f_i(x)}}{\sum_k e^{f_k(x)}} - \left(\sum_i e^{f_i(x)}\right)\left(\frac{\log\left(\sum_k e^{f_k(x)}\right)}{\sum_k e^{f_k(x)}}\right) - \sum_i \log q_i(x)\left(\frac{e^{f_i(x)}}{\sum_k e^{f_k(x)}}\right) \\ &= \frac{\sum_i f_i(x)e^{f_i(x)}}{\sum_k e^{f_k(x)}} - \log\left(\sum_k e^{f_k(x)}\right) - \frac{\sum_i \log q_i(x)e^{f_i(x)}}{\sum_k e^{f_k(x)}}\end{aligned}$$

Now we calculate the partial derivative with respect to f_j: this mostly follows from standard rules of calculus.

$$\begin{aligned}\frac{\partial}{\partial f_j}\frac{\sum_i f_i(x)e^{f_i(x)}}{\sum_k e^{f_k(x)}} &= \frac{\left(\sum_k e^{f_k(x)}\right)\frac{\partial}{\partial f_j}\left(\sum_i f_i(x)e^{f_i(x)}\right) - \left(\sum_i f_i(x)e^{f_i(x)}\right)\left(\frac{\partial}{\partial f_j}\sum_k e^{f_k(x)}\right)}{\left(\sum_k e^{f_k(x)}\right)^2} \\ &= \frac{\frac{\partial}{\partial f_j}\left(\sum_i f_i(x)e^{f_i(x)}\right)}{\sum_k e^{f_k(x)}} - \frac{e^{f_j(x)}\left(\sum_i f_i(x)e^{f_i(x)}\right)}{\left(\sum_k e^{f_k(x)}\right)^2} \\ &= \frac{e^{f_j(x)} + f_j(x)e^{f_j(x)}}{\sum_k e^{f_k(x)}} - \frac{e^{f_j(x)}}{\sum_k e^{f_k(x)}}\left(\sum_i f_i(x)\left(\frac{e^{f_i(x)}}{\sum_k e^{f_k(x)}}\right)\right) \\ &= p_j(x) + f_j(x)p_j(x) - p_j(x)\left(\sum_i f_i(x)p_i(x)\right)\end{aligned}$$

Similarly,

$$\frac{\partial}{\partial f_j}\log\left(\sum_k e^{f_k(x)}\right) = \frac{e^{f_j(x)}}{\sum_k e^{f_k(x)}} = p_j(x)$$

And for the third term,

$$\begin{aligned}\frac{\partial}{\partial f_j}\frac{\sum_i \log q_i(x)e^{f_i(x)}}{\sum_k e^{f_k(x)}} &= \frac{\left(\sum_k e^{f_k(x)}\right)\frac{\partial}{\partial f_j}\left(\sum_i \log q_i(x)e^{f_i(x)}\right) - \left(\sum_i \log q_i(x)e^{f_i(x)}\right)\frac{\partial}{\partial f_j}\left(\sum_k e^{f_k(x)}\right)}{\left(\sum_k e^{f_k(x)}\right)^2} \\ &= \frac{e^{f_j(x)}\log q_j(x)}{\sum_k e^{f_k(x)}} - \left(\frac{e^{f_j(x)}}{\sum_k e^{f_k(x)}}\right)\left(\sum_i \log q_i(x)\frac{e^{f_i(x)}}{\sum_k e^{f_k(x)}}\right) \\ &= p_j(x)\log q_j(x) - p_j(x)\sum_i p_i(x)\log q_i(x)\end{aligned}$$

Putting it all together, we obtain:

$$\begin{aligned}\nabla_{f_j}\mathbb{D}_{\text{KL}}(p(x)||q(x)) &= p_j(x)\cdot(f_j(x)-\log q_j(x)) - p_j(x)\cdot\left(\sum_i p_i(x)\cdot(f_i(x)-\log q_i(x))\right)\\ &= p_j(x)\cdot\log\frac{p_j(x)}{q_j(x)} - p_j(x)\sum_i p_i(x)\cdot\log\frac{p_i(x)}{q_i(x)}\\ &= p_j(x)\left[\log\frac{p_j(x)}{q_j(x)} - \mathbb{D}_{\text{KL}}(p(x)||q(x))\right]\end{aligned}$$

completing our proof. □

C.3. Characterization of Committal Behavior

Eq. (C.17) and Eq. (C.18) gives us expressions for the derivative of the forward and reverse KL divergences respectively.

Now, if the logits f_t are being updated with gradient descent on loss $\mathcal{L}$, the distribution at the next step p^{t+1} is given by:

$$\begin{aligned}p_j^{t+1}(x) &= \exp(f_j^{t+1}(x))/\sum_i \exp(f_i^{t+1}(x))\\ &= \frac{\exp(f_j^t(x)) - \eta\nabla_{f_j^t}\mathcal{L})}{\sum_i \exp(f_i^t(x))}\cdot\frac{\sum_i \exp(f_i^t(x))}{\sum_i \exp(f_i^t(x) - \eta\nabla_{f_i^t}\mathcal{L})}\\ &= p_j^t(x)\cdot\frac{\exp\left(-\eta\nabla_{f_j^t}\mathcal{L}\right)}{\sum_i p_i^t(x)\exp\left(-\eta\nabla_{f_i^t}\mathcal{L}\right)}\end{aligned}$$

Let's consider what the characterization of p^{t+1} for the forward kl:

$$p_j^{t+1}(x) = p_j^t(x)\cdot\frac{\exp\left(-\eta\left(p_j^t(x) - q_j(x)\right)\right)}{\sum_i p_i^t(x)\exp\left(-\eta\left(p_i^t(x) - q_i(x)\right)\right)}$$

Noticing that the denominator is just a normalization constant, we can write this as:

$$\frac{p_j^{t+1}(x)}{p_j^t(x)} \propto \exp\left(-\eta\left(p_j^t(x) - q_j(x)\right)\right) \tag{C.19}$$

Similarly the characterization of p^{t+1} for the reverse KL looks like:

$$\frac{p_j^{t+1}(x)}{p_j^t(x)} \propto \exp\left(-\eta\left(p_j^t(x)\left[\log\frac{p_j^t(x)}{q^t(x)} - \mathbb{D}_{\text{KL}}(p^t(x)||q(x))\right]\right)\right) \tag{C.20}$$

This completes the proof of Lemma 6.4.

C.4. Quantifying the Differences Between Different Approaches

In this section we will prove Theorem 6.5: by doing so, we will study certain special cases to explain the differences between approaches that optimize the forward and reverse KL divergences.

Proof. □

D. Additional Algorithmic Details

D.1. Score/Reward Standardization

Online methods such as PPO or RWR that uses a learned reward model can suffer from gradient variance issues due to the differences in the reward score. In particular, adding or subtracting a baseline b from the reward $r_\phi(\boldsymbol{x}, \boldsymbol{y})$ does not change the relative order of preferred or dispreferred responses; however, it can change the variance of the gradients, leading to instability of the optimization routine. To mitigate this, prior work (Ziegler et al., 2020) often normalizes the reward to have zero mean and unit variance. This can be done during the training process by computing the mean and variance of the reward from an online batch. Formally, let $\{\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}\}_{i=1}^{\mathcal{B}}$ be a batch of data with batch size $\mathcal{B}$ sampled from policy π_θ: one calculates the standardized reward $\bar{r}_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$ as:

$$\bar{r}_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) = \frac{r_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}) - \hat{\mu}}{\hat{\sigma}} \tag{D.1}$$

where $\hat{\mu} = \frac{1}{\mathcal{B}} \sum_{i=1}^{\mathcal{B}} r_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})$, $\hat{\sigma} = \sqrt{\frac{1}{\mathcal{B}-1} \sum_{i=1}^{\mathcal{B}} (r_\phi(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})^2 - \hat{\mu})^2}$.

D.2. IPO

IPO (Gheshlaghi Azar et al., 2023) is similar to DPO in the sense that it also tries to minimize the implicit reward of the dispreferred responses, $\log \pi_\theta(\boldsymbol{y}_l|\boldsymbol{x}) - \log \pi_{\text{ref}}(\boldsymbol{y}_l|\boldsymbol{x})$ and maximize the reward of the preferred responses, $\log \pi_\theta(\boldsymbol{y}_w|\boldsymbol{x}) - \log \pi_{\text{ref}}(\boldsymbol{y}_w|\boldsymbol{x})$. The key difference is the loss fucntion: DPO optimizes the negative log-sigmoid loss whereas IPO optimizes an MSE-type objective. Formally, the IPO objective is:

$$\mathcal{L}_{\text{IPO}}(\pi_\theta; \pi_{\text{ref}}) = \mathbb{E}_{(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l) \sim \mathcal{D}_{\text{pref}}} \left(\log \left(\frac{\pi_\theta(\boldsymbol{y}_w|x) \pi_{\text{ref}}(\boldsymbol{y}_l|x)}{\pi_{\text{ref}}(\boldsymbol{y}_w|x) \pi_\theta(\boldsymbol{y}_l|x)} \right) - \frac{\tau^{-1}}{2} \right)^2 \tag{D.2}$$

where τ is a hyperparameter controlling how much the learned policy π_θ deviates from the reference policy π_{ref}.

E. Method Hyperparameters

We did an extensive sweep over hyperparamaters for individual offline and online algorithms for the language model experiments. We built our algorithm implementations off of the Huggingface TRL implementation (von Werra et al., 2020).

E.1. Standardized Parameters (Consistent for all Methods)

Table 4: Algorithm Agnostic Hyperparamters

Hyperparameters	Values	Description
B	64	Batch Size
B_{mini}	8	Mini-Batch Size
G	8	Gradient Accumulation Steps
$\hat{\pi}_\theta$	Pythia1.4B, Mistral-7b	Policy Architecture
$\hat{R}_\theta$	Pythia410M, Mistral-7B	Reward Model Architecture
optimizer	Adam	Gradient Optimizer

Table 5: Sampling Hyperparamters

Hyperparameters	Values	Description
top_k	0.0	Disables top-k sampling
top_p	1.0	Disables nucleus sampling
do_sample	True	Enables sampling
max_new_tokens	256	Maximum number of new tokens to generate
temperature	1.0	Sets sampling temperature (1.0 for default)
use_cache	True	Uses past key/values attentions if supported by the model

E.2. DPO (Rafailov et al., 2023)

Table 6: DPO Hyperparameters

Hyperparameters	Values	Description
lr	1e-7, 5e-7, 1e-6, 5e-6, 1e-5	learning rate
β	0.01, 0.05, 0.1, 0.5	KL weight

E.3. Pref-FT (Dubois et al., 2024)

Table 7: Pref-FT/Binary FeedMe Hyperparameters

Hyperparameters	Values	Description
η	1e-7, 5e-7, 1e-6, 5e-6	learning rate

E.4. PPO (Schulman et al., 2017)

Table 8: PPO Hyperparameters

Hyperparameters	Values	Description
η	1e-7, 5e-7, 1e-6, 5e-6, 1e-5	Learning rate.
vf_coef	0.1	Coefficient for the value function loss.
adap_kl_ctrl	True	Enables adaptive KL penalty control.
init_kl_coef	0.2	Initial coefficient for KL penalty.
target_kl	0.1	Target KL divergence for policy updates.
N	1	actions per prompt

E.5. RWR

Table 9: RWR Hyperparameters

Hyperparameters	Values	Description
η	1e-7, 5e-7, 1e-6, 5e-6, 1e-5	learning rate
β	0.1, 1, 10, 20	temperature
N	1	actions per prompt

E.6. Iterated BofN (Mukobi et al., 2023)

Table 10: Iterated BofN Hyperparameters

Hyperparameters	Values	Description
η	1e-7, 5e-7, 1e-6, 5e-6, 1e-5	learning rate
N	4, 10	actions per prompt

F. Code For Running Experiments

We have made the code for this project public in this repository. The additional datasets used in our experiments are listed below:

- Min Length
- Mode Length
- Skew Length
- Relabelled AlpacaFarm

We gratefully acknowledge the following codebases: TRL (von Werra et al., 2020), HALOs (Ethayarajh et al., 2023), DrQ-v2 (Yarats et al., 2021a,b) and PAINT Xie et al. (2022).

G. More on Didactic Bandit Problems

G.1. Problem Setup

Here we present details of our didactic bandit problem. The reference policy shown in Figure 2 is obtained by collecting 10000 samples from a Cauchy distribution with location $x_0 = -0.7$, scale $\gamma = 0.4$. Next, we clip this samples between the interval $(-1, 1)$, and divide the interval into 100 equally spaced bins. Starting from -1, we label these bins $0, \ldots, 99$ sequentially, and calculate the frequency of samples that fell into each bin. Finally, we define,

$$\pi_{\text{ref}}(a_i) = \frac{\text{Freq}(bin_i)}{10000}$$

The reward functions $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are defined as:

$$\boldsymbol{R}_1(a) = \exp\left(-\left(\frac{a-70}{10}\right)^2\right)$$

and

$$\boldsymbol{R}_2(a) = \exp\left(-\left(\frac{a-20}{10}\right)^2\right)$$

G.2. Algorithmic Details

In the bandit setting, we consider five algorithms: (1) Best-of-N, (2) IPO, (3) REINFORCE, (4) PPO and (5) RWR.

G.2.1. Best-of-N

Best-of-N is similar to SuperHF Mukobi et al. (2023)/ReST Gulcehre et al. (2023) and in some way their simplification for the bandit setting. Best-of-N collects N actions/responses for a prompt/state $\boldsymbol{x}$, namely $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N$. Next, we collect the rewards $\{\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}_i)\}_{i=1}^N$, and based on these rewards, choose the best action $\boldsymbol{y}_{\text{best}} = \arg\max_{\boldsymbol{y}_i} \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}_i)$. Finally, the loss function is the negative log-likelihood of this best action.

$$\mathcal{L}_{\text{bofn}}(\pi_\theta; \boldsymbol{x}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) = -\log \pi_\theta(\boldsymbol{y}_{\text{best}}|\boldsymbol{x})$$

In both the online and offline setting, we have a fixed set of prompts $\mathcal{D}_{\text{prompts}}$, and we also always start with π_θ initialized to π_{ref}. Formally, given a policy π, we can form a training set as:

$$\mathcal{D}_{\text{train}}(\mathcal{D}_{\text{prompts}}, \pi) = \{(\boldsymbol{x}, \boldsymbol{y}) : \boldsymbol{x} \in \mathcal{D}_{\text{prompts}}, \boldsymbol{y} = \arg\max_{\boldsymbol{y}_i} \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}_i) \text{ where } \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N \sim \pi_{\text{ref}}(.|\boldsymbol{x})\}$$

In the offline setting, we collect a fixed training dataset where actions are sampled from π_{ref}, namely $\mathcal{D}_{\text{train}}(\mathcal{D}_{\text{prompts}}, \pi_{\text{ref}})$. In the online setting, we collect a new training dataset by sampling actions from the current policy π_θ, namely $\mathcal{D}_{\text{train}}(\mathcal{D}_{\text{prompts}}, \pi_\theta)$, after every T gradient steps, and discard the previous dataset.

To show the efficacy of negative gradient, we can also directly add a term to this loss function minimizing log probability on less preferred actions. Explicitly, we consider the following loss function:

$$\mathcal{L}_{\text{bofn + neg-grad}}(\pi_\theta; \boldsymbol{x}, \boldsymbol{y}_1, \ldots, \boldsymbol{y}_N) = -\log \pi_\theta(\boldsymbol{y}_{\text{best}}|\boldsymbol{x}) + \beta \sum_{\boldsymbol{y}_j \neq \boldsymbol{y}_{\text{best}}} \log \pi_\theta(\boldsymbol{y}_j|\boldsymbol{x})$$

where β is a hyperparameter that we usually set to 1.0. We note that in practice this loss can quickly become unstable and proceed to $-\infty$, in practice we only minimize probability on less preferred action if it is above a certain threshold.

G.2.2. IPO

In contrast, IPO uses the loss function defined in Eq. (D.2). While regular IPO is a offline algorithm that uses a fixed preference dataset $\mathcal{D}_{\text{pref}}$, since we have access to the true reward function in the bandit setup, we create an online version of this algorithm as well. Here we also have a fixed set of prompts $\mathcal{D}_{\text{prompts}}$, and given a policy π, we can generate a preference dataset as follows: for each prompt $\boldsymbol{x} \in \mathcal{D}_{\text{prompts}}$, we can generate completions $\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_N \sim \pi(.|\boldsymbol{x})$. For any $i \neq j$, without loss of generality, assume $\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}_i) > \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y}_j)$. Then $\boldsymbol{y}_i$ and $\boldsymbol{y}_j$ are the preferred and dispreferred completions respectively, and we can form a preference dataset with all such $(\boldsymbol{x}, \boldsymbol{y}_w, \boldsymbol{y}_l)$ tuples.

In the offline setting, the preference dataset is collected by generating samples from the reference policy π_{ref}, and kept fixed during training. In the online setting, we generate the preference dataset from the current policy π_θ, after every T gradient steps, and discard the previous dataset.

G.2.3. REINFORCE

For REINFORCE, we sample $\boldsymbol{y} \sim \pi_\theta(.|\boldsymbol{x})$, calculate the normalized reward $\overline{\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y})}$, and use the following loss:

$$\mathcal{L}_{\text{REINFORCE}}(\pi_\theta; \mathcal{D}_{\text{prompts}}) = -\mathbb{E}_{\boldsymbol{x} \in \mathcal{D}_{\text{prompts}}}[\mathbb{E}_{\boldsymbol{y} \sim \pi_\theta}[\log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) \overline{\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y})}]]$$

G.2.4. PPO

For PPO, let π_{gen} be the policy used to generate the responses, and define $r(\boldsymbol{x}, \boldsymbol{y}) = \frac{\pi_\theta(\boldsymbol{y}|\boldsymbol{x})}{\pi_{\text{gen}}(\boldsymbol{y}|\boldsymbol{x})}$. Then we use the following loss function:

$$\mathcal{L}_{\text{PPO}}(\pi_\theta; \mathcal{D}_{\text{prompts}}) = -\mathbb{E}_{\boldsymbol{x} \in \mathcal{D}_{\text{prompts}}}\left[\mathbb{E}_{\boldsymbol{y} \sim \pi_\theta}\left[\max\left(r(\boldsymbol{x}, \boldsymbol{y})\overline{\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y})}, \text{Clip}\left(r(\boldsymbol{x}, \boldsymbol{y}), 1-\epsilon, 1+\epsilon\right)\overline{\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y})}\right)\right]\right]$$

where $\epsilon > 0$ is a hyperparameter that controls how much we clip off-policy updates.

G.2.5. RWR

For RWR, we use the following loss function:

$$\mathcal{L}_{\text{REINFORCE}}(\pi_\theta; \mathcal{D}_{\text{prompts}}) = -\mathbb{E}_{\boldsymbol{x} \in \mathcal{D}_{\text{prompts}}}\left[\mathbb{E}_{\boldsymbol{y} \sim \pi_\theta}\left[\log \pi_\theta(\boldsymbol{y}|\boldsymbol{x}) \exp\left(\frac{\overline{\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{y})}}{\beta}\right)\right]\right]$$

where β is a hyperparameter, usually $\beta = 0.1$ in our experiments unless otherwise noted.

G.3. Experiment Details

For all experiments, we use $N = 10$. For negative gradient experiments, we are in the fully offline setting, and vary the size of the prompt dataset $\mathcal{D}_{\text{prompts}}$, with $T = 100$ number of gradient steps performed. For on policy sampling experiments, we hold $|\mathcal{D}_{\text{prompts}}| = 10$ randomly sampled prompts from tokens $\{0, \ldots, 99\}$, and vary T. We also a new training dataset from the current policy after each T gradient steps, and perform this data collection step 100 times for all experiments. We set $\tau = 0.05$ for IPO, and search for the optimal learning rate from 0.3, 0.1, 0.03, 0.01, 0.003, 0.001, 0.0003, 0.0001, 0.00003 and 0.00001 for each experiment and use an Adam (Kingma and Ba, 2017) optimizer for all experiments. We run each experiment for 5 seeds, and the shaded region in the plots refer to the standard error of the mean obtained from these runs. Finally, to initialize π_θ to π_{ref}, we minimize the KL divergence between π_θ and π_{ref} with an Adam optimizer with learning rate 0.01.

For all experiments, we use a small GPT Radford et al. (2018); Brown et al. (2020)-like transformer architecture (named 'GPT-Nano') with 0.9M parameters. We took the implementation from this public repository: https://github.com/karpathy/minGPT.