RL4LLM: A TECHNICAL TRAJECTORY FROM ALIGNMENT TO REASONING

A PREPRINT

Haiquan Wang
Chengdu Institute of Computer Applications, Chinese Academy of Sciences
University of Chinese Academy of Sciences
wanghaiquan22@mails.ucas.ac.cn

August 20, 2025

ABSTRACT

Reinforcement Learning from Human Feedback (RLHF), particularly since the application of the Proximal Policy Optimization (PPO) algorithm, has become a core paradigm for enhancing the capabilities of Large Language Models (LLMs). However, the classic RLHF pipeline, exemplified by PPO, relies on complex online sampling and multiple separate models, leading to high training costs and process instability. To address these issues, Direct Preference Optimization (DPO) and its variants, such as Identity Preference Optimization (IPO), have been proposed. Through an elegant mathematical transformation, they convert the complex reinforcement learning problem into an efficient, supervised-like loss function, thereby bypassing the explicit modeling of a reward model. Furthermore, to tackle the challenges of reasoning-intensive tasks, Group Relative Policy Optimization (GRPO) was introduced, exploring an alternative Reinforcement Learning for LLM (RL4LLM) path by replacing the value model with intra-group reward normalization. Although these methods are continuously evolving, their theoretical underpinnings are deeply intertwined. This paper provides a systematic, first-principles review of this key technical trajectory from PPO to DPO and GRPO. We deeply analyze the core motivations and optimization objectives of each algorithm, complete with detailed mathematical derivations, offering a clear and insightful analytical perspective for understanding and applying state-of-the-art LLM alignment techniques.

Keywords Large Language Models · Reinforcement Learning from Human Feedback · Reasoning model · RL4LLM

1 Introduction

The advent of GPT-3 [Brown et al., 2020] signaled that the pre-training of Large Language Models (LLMs) is a promising trajectory towards Artificial General Intelligence (AGI). While GPT-3, upon completion of its pre-training, possesses a vast repository of internal knowledge, adapting these pre-trained models to perform effectively in specific engineering and application scenarios became a pressing research challenge. Although post-training techniques such as templates and Instruction Fine-Tuning (IFT) exist to adapt models for specific downstream tasks, their application is often limited in scope, preventing the pre-trained model from being considered a truly versatile, user-facing product.

To address this challenge, OpenAI pioneered Reinforcement Learning from Human Feedback (RLHF) [Ouyang et al., 2022]. Through a two-phase post-training process (a preparation phase and a reinforcement learning phase), RLHF enables pre-trained models to achieve robust performance across a wide range of general-purpose tasks, dramatically improving both their capabilities in downstream scenarios and the user's conversational experience. This led to the creation of InstructGPT, which was subsequently evolved into the GPT-3.5 series and ChatGPT, marking the transition from a research artifact to a mature, widely-used application.

The reinforcement learning (RL) stage of RLHF employs the Proximal Policy Optimization (PPO) [Schulman et al., 2017] algorithm, a member of the actor-critic family. A core challenge in PPO is that the objective function—the expected cumulative reward—cannot be directly differentiated. This is because the expectation is computed over

trajectories generated by stochastic sampling from the policy network, making it impossible to establish a direct gradient flow from the rewards back to the network parameters θ. To overcome this, PPO leverages the Policy Gradient Theorem [Sutton et al., 1999] to enable gradient backpropagation. Concurrently, the standard PPO pipeline is notoriously complex, requiring online sampling and the coordination of four distinct models: a policy model, a value model, a reward model, and a reference model. This complexity results in high training costs and instability. It is in response to these limitations that Direct Preference Optimization (DPO) emerged.

The breakthrough of DPO [Rafailov et al., 2023] lies in its elegant mathematical derivation, which demonstrates that the RLHF objective of maximizing rewards subject to a Kullback-Leibler (KL) divergence constraint can be equivalently transformed into a simple pairwise ranking loss function, optimized directly on human preference data. This clever reformulation bypasses the need for an explicit reward model and the entire online RL sampling loop, reframing the complex RL problem as a maximum likelihood estimation problem, akin to supervised learning. Consequently, DPO substantially simplifies the training pipeline, reducing the number of required models from four to two (the policy and reference models), which significantly lowers both computational cost and implementation complexity. Since its introduction, numerous DPO-based optimization algorithms have emerged, a selection of which will be discussed in this paper.

While the RLHF paradigm has been widely adopted by mainstream large models, an alternative trajectory has been proposed by DeepSeek, specifically tailored for enhancing the model's deep reasoning capabilities. In their work on DeepSeek-Math, they introduced a novel RL algorithm named Group Relative Policy Optimization (GRPO) [Shao et al., 2024]. Whereas PPO relies on a learned value model as a baseline, GRPO bypasses explicit value modeling. Instead, it generates multiple responses for a single prompt and models the advantage function by normalizing the rewards within this group. It further employs rejection sampling to mitigate reward hacking. Ultimately, the resulting model, DeepSeek-R1, trained from DeepSeek-v3-base, has demonstrated remarkable reasoning abilities.

This paper provides a technical survey of these pivotal RL4LLM algorithms, using the evolution around GPT-like models as a central narrative. To this end, Section 2 introduces foundational concepts, such as the advantage function, to lay the groundwork for subsequent technical analysis. Section 3 offers an intuitive overview of the PPO algorithm. Section 4 provides a detailed examination of the complete RLHF pipeline and the intricacies of PPO from a research trajectory perspective. Following this, Section 5 analyzes the DPO algorithm, and Section 6 introduces the GRPO algorithm.The final Section will provide a brief summary and outlook of these algorithms.

2 Preliminaries

2.1 The Advantage Function (A)

In policy gradient methods, the return R_t is commonly used to evaluate the quality of an action. However, the return R_t suffers from high variance. Even for a good policy, a single trajectory sampled from a given state might yield a poor R_t due to inherent stochasticity. Using the high-variance R_t directly as a gradient signal can cause severe oscillations during training, making convergence difficult. Furthermore, in practical reinforcement learning scenarios, reward signals can be categorized as immediate, conditional, or terminal. The latter two types are often sparse, which further exacerbates the variance of R_t.

The advantage function, $A(s, a)$, re-evaluates the "excess" quality of an action by subtracting an action-independent baseline—typically the state-value function $V(s_t)$—from the return R_t:

$$A(s_t, a_t) \approx R_t - V(s_t) \tag{1}$$

This operation significantly reduces the variance of the gradient. By acting as an average expectation, $V(s_t)$ filters out a substantial portion of the value fluctuations inherent to the state itself. This allows $A(s_t, a_t)$ to provide a purer measure of the intrinsic merit of the action a_t. Consequently, the advantage function is a key technique for making policy gradient training more stable and efficient.

2.2 GPT-3, InstructGPT, GPT-3.5, and ChatGPT

GPT-3, introduced in OpenAI's 2020 paper "Language Models are Few-Shot Learners" [Brown et al., 2020], was a model of unprecedented scale. It featured a staggering 175 billion parameters—approximately 500 times that of BERT-large—and was trained on 45 terabytes of data. This immense scale endowed it with a vast repository of knowledge, providing the infrastructural foundation that allowed it to remain impressive even years later. The advent of GPT-3 pioneered numerous research directions and effectively ushered in the era of LLMs. Despite its monumental value in academic research, its practical application in downstream tasks still required IFT, which presented significant technical barriers and costs.

InstructGPT was the first model with "chat capabilities" built upon GPT-3 through the application of RLHF [Ouyang et al., 2022]. This technique endowed the LLM with superior instruction-following abilities and enabled it to generate content that better aligns with human values.

GPT-3.5 represents a family of dialogue models that applied the RLHF methodology at scale, incorporating a series of process optimizations. As the official and upgraded successor to InstructGPT, it inherited and advanced the core ideas of RLHF and served as the engine for the initial version of ChatGPT. The GPT-3.5 series was refined into various versions, such as `gpt-3.5-turbo`, which was specifically optimized for dialogue, offering lower costs and faster speeds. Its success also spurred the open-source community to synthesize data based on its outputs, leading to the creation of many well-known datasets and research directions, such as ShareGPT and Reinforcement Learning from AI Feedback (RLAIF).

ChatGPT is a product, an application. It provides a user interface (a dialogue box) that allows general users to conveniently interact with the powerful language model operating behind the scenes, which was initially the GPT-3.5 family.

(From Gemini 2.5 Pro) To offer an analogy: GPT-3 is like a classic, powerful V8 muscle car engine—immensely potent but difficult to control, fuel-guzzling, and untuned. InstructGPT is the prototype where this V8 engine is equipped with electronic fuel injection and an engine control unit (ECU); through RLHF, this "beast" is fitted with a sophisticated control system, making it not only powerful but also controllable, efficient, and safe. GPT-3.5 is the mass-produced, modern turbocharged engine based on this prototype's technology. It inherits all the advantages of the prototype and, through manufacturing optimizations, achieves lower costs, more stable performance, and is better suited for large-scale assembly. Finally, ChatGPT is the complete "family sedan or SUV" equipped with this modern engine—a complete, user-oriented product that includes not only a powerful engine but also comfortable seats and a comprehensive safety system, allowing anyone to easily and safely enjoy the benefits of the technology.

2.3 The Policy Gradient Theorem

In supervised learning, the objective is to make the model's output, y_{pred}, approximate the ground truth, y_{true}. This is achieved by defining a differentiable loss function, such as Mean Squared Error (MSE), $L_{MSE} = (y_{pred} - y_{true})^2$, to measure the discrepancy between them. The goal is to minimize this loss. Since the loss function is differentiable, one can directly compute its gradient with respect to the model parameters θ, i.e., $\frac{\partial L}{\partial \theta}$, and use gradient descent to optimize the model. This method of gradient backpropagation is a cornerstone of modern neural networks.

In contrast, the objective of RL is to maximize the expected cumulative reward, defined as $J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)]$. The reward signal provided by the environment (or a reward model) is a scalar, and the environment itself is typically a non-differentiable black box. Therefore, standard gradient descent cannot be applied directly to this objective.

The Policy Gradient Theorem provides a bridge between this non-differentiable objective and the differentiable model parameters. It proves that the gradient of the objective, which is what we want to maximize, can be expressed as a computable expectation. While the expected return $J(\theta)$ is not directly differentiable with respect to θ, its gradient $\nabla J(\theta)$—the direction for updating model parameters—can be transformed into an expectation that depends only on the policy π_θ itself: $\nabla J(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t]$. This leads to a simple loss function, often expressed as:

$$L^{PG}(\theta) = -(\log \pi_\theta(a_t|s_t))\hat{A}_t \tag{2}$$

where $\hat{A}_t$ is the advantage function. With this transformation, the optimization target shifts from a non-differentiable scalar to a differentiable loss function with respect to θ. This is often referred to as a "pseudo-loss."

The intuition behind this theorem is that, although we do not know the "correct" action, we can provide the model with a gradient signal for parameter updates through trial-and-error and post-hoc credit assignment. The optimization process is actually gradient ascent. While an optimizer executes $\theta \leftarrow \theta - \alpha \nabla_\theta L$, substituting the gradient of Eq. (2) results in $\theta \leftarrow \theta + \alpha \hat{A}_t \nabla_\theta \log \pi_\theta(a_t|s_t)$, which is equivalent to performing gradient ascent on the policy gradient in the desired direction.

2.4 Probability, Likelihood, MLE, and Network Training

The concepts of probability, likelihood, and Maximum Likelihood Estimation (MLE) appear frequently in this field. It is therefore necessary to clarify these concepts and their relationship to network training.

Probability Knowing the probability of an event allows one to predict the likelihood of various outcomes. This is reasoning **from cause to effect**.

Likelihood Having observed a result, one can infer which model parameters were most likely to have produced it. This is reasoning **from effect back to cause**.

Maximum Likelihood Estimation (MLE) Given a set of known observations, MLE is a systematic mathematical method for finding the set of model parameters that most likely generated these observations. This is a formalized approach of reasoning **from effect to cause**.

Network Training The training of a neural network, particularly when using a loss function derived from or equivalent to MLE, aims to adjust the network's parameters so that its internal data distribution approximates the overall probability distribution of the training dataset.

2.5 Notation Clarification (for Policy-Based Methods in this Paper)

- r_t **(Reward)** The immediate, single-step benefit received right after executing an action.
- R_t **(Return)** The cumulative return; the discounted sum of all future rewards from the current time step onward.
  - $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$
- $Q(s_t, a_t)$ **(Action-Value)** The expected return after taking action a_t in state s_t.
  - Analogy: At an intersection (state), what are the respective future returns of turning left versus turning right (actions)?
  - It is formally defined by the Bellman equation, $Q(s_t, a_t) = \mathbb{E}[r_t + \gamma V(s_{t+1})]$. In practice, it is often approximated using a single-step estimate: $Q(s_t, a_t) \approx r_t + \gamma V(s_{t+1})$.
- $V(s_t)$ **(State-Value)** The expected return starting from the current state s_t.
  - Analogy: Starting from the current intersection (state), what is the expected return for the entire journey ahead?
  - $V(s) = \mathbb{E}[R_t | s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots | s_t = s]$. During training, the value network is trained to approximate the observed return R_t.

3 An Intuitive Understanding of PPO

This chapter will first analyze the complete RLHF process from the perspective of PPO, introducing the key components of the PPO algorithm. Then, guided by Figure 1, we will provide a detailed analysis of the complete forward pass, intermediate computation, and backward pass of the PPO algorithm. This section intentionally simplifies the technical details to focus more on the conceptual significance of each component.

3.1 High-Level Methodology

In the InstructGPT paper (Figure 2), RLHF is defined as a three-step process: training an initial policy (SFT), training a reward model, and the RL phase. Since the core of RLHF is the PPO algorithm, this paper re-categorizes the stages of RLHF from a PPO-centric perspective: the PPO Preparation Phase, the PPO Phase, and the PPO Finalization Phase. This section focuses on the role of each technique rather than its specific implementation details.

3.1.1 PPO Preparation Phase (Steps 1 & 2)

SFT Model Training Starting from a base model (e.g., GPT-3), the Supervised Fine-Tuning (SFT) model is trained on a dataset of collected dialogue demonstrations (instructions). This enables it to understand and follow specific instruction formats (templates) and perform tasks via in-context learning (ICL). The training objective itself is identical to pre-training: next-token prediction. The loss function is the cross-entropy loss derived from maximum likelihood:

$$L(\theta) = -\sum_{t=1}^{T} \log P(x_t | x_{<t}; \theta) \tag{3}$$

Reward Model (RM) Training Starting from a base model (or a smaller variant, or even the SFT model itself), the RM is trained on preference data. This involves sampling different responses to the same prompt from the SFT model and having them ranked by human annotators. The core task is to learn this preference ranking, with the ultimate goal

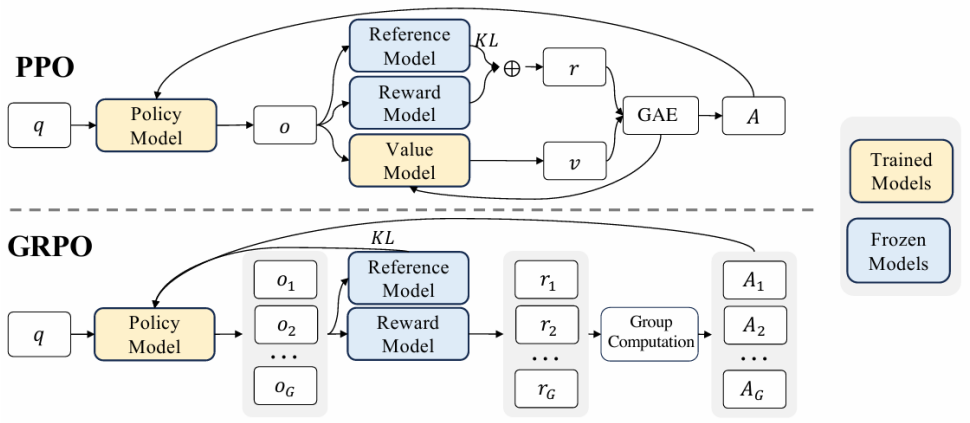

Figure 1: This figure is from the DeepSeek-Math research paper [Shao et al., 2024] and illustrates the high-level processes of PPO and GRPO. The discussion in this section follows the PPO workflow depicted here. In the diagram, blue components represent frozen models during PPO training, while yellow components are trainable models. Here, 'q' denotes the input query, 'o' the model's output, 'r' the final reward (computed using the reward model and a KL penalty from the reference model), 'v' the value estimate from the value model, 'GAE' refers to Generalized Advantage Estimation, and 'A' represents the resulting advantage estimate.

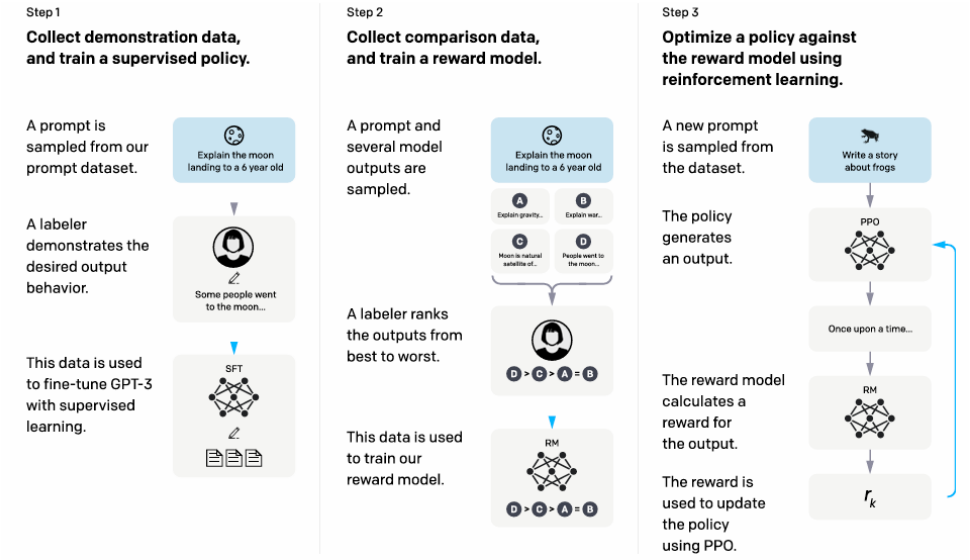

Figure 2: This figure is from the InstructGPT research paper [Ouyang et al., 2022] and shows the complete RLHF process.

of outputting a scalar reward. The loss function is the Bradley-Terry Pairwise Ranking Loss. Its key idea is not to regress to an absolute score, but to ensure that for the same prompt, the reward for the chosen response is higher than for the rejected one: $R_{RM}(\text{chosen}) > R_{RM}(\text{rejected})$. It optimizes the **difference** between reward values, not their absolute magnitudes.

3.1.2 PPO Phase (Online Learning, Step 3)

Policy Model (Actor) Initialized from the SFT model, it generates responses to prompts from the current phase's dataset and learns to align with human preferences based on the advantage estimate $\hat{A}_t$ produced by GAE. Its loss function is the pseudo-loss derived from the Policy Gradient Theorem, a simple form of which is:

$$L_{PPO}^{PG}(\theta) = -\hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right] \tag{4}$$

where $\hat{A}_t$ is the estimate of the advantage A_t calculated using the GAE method.

Reward Model (Judge) This is the trained RM from the preceding stage, which is kept **frozen**. Its sole task is to provide a scalar reward for the policy model's output. It has no loss function in this phase.

Reference Model This is simply the original SFT model, also kept **frozen**. Its purpose is to constrain the policy model via KL divergence, which helps prevent the policy from "reward hacking" by deviating too far from the initial, safe distribution. It has no loss function.

Value Model (Critic) Initialized from the RM of the previous stage (or the base model), its input is the prompt combined with the policy model's output. Its task is to predict the expected return given the output from the current policy model, aiming to approximate the actual return R_t. Together with the reward model's output, it is used to compute the advantage estimate $\hat{A}$ via GAE. Its loss function is typically a Mean Squared Error (MSE):

$$L_{MSE}^{value} = (R_t - V(s_t))^2 \tag{5}$$

3.1.3 PPO Finalization Phase (Final Output)

LLM-Chat This is the policy model at the end of the RL stage. At this point, the LLM transitions from a base model to a true Chat model and becomes the final, deliverable dialogue model. (The chat model variants in open-source LLM families often have a 'chat' or 'instruct' suffix, while those from closed-source families are typically referred to by the model name itself).

Value Model After the RL stage, the Value Model can make value judgments on a complete response that are similar to those of the Reward Model. However, it is merely a **scaffolding** used to stabilize the policy model's training process and is therefore typically discarded after training.

3.2 The Forward Pass of PPO

The forward pass of PPO will be detailed in subsections based on key processes and values, following the topological sort of the computation graph.

3.2.1 Policy Model

The policy model is initialized from the SFT model. It takes an input query 'q' from the current dataset and produces an output 'o'. Decomposing the inference process for a single query 'q' into finer steps, the policy model generates logits over the action space (the vocabulary) for each state. From these logits, an action (a single token) is sampled. This continues until an end-of-text token (<|endoftext|>) is generated. Throughout this process, for each state s_t, the logits of the action space produced by the old policy, π_{old}, are stored. (The policy before the backward pass is considered π_{old}; after the update, it becomes the new policy π_θ). These stored logits, denoted as $\pi_{actor}(\cdot|s_t)$, are used for subsequent KL divergence calculations and for extracting the log-probability $\log \pi_{old}(a_t|s_t)$.

3.2.2 Value Model and the Value Estimate *v*

The value model receives the output 'o' from the policy model and provides a value estimate $V_{old}(s_t)$ for each state s_t in the generation process. This value represents the expected future cumulative return that can be obtained by following the current policy from state s_t.

3.2.3 Reference Model

The reference model receives both the input query ‘q‘ and the policy model's output ‘o‘. For each state s_t generated by the policy model at timestep t (i.e., ‘q‘ concatenated with all tokens generated before t), the reference model runs a forward pass in parallel. This produces its own logits distribution over the action space, $\pi_{ref}(\cdot|s_t)$, which is used for the subsequent KL divergence calculation.

3.2.4 KL Divergence

KL divergence measures the difference between the action spaces of π_{policy} and $\pi_{reference}$:

$$KL(\pi_{policy}||\pi_{reference}) = \mathbb{E}_{y\sim\pi_{policy}}[\log(\pi_{policy}(y|x)/\pi_{reference}(y|x))] \tag{6}$$

A separate KL_t is computed for each timestep t. The role of KL divergence here is to constrain the update range of the policy model by measuring its distance from the reference model, ensuring that the policy update remains within a "trust region." This idea originates from Trust Region Policy Optimization (TRPO). By sacrificing potentially large single-step updates, it achieves greater stability over the entire training process, thereby mitigating the problem of reward hacking.

3.2.5 Reward Model and the Reward Signal *r*

The purpose of the reward model is to score the quality of the entire generated sequence. At the end of the forward pass, once the policy model has generated a complete sequence ‘o‘ for a query ‘q‘ (at the final timestep T), the combined ‘q+o‘ is fed into the reward model to obtain a scalar value, R_{RM}. Due to the sparsity of this reward (it is only available at timestep T), the base reward for all other timesteps is zero. However, the subsequent GAE algorithm requires a dense, per-timestep reward signal r_t. Therefore, this signal is constructed as follows: for all intermediate timesteps $t < T$, the reward is the negative KL penalty, $r_t = -\beta \cdot KL_t$. For the final timestep:

$$r_{t=T} = R_{RM} - \beta \cdot KL_t \tag{7}$$

This procedure yields a reward sequence $(r_1, r_2, \ldots, r_T)$ that is used to estimate the advantage with GAE.

3.3 Intermediate Computation Phase (GAE and A)

GAE [Schulman et al., 2015a], which stands for Generalized Advantage Estimation, is a method used to estimate the advantage of the current action. The advantage function $A(s, a)$ measures how much better a specific action a is compared to the average action (the baseline) in a given state s. It not only indicates whether the action is better or worse than average but also quantifies the degree of this difference. The simplest form of the advantage function is:

$$A_t \approx R_t - V(s_t) \tag{8}$$

Here, R_t represents the actual cumulative return the policy obtained in this trajectory starting from timestep t. $V(s_t)$ is the value network's estimate of the value of the current state s_t (before the action is taken), which serves as the baseline. The policy's learning objective is to select actions that lead to performance exceeding this baseline. Therefore, if R_t is the score the agent **actually received** and $V(s_t)$ is its **average expectation** before starting, then the advantage A_t precisely tells the policy: the action a_t you chose this time resulted in a better/worse outcome than expected, and here is by how much. Thissignal is then used to guide the model's updates in a more stable manner.

3.4 The Backward Pass of PPO

In the backward pass of PPO, only the policy model and the value model are updated. Therefore, this section is divided into two parts.

3.4.1 Policy Model

During the backward pass, the policy model is updated based on the objective $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\sum_{t=0}^{T}\gamma^t r_t]$. Guided by the Policy Gradient Theorem, its loss function takes the form $L^{PG}(\theta) = \hat{\mathbb{E}}_t\left[\log\pi_\theta(a_t|s_t)\hat{A}_t\right]$. However, directly optimizing with L^{PG} is notoriously unstable.

To address this, the TRPO algorithm introduced importance sampling and imposed a stricter constraint on the policy update, transforming the loss function into a constrained optimization problem:

$$\begin{aligned} \text{maximize}_\theta \quad & L_{TRPO}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \hat{A}_t \right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t \left[\text{KL} \left[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t) \right] \right] \leq \delta \end{aligned} \tag{9}$$

Here, $\pi_\theta(a_t|s_t)$ is the result of the policy model re-evaluating the state s_t during the backward pass, while $\pi_{old}(a_t|s_t)$ is taken from the forward pass. TRPO adds a complex KL divergence trust region constraint to ensure that each update does not stray too far from the old policy.

PPO simplifies TRPO while inheriting its core idea of "constraining the update step size." It forms a surrogate objective by clipping the importance sampling ratio, $r_t(\theta)$, thus replacing the complex KL constraint in TRPO. The final PPO clipped main objective function is:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\varepsilon, 1+\varepsilon)\hat{A}_t \right) \right] \tag{10}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}$. The 'min' and 'clip' operations ensure that each gradient update stays within a trusted range without performing complex calculations.

3.4.2 Value Model

The training objective of the value model is to make its prediction $V_\phi(s_t)$ approximate the actual return R_t. Theoretically, the loss function is:

$$L_t^{VF}(\phi) = (R_t - V_\phi(s_t))^2 \tag{11}$$

In practice, using the raw return R_t from a single trajectory introduces high variance and leads to unstable training. Therefore, an estimated value target, V_{target}, is used instead. This target is typically constructed from the GAE advantage estimate A_t and the old value estimate $V_{old}(s_t)$: $V_{target} = A_t + V_{old}(s_t)$. Consequently, the loss function is commonly written as:

$$L_t^{VF}(\phi) = (V_{target} - V_\phi(s_t))^2 = (A_t + V_{old}(s_t) - V_\phi(s_t))^2 \tag{12}$$

Here, $V_{old}(s_t)$ is the fixed value estimate from the forward pass, and $V_\phi(s_t)$ is the real-time prediction from the value model during the backward pass.

3.5 Implementation Details

3.5.1 Forward Pass (Including Preparation Phase)

Dataset The dataset used for the PPO forward pass contains not only newly collected data but also a portion of the data from the SFT phase.

Reward Model Although the reward model is initialized from a base model (e.g., the RM in InstructGPT was initialized from a 6B GPT-3), implementation details in RLHF papers mention that it is first fine-tuned on a general QA dataset before being trained with the Bradley-Terry Pairwise Ranking Loss. Therefore, the true initialization of the reward model can be considered a "generalized SFT model."

Value Model The value model is initialized from the trained reward model. This detail, found in the InstructGPT paper, is a highly effective practice. The weights of the RM already encode an understanding of human preferences, providing an excellent starting point for the value model to learn to predict cumulative rewards. While in some other implementations the value model might be initialized from the SFT model, following the original paper's setup provides better insight into its design philosophy.

Reference Model During the forward pass of the reference model, the entire sequence of states up to timestep $t-1$, i.e., $[s_1, s_2, \ldots, s_{T-1}]$ which corresponds to $[q+o_1, q+o_2, \ldots, q+o_{T-1}]$, can be concatenated into a single batch for parallel forward inference. This significantly reduces the computational cost of inference.

Products of the Forward Pass The key values generated during the forward pass (for each timestep t) are the tuple $(s_t, a_t, r_t, V_{old}(s_t), \log \pi_{old}(a_t|s_t))$. These values from the sequence are then used for the intermediate computations and the subsequent backward pass.

3.5.2 Backward Pass

Dataflow Changes During Propagation The dataset $\mathbb{D}$ contains N prompts. During the forward pass (the rollout phase), the policy model generates sequences autoregressively, similar to pre-training, as it is an online learning process. The value model, like the reference model, can perform inference on a batch of size b. This results in a "Rollout Buffer" containing N trajectories (‘q+o‘). This buffer, which consists of token-level data points, is then shuffled and reorganized into multiple minibatches. This process breaks the temporal correlations between consecutive tokens and improves GPU efficiency. A single minibatch may contain m token-level data points from different sequences, but each point retains its original context.

Value Model Training During the backward pass, the value model's parameters ϕ are updated in each step. To ensure the predictions are accurate relative to the current state of the model, the value $V_\phi(s_t)$ is re-computed for the states s_t in the current minibatch just before the backward pass. The training process can be understood with an analogy: a student completes many different exams (rollouts). The teacher then grades a whole stack of them (a minibatch), calculates an average score, and provides unified feedback (one gradient update). When moving to the next minibatch, the value model re-evaluates the new set of problems, providing a new prediction $V_\phi(s_t)$. The value model then learns from the loss $L_t^{VF}(\phi) = (V_{target} - V_\phi(s_t))^2$.

4 The Developmental Trajectory of RLHF and PPO

This chapter analyzes the algorithmic trajectory based on the original research papers for RLHF, GAE, and PPO. It focuses on the algorithms, implementation details, and some experimental conclusions, starting with a high-level overview of RLHF, followed by the core component for advantage estimation, GAE, and concluding with the complete core algorithm, PPO.

4.1 Training Language Models to Follow Instructions with Human Feedback

4.1.1 Overview

This study, published in 2022, is the successor to "Learning to summarize from human feedback" [Stiennon et al., 2020] and provides a comprehensive exploration of the RLHF training pipeline. As shown in Figure 2, the complete RLHF process is divided into three stages: the supervised fine-tuning stage, the reward model training stage, and the reinforcement learning stage. It also introduced PPO-ptx, an optimization of the PPO algorithm. Although RLHF was designed to train models to follow explicit instructions and adhere to implicit intentions such as truthfulness, and avoiding bias, toxicity, or other harmful behaviors, it also established RL as a mainstream approach for enhancing the capabilities of LLMs.

4.1.2 Supervised Fine-Tuning (SFT)

The SFT stage fine-tunes GPT-3 using supervised learning. The paper identified a key paradox: on the validation set for the SFT task itself, the model began to overfit after just one epoch (i.e., the loss stopped decreasing). However, when training was continued for more epochs, these models, which were "overfitting" on the SFT task, actually helps both the RM score and human preference ratings. This indicates a discrepancy between the simple imitation learning objective and true human preferences, providing a strong motivation for the subsequent RLHF phase.

4.1.3 Reward Modeling

The InstructGPT paper mentions several strategies for initializing the Reward Model (RM). The main body of the paper describes initializing from the SFT model, while the appendix notes initializing from a 6B parameter version of GPT-3, which was pre-fine-tuned on public datasets (including a large amount of QA data). Due to issues like training instability and high computational costs, the 175B model was not suitable as an RM. The performance of the 6B model was found to be comparable to initializing from either the base GPT-3 or the SFT model, so the 6B GPT-3 was used. For each prompt, the number of corresponding responses for ranking was between $4 \leq K \leq 9$. Since the learning objective is to favor good responses over bad ones rather than regressing to a specific reward scalar, the loss function was modeled using the sigmoid form of the Bradley-Terry model (see Section 5.1.3 for derivation). The loss function is:

$$L(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathbb{D}}[\log(\sigma(r_\theta(x,y_w) - r_\theta(x,y_l)))] \tag{13}$$

where x is the input prompt, y_w is the chosen response, y_l is the rejected response, and $\mathbb{D}$ is the preference dataset. Finally, the RM's scores were normalized before entering the RL stage.

4.1.4 Reinforcement Learning (RL)

The optimization objective for the RL stage is based on PPO but incorporates an additional pre-training objective. This new algorithm, named PPO-ptx, has an objective function composed of two weighted terms:

$$\text{objective}(\phi) = \underbrace{\mathbb{E}_{(x,y)\sim\mathcal{D}_{\pi_\phi^{RL}}}[r_\theta(x,y) - \beta\log(\pi_\phi^{RL}(y|x)/\pi^{SFT}(y|x))]}_{\text{PPO Reward Objective}} + \underbrace{\lambda\cdot\mathbb{E}_{x\sim\mathcal{D}_{pretrain}}[\log(\pi_\phi^{RL}(x))]}_{\text{Pre-training Gradient Mixin (PTX)}} \quad (14)$$

PPO Reward Objective This is the core PPO part, which maximizes the reward r_θ from the RM while using a KL divergence term as a penalty to prevent the policy π_ϕ^{RL} from deviating too far from the SFT model π^{SFT}. The per-timestep calculation for this objective is as described in Section 3.2.5.

Pre-training Gradient Mixin (PTX) This term calculates the original language model loss of the current policy on the pre-training corpus, where $\mathcal{D}_{pretrain}$ is the pre-training distribution. By re-training the policy model on a portion of the pre-training dataset, it introduces pre-training gradients. This serves to "pull back" some of the model's pre-trained capabilities, mitigating potential performance degradation on general capabilities that can occur during alignment—a phenomenon sometimes called the "alignment tax." Here, λ is a coefficient that controls the mixing strength of the pre-training gradients. When $\lambda = 0$, the model is equivalent to standard PPO. On the 175B model, PPO-ptx was found to outperform PPO.

4.1.5 Other Details

Iteration Steps 2 and 3 of the proposed process can be iterated. One can collect more pairwise data from the current best policy model to train a new RM, which in turn can be used to train an even better policy model.

Dataset The process uses three types of datasets: the **SFT dataset**, containing prompt-demonstration pairs for training the SFT model, with the demonstration as the label, collected from the API and human labelers; the **RM dataset**, containing pairwise preference data for training the RM, with rankings as labels, also collected from the API and labelers; and the **PPO dataset**, which serves as the unlabeled prompt input for the RL stage, collected from the API.

Normalization of the Reward Model Before the RL stage, the output of the RM needs to be normalized to provide a stable and meaningful reward baseline. The theoretical basis for this is that the RM's loss function (Bradley-Terry Loss) depends only on the difference between pairwise rewards, making it invariant to a uniform shift across all reward outputs (translation invariance). Leveraging this property, this study selected the high-quality human demonstrations from the SFT dataset as an anchor point. By adding a bias term b to the raw output of the RM, the model was adjusted so that the average score for this anchor dataset was exactly zero. This effectively calibrates the "average expert human level" to be the zero point of the reward signal.

The Meaning of $r(x, y)$ In the RL stage of RLHF, the score $r(x, y)$ given by the RM is not just an absolute score for the current output. It implicitly contains a comparison with the reference policy π_{ref} (the SFT model). This is because the RM's training data was sampled from π_{ref}, and its output was normalized to have an expected score of zero for generations from π_{ref}. Therefore, the true meaning of the output $r(x, y)$ is how much better or worse the current policy's output y is compared to the average performance of π_{ref}, according to the standard of human preference. It acts like a calibrated ruler, providing a stable and meaningful value signal for PPO's optimization.

Two Important Approximations in RLHF 1. It is assumed that pairwise preferences can be adequately represented by a pointwise reward. 2. It is assumed that a reward model trained on pointwise rewards can generalize from the data distribution it was collected on to out-of-distribution data sampled from the evolving policy. The subsequent DPO algorithm bypasses the second assumption.

4.2 High-Dimensional Continuous Control Using Generalized Advantage Estimation

4.2.1 Overview

This research introduced the Generalized Advantage Estimation (GAE) algorithm, which generalizes and unifies earlier techniques such as TD-error and TD(λ) [Sutton, 1988]. GAE inherits the core idea of TD(λ) by performing a k-step exponentially-weighted summation of Temporal Difference (TD) errors. By adjusting a parameter λ, this framework can elegantly unify and generalize various advantage estimation methods from prior work. Therefore, this section will begin with the problem formulation and the TD-error technique, supplemented with additional interpretations to aid in understanding the formulas, and conclude with practical implementation methods.

4.2.2 The Credit Assignment Problem

In reinforcement learning, the agent's goal is to maximize long-term return. A final success (like winning a game of chess) or failure is the result of a sequence of actions. The challenge of fairly and accurately distributing the "credit" (reward) or "blame" (penalty) of the final outcome to each action in the sequence is known as the credit assignment problem.

The following example illustrates this problem (example inspired by Gemini 2.5 Pro):

Action Sequence You and an AI play a 50-move game of chess. On move 5, you make a subtle pawn move; on move 20, you exchange pieces; on move 40, you deliver a check.

Final Outcome On move 50, you win! You receive a large positive reward of $+1$.

The Credit Assignment Problem How does the algorithm know which move led to the victory? Did the seemingly insignificant pawn move at step 5 lay the foundation for the win? Should it receive most of the credit? Or was the check at step 40 the decisive blow? Alternatively, perhaps your piece exchange at move 20 was a blunder, and you only won because the opponent made a bigger mistake later. Should move 20 be assigned "negative credit" (i.e., blame)? If the algorithm simply distributes the $+1$ reward evenly across all 50 moves, it learns nothing meaningful. It must intelligently determine which actions were "good moves" and which were "bad moves."

4.2.3 Learning Directly from Returns

Policy Update Based on G_t The objective of RL is to maximize cumulative return, not just the immediate reward r_t. Therefore, the most direct policy gradient method (also known as the REINFORCE algorithm) uses the complete Monte Carlo Return, G_t, as the reward signal for action a_t.

$$G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots \tag{15}$$

where γ is the discount factor, representing the importance of future rewards. G_t, via the Monte Carlo method, uses discounted future rewards as the signal for the current action. Substituting G_t into the policy gradient theorem, the gradient becomes:

$$\nabla J(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t) G_t] \tag{16}$$

Using G_t as the reward signal means the learning objective is to "maximize future return."

4.2.4 Introducing a Baseline to Reduce Variance: Learning the Advantage

Policy Update Based on $V(s_t)$ While using G_t solves the problem of short-sightedness, its absolute value can be misleading. In an inherently good state (e.g., a dominant position in chess), even a mediocre move can result in a large positive G_t. Conversely, in a poor state, even a brilliant move might still yield a negative G_t. Furthermore, the full return G_t accumulates randomness from many steps, leading to extremely high variance in the gradient signal. To solve this, a baseline is introduced, typically the state-value function $V(s_t)$. $V(s_t)$ estimates the "average performance" of the policy from the current state s_t, i.e., the expected future return. The reward signal becomes $G_t - V(s_t)$, and the gradient is:

$$\nabla J(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)(G_t - V(s_t))] \tag{17}$$

The term $(G_t - V(s_t))$ is known as the advantage A_t. The learning objective shifts from maximizing the absolute "future return" to maximizing the "excess return beyond expectation." When the policy converges to optimality, A_t approaches zero.

Meaning and Connection of $V(s_t)$ and G_t G_t is the return of an actual action sequence $[a_1, a_2, \dots, a_T]$. Because each a_t is sampled from the action space, G_t contains significant randomness. $V(s_t)$ is the expected value of all future returns, an action-independent value representing how good the current state is: $V(s_t) = \mathbb{E}[G_t|s_t = s]$. It is the expectation of all possible G_t's from state s, which can be thought of as the arithmetic mean of all sampled G_t's given enough samples. In theory, the true value function $V(s_t)$ is an unbiased estimate, but in practice, the learned approximation $V_\phi(s_t)$ is typically a biased but more stable estimate.

4.2.5 Temporal Difference Target and Temporal Difference Error

Using $V(s_t)$ as a baseline addresses the issue of large fluctuations in the reward signal. However, G_t still requires waiting until the end of an episode to be calculated, and its high variance problem remains. Thus, an estimated quantity

$Q(s,a)$ is introduced to replace G_t, changing the advantage function to $A_t = Q(s,a) - V(s_t)$. While G_t is an unbiased estimate of $Q(s,a)$, it has high variance. From $G_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$, we can derive:

$$G_t = r_t + \gamma(r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots) = r_t + \gamma G_{t+1} \tag{18}$$

A one-step decomposition of $V(s_t)$ can be expressed as "the immediate reward at $t+1$ plus the estimated future return from $t+1$ onwards, $V(s_{t+1})$." Since $V(s_{t+1}) = \mathbb{E}[G_{t+1}|s_{t+1} = s]$, we can use $V(s_{t+1})$ as an estimate for G_{t+1} before it is known. The advantage function then becomes:

$$\delta_t = A_t = Q(s,a) - V(s_t) \approx \underbrace{(r_t + \gamma V(s_{t+1}))}_{\text{TD-target}} - V(s_t) \tag{19}$$

δ_t is known as the TD-error, and the first term is the TD-target. The TD-target is a hybrid between G_t and $V(s_t)$, composed of the real one-step reward and an estimated value for the rest of the trajectory. It replaces one step of "pure estimation" with one step of "real observation." The TD-error can also be interpreted as a "surprise" signal, indicating the estimation error of $V(s_t)$ after observing more information at s_{t+1}. As $V(s_t)$ becomes more accurate, δ_t approaches zero. It is worth noting that the original GAE paper refers to this as the "TD residual," but this text uses the more common term "TD-error."

The policy gradient update using the TD-error is:

$$\nabla J(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\delta_t] \tag{20}$$

4.2.6 TD(λ)

In the construction of the TD-target, using the single-step reward r_t and the value estimate $V(s_{t+1})$ to estimate G_t is known as TD(0). This approach effectively addresses the issue of temporal delay and has low variance, as it only depends on the randomness of one step. However, it introduces bias, as $V(s_{t+1})$ is a biased estimate of the true future return.

The Monte Carlo return G_t and the TD-target can be seen as looking ahead $T-t$ steps and 1 step, respectively (where T is the final timestep). G_t is unbiased but has high variance, while the TD-target is biased but has low variance. The problem now transforms into a **bias-variance tradeoff**. The TD(λ) algorithm was proposed to address this by introducing the concept of the n-step return. The n-step return $G_t^{(n)}$ is defined by looking ahead n real reward steps and bootstrapping with the value estimate at the n-th step:

$$\begin{cases} G_t^{(1)} & = r_t + \gamma V(s_{t+1}) \\ G_t^{(2)} & = r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\ & \vdots \\ G_t^{(n)} & = r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n V(s_{t+n}) \end{cases} \tag{21}$$

As n increases, the bias of the n-step return decreases (more real rewards are included), but the variance increases (more steps of randomness are accumulated). The core idea of TD(λ) is to take an exponentially-weighted average of all n-step returns, using a decay factor λ, to obtain a composite λ-return:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \tag{22}$$

When $\lambda = 0$, $G_t^\lambda = G_t^{(1)}$, and TD(λ) degenerates to TD(0). When $\lambda = 1$, $G_t^\lambda = G_t^{(\infty)}$, making it equivalent to the Monte Carlo return G_t. For $0 < \lambda < 1$, TD(λ) provides a flexible balance between bias and variance, enabling the learning of a better $V(s_t)$ through a superior estimate of G_t.

4.2.7 GAE

GAE inherits the bias-variance tradeoff idea from TD(λ). While TD(λ) aims to find a better learning target for $V(s_t)$, GAE aims to find a better advantage estimate $\hat{A}_t$ to provide a better and more stable reward signal for the policy.

Starting from the TD-error, based on the Bellman equation, we have:

$$\begin{aligned} \mathbb{E}_{s_{t+1}}[\delta_t^{V^{\pi,\gamma}}] &= \mathbb{E}_{s_{t+1}}[r_t + \gamma V^{\pi,\gamma}(s_{t+1}) - V^{\pi,\gamma}(s_t)] \\ &= \mathbb{E}_{s_{t+1}}[Q^{\pi,\gamma}(s_t,a_t) - V^{\pi,\gamma}(s_t)] = A^{\pi,\gamma}(s_t,a_t) \end{aligned} \tag{23}$$

This shows that the TD-error δ_t is a one-step estimate of the advantage function A_t. This estimate is unbiased if the value function V is perfectly accurate (i.e., $V = V^\pi$). Similar to TD(0), this estimate is based on a one-step lookahead. Following the balancing idea of TD(λ), the k-step advantage estimate is introduced:

$$\begin{cases} \hat{A}_t^{(1)} := \delta_t^V & = -V(s_t) + r_t + \gamma V(s_{t+1}) \\ \hat{A}_t^{(2)} := \delta_t^V + \gamma\delta_{t+1}^V & = -V(s_t) + r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) \\ \quad \vdots & \\ \hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V & = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{k-1} r_{t+k-1} + \gamma^k V(s_{t+k}) \end{cases} \tag{24}$$

As with TD(λ), as $k \to 1$, variance decreases, and as $k \to \infty$, bias decreases, returning to the classic bias-variance tradeoff. Similarly, GAE takes a weighted sum of these k-step estimates using a decay factor λ:

$$\begin{aligned} \hat{A}_t^{\text{GAE}(\gamma,\lambda)} &:= (1-\lambda)\left(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + \dots\right) \\ &= (1-\lambda)\left(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + \dots\right) \\ &= (1-\lambda)\left(\delta_t^V(1 + \lambda + \lambda^2 + \dots) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \lambda^3 + \dots)\right. \\ &\quad \left. + \gamma^2\delta_{t+2}^V(\lambda^2 + \lambda^3 + \lambda^4 + \dots) + \dots\right) \\ &= (1-\lambda)\left(\delta_t^V\left(\frac{1}{1-\lambda}\right) + \gamma\delta_{t+1}^V\left(\frac{\lambda}{1-\lambda}\right) + \gamma^2\delta_{t+2}^V\left(\frac{\lambda^2}{1-\lambda}\right) + \dots\right) \\ &= \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V \end{aligned} \tag{25}$$

Here, $(1-\lambda)$ acts as a normalization constant in the full expansion. The cases where $\lambda = 0$ and $\lambda = 1$ are two special instances:

$$\begin{cases} \text{GAE}(\gamma, 0): & \hat{A}_t := \delta_t^V & = -V(s_t) + r_t + \gamma V(s_{t+1}) \\ \text{GAE}(\gamma, 1): & \hat{A}_t := \sum_{l=0}^{\infty}\gamma^l\delta_{t+l}^V = -V(s_t) + \sum_{l=0}^{\infty}\gamma^l r_{t+l} \end{cases} \tag{26}$$

When $\lambda = 0$, GAE$(\gamma, 0)$ is the classic TD-error, exhibiting low variance and high bias. When $\lambda = 1$, GAE$(\gamma, 1)$ reverts to the simplest form $G_t - V(s_t)$, exhibiting zero bias and high variance. For $0 < \lambda < 1$, the estimator strikes a balance between bias and variance.

The policy gradient update using GAE is:

$$\nabla J(\theta) = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t)\hat{A}_t^{\text{GAE}(\gamma,\lambda)}] = \hat{\mathbb{E}}_t\left[\nabla_\theta \log \pi_\theta(a_t|s_t)\sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V\right] \tag{27}$$

4.2.8 The Backward View of GAE

The final expression in Equation (25), $\hat{A}_t^{\text{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V$, represents a forward view (i.e., computing from $t = 0$ onwards). However, in practice, $\hat{A}_t$ is calculated using a backward pass. Expanding $\hat{A}_t^{\text{GAE}(\gamma,\lambda)}$:

$$\begin{aligned} \hat{A}_t^{\text{GAE}(\gamma,\lambda)} &= \sum_{l=0}^{\infty}(\gamma\lambda)^l\delta_{t+l}^V \\ &= \delta_t + (\gamma\lambda)[\delta_{t+1} + (\gamma\lambda)\delta_{t+2} + (\gamma\lambda)^2\delta_{t+3} + \dots] \\ &= \delta_t + (\gamma\lambda)\hat{A}_{t+1}^{\text{GAE}(\gamma,\lambda)} \end{aligned} \tag{28}$$

Equation (28) gives the practical computation formula. First, all TD-errors δ_t are computed in a forward pass. Then, starting from the end of the trajectory ($A_T = 0$ if the episode terminates; in PPO, a truncated sequence also has a terminal value), one calculates $\hat{A}_t^{\text{GAE}(\gamma,\lambda)}$ step-by-step backwards, avoiding the complex infinite sum at each timestep.

4.2.9 Conceptual Aids

MC vs. TD In practice, tasks can be episodic (finite) or continuing (infinite), distinguished by the presence or absence of a terminal state. Since the Monte Carlo method requires a final outcome to compute returns, it is only applicable to episodic tasks. For continuing tasks, estimation is necessary to obtain a reward signal, making TD methods the standard and only feasible approach.

TD(λ) vs. GAE The forms of TD(λ) and GAE are very similar, but their objects and objectives differ. TD(λ) operates on the return G_t with the goal of learning a better value function $V(s_t)$. GAE operates on the advantage estimate $\hat{A}_t$ with the goal of providing a better and more stable reward signal for the policy. Although a better $V(s_t)$ is also learned during the GAE process, it is a byproduct.

λ vs. γ In the final GAE expression, both λ and γ act on δ_t. Although they are multiplied together, their meanings are distinct. Returning to Equation (25), γ (the discount factor) defines the very structure of the return G_t. It is embedded within each δ_t and k-step advantage $\hat{A}^{(k)}$, determining how the value of future rewards decays over time. It is part of the problem definition. In contrast, λ (the trace-decay parameter) defines how to combine advantage estimates over different time scales. It acts as a mixing weight between different k-step advantages, determining whether the final estimate $\hat{A}_{\text{GAE}}$ trusts short-term TD-errors more or long-term Monte Carlo returns more. Therefore, they can be distinguished as follows: γ is like an **objective law** (importance of the future), defining the value concept itself, akin to a world rule that is the same for all policies. λ is more like a **subjective choice** (confidence in future estimates), a component of the solution algorithm, determining whether the learning strategy is "conservative" or "aggressive," independent of the task itself.

4.3 Proximal Policy Optimization Algorithms

4.3.1 Overview

This research introduced the PPO algorithm, which inherits the idea of constraining gradient updates from TRPO [Schulman et al., 2015b]. By using ‘min‘ and ‘clip‘ operations, PPO simplifies TRPO, ultimately becoming a cornerstone for modern LLM training and RL research. This section will start from the optimization objective of TRPO and introduce two improvement pathways in PPO, culminating in the unified objective form of the PPO algorithm framework.

4.3.2 TRPO: Trust Region Policy Optimization

With an advantage estimator, the optimization objective evolved from maximizing an absolute return to maximizing an excess return. The pseudo-loss takes the form $L^{PG}(\theta) = \hat{\mathbb{E}}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t]$. TRPO builds upon this by introducing importance sampling and adding a KL divergence constraint to the policy update, ensuring each update remains within a trust region. For reader convenience, we restate the TRPO objective:

$$\begin{aligned} \text{maximize}_\theta \quad & L_{TRPO}(\theta) = \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}\hat{A}_t\right] \\ \text{subject to} \quad & \hat{\mathbb{E}}_t\left[\text{KL}\left[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \leq \delta \end{aligned} \tag{9}$$

Here, $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the importance sampling ratio, which scales the reward signal $\hat{A}_t$, effectively strengthening or weakening the current update's gradient. Its core function is to correct the bias that arises from using data from the old policy π_{old} to evaluate the new policy π_θ by re-weighting the advantage $\hat{A}_t$. The constraint ensures that the KL divergence between π_{old} and π_θ is less than a value δ, which controls whether the policy's learning strategy is "conservative" or "aggressive."

4.3.3 PPO Research Motivation

Although TRPO further stabilized policy training, its constrained optimization approach increased the complexity of the objective. While one could turn the hard KL constraint into a soft penalty:

$$\text{maximize}_\theta \, \hat{\mathbb{E}}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)}\hat{A}_t - \beta\text{KL}\left[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \tag{29}$$

in Equation (29), choosing a fixed β that performs well under all conditions is difficult, if not impossible. As the policy converges, the KL penalty term can become relatively too large. Furthermore, the original TRPO objective only requires the KL divergence to be within a certain bound, so simply adding it as a penalty term is not an ideal solution.

4.3.4 PPO-Clipped Surrogate Objective

PPO first isolates the optimization objective, terming it Conservative Policy Iteration (CPI), and defines it as $L^{CPI}(\theta)$:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \hat{A}_t \right] = \hat{\mathbb{E}}_t \left[r_t(\theta) \hat{A}_t \right] \tag{30}$$

Without the KL constraint, when $r_t(\theta)$ strays far from 1, $L^{CPI}(\theta)$ will produce large gradients, leading to excessive policy updates. Therefore, 'min' and 'clip' operations are applied to $r_t(\theta)$, transforming the objective into:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t) \right] \tag{10}$$

The 'clip' function constrains $r_t(\theta)$ to the range $[1-\epsilon, 1+\epsilon]$. By then taking the minimum, the gradient range of $L^{CLIP}(\theta)$ is controlled, ensuring the policy update remains within a trust region and preventing training collapse from overly large updates.

4.3.5 PPO with Adaptive KL Penalty

Besides clipping the importance sampling ratio, PPO also explored another approach: an adaptive KL penalty coefficient β. Its objective function is similar to Equation (29):

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\text{old}}(a_t|s_t)} \hat{A}_t - \beta \text{KL}\left[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right] \right] \tag{31}$$

Here, β is adjusted based on the actual KL divergence observed during training:

$$\begin{aligned} &\text{Compute } d = \hat{\mathbb{E}}_t \left[\text{KL}\left[\pi_{\text{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)\right]\right] \\ &\text{If } d < d_{targ}/1.5,\ \beta \leftarrow \beta/2 \\ &\text{If } d > d_{targ} \times 1.5,\ \beta \leftarrow \beta \times 2 \end{aligned} \tag{32}$$

This is a straightforward idea: since a fixed β is unsuitable as the policy changes, adjust it based on the actual situation. A target KL divergence, d_{targ}, is set at the beginning, and β is adapted based on the observed KL divergence.

4.3.6 The Unified Objective of the PPO Framework

PPO is built on an actor-critic framework. In practice, it is common for the actor and critic to share network parameters. Although in LLM RLHF training, the actor, critic, and judge are often separate models, it is still valuable to analyze the parameter-sharing paradigm. A unified objective for the PPO framework is as follows:

$$L_t^{\text{CLIP+VF+S}}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{33}$$

The overall objective consists of three parts: the policy objective $L_t^{\text{CLIP}}(\theta)$, the value model objective $L_t^{\text{VF}}(\theta)$, and the policy's action space entropy $S[\pi_\theta](s_t)$.

Policy Objective This is the term from Equation (10), which aims to learn a better policy within a trust region, guided by the advantage function.

Value Model Objective This corresponds to the content in Section 3.4.2, aiming to better predict the expected future return and approximate the true value function V^π.

Policy Entropy Bonus This is an additional term. The entropy bonus encourages the probability distribution of the action space to be smoother, preventing the policy from becoming overly deterministic by drastically reducing/increasing the probability of an action after a large loss/reward. This promotes exploration and stabilizes training.

Backward Pass with Shared Parameters When network parameters are shared, typically the initial layers are shared (Shared Network), followed by separate heads for the actor and critic (Actor Head and Critic Head). The hyperparameters c_1 and c_2 in Equation (33) serve to weight these different objectives. During the backward pass, the gradients from $L_{policy} + L_{entropy}$ flow to the actor head, while the gradients from L_{value} flow to the critic head. These gradients are then combined and flow back through the Shared Network, and the optimizer performs a unified update based on the total gradient.

4.3.7 Other Details

The PPO paper proposed two main variants: clipping and adaptive KL. Experiments compared four scenarios for $L^{CPI}(\theta)$: no clipping or penalty (Eq. (30)), clipping (Eq. (10)), adaptive KL (Eq. (31)), and a fixed KL penalty (Eq. (29)). Across multiple hyperparameter settings, the performance ranking was: clipping > adaptive KL > fixed KL > no clipping or penalty. As the clipping approach is simpler, more elegant, and easier to implement than the adaptive KL penalty, it is the most commonly used variant.

4.3.8 PPO Algorithm Implementation

With the GAE advantage estimator, the actor-critic framework, and the clipping function, the implementation can be described by the following pseudocode (from the PPO paper):

Algorithm 1 PPO, Actor-Critic Style

for iteration=1, 2, ... **do**
  for actor=1, 2, ..., N **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

4.3.9 The Details and Meaning of 'min' and 'clipping'

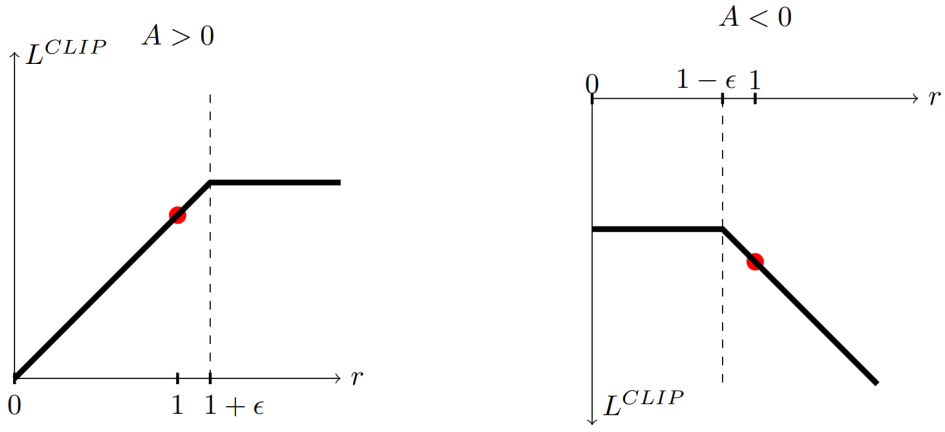

Figure 3: This figure, from the PPO paper, shows how L_{CLIP} changes based on the sign of the advantage after the 'min' and 'clipping' operations are applied.

The clipping function directly trims the importance sampling ratio based on a given trust interval $[1-\epsilon, 1+\epsilon]$, thereby limiting the policy's gradient update range. The combination of 'min' and 'clip' is the essence of PPO, aiming to balance exploration and stability. The 'min' operation acts as a selector, always choosing the **more pessimistic** value between the original and clipped objectives. This asymmetric clipping design embodies a sophisticated "reward-penalty philosophy."

The overall goal is to adjust the policy π_θ to maximize the objective function L. When the advantage A_t is positive, the update is a reward; when A_t is negative, it's a penalty. $r > 1$ means the current action is more probable under π_θ than π_{old}; $r < 1$ means it is less probable.

When $A_t > 0$ (Encouraging Good Actions) The goal is to maximize L by increasing r_t. L^{CLIP}, via the 'min' operation, sets an upper bound of $(1+\epsilon)A_t$ on the "gain" from a single update. This is a conservative reward strategy, preventing the model from over-updating due to a single, unusually high advantage, which could sacrifice overall

stability. For the case where $r_t < 1$ (i.e., the policy is "forgetting" a good action), the objective $r_t \times A_t$ naturally decreases, so no hard lower bound is needed.

When $A_t < 0$ (Penalizing Bad Actions) The goal is to maximize L by decreasing r_t (i.e., making a negative number closer to zero). L^{CLIP} sets a lower bound on the magnitude of the penalty, preventing excessive correction, while allowing for an unlimited penalty in magnitude. For the case where $r_t < 1$ (i.e., the policy is correctly "learning from its mistake"), the penalty's magnitude is capped, preventing the model from over-penalizing an action based on one bad sample. If r_t becomes very large (i.e., the policy drastically increases the probability of a bad action), the 'min' operation will select the more negative $r_t \times A_t$ term, applying a large penalty gradient to forcefully pull the policy back on track. This reflects a strict correctional mechanism, thus no hard upper bound on the penalty magnitude is needed.

The Gradient of the Reward Signal: The True Learning Signal Through the policy gradient theorem, the model learns by trial and error, responding to the **gradient of the reward signal**, not simply maximizing its absolute value. The gradient indicates how the model should treat the current action. When the signal is positive ($r \times A > 0$), the model increases the probability of the sampled action; when negative ($r \times A < 0$), it decreases it. The clipping function prevents the model from excessively increasing/decreasing an action's probability. When the probability is already high/low enough, the reward/penalty is muted, indicating the update is sufficient.

This asymmetric clipping ensures that for both cases, the objective is to increase $L_t(\theta)$. The unclipped side tells the actor that the current policy's action is either particularly bad or not good enough, and the policy can be adjusted to decrease/increase the probability of this action. In summary, PPO's asymmetric clipping philosophy can be described as: be cautious with "good news" to prevent rashness; be alert to "bad news" to allow for strict correction. This design makes PPO extremely robust and stable in complex optimization landscapes.

4.4 Summary: The RLHF Jigsaw Puzzle

Starting from the highest-level RLHF framework, the core idea is to use RL to align with human preferences. To provide a dense training signal for PPO, RLHF cleverly constructs a per-step immediate reward, r_t, composed of two parts: a sparse final score from the Reward Model and a dense KL divergence penalty from comparison with the SFT model.

GAE, as a core component of RL, provides PPO with an advantage estimate $\hat{A}_t$ that flexibly trades off bias and variance. By taking an exponentially weighted sum of TD-errors, it strikes a delicate balance between the high variance of Monte Carlo methods and the bias of single-step TD.

PPO, as the executor of RLHF and a successor to TRPO, uses its signature Clipped Surrogate Objective to constrain policy updates within a trust region in a simpler and more efficient manner, effectively preventing policy collapse.

RLHF is the complete practice of PPO for LLMs; PPO is an improvement over TRPO; GAE is an improvement over simpler advantage functions. The practical process of PPO within RLHF is as follows: the actor (policy model) and reference model perform forward inference to generate tokens. The critic (value model) provides a value V_t for each token. After a full sequence is generated, the judge (reward model) gives a raw score R_{RM} for the complete sequence. Then, the algorithm combines R_{RM} and the per-step KL divergence to construct the immediate reward r_t for each step. Next, it computes δ_t for each timestep in a forward pass, and then computes A_t for each timestep in a backward pass. Using A_t and $V_{old}(s_t)$, the target value V_{target} (R_t) is calculated. Finally, the policy model updates based on A_t, and the value model updates based on V_{target}.

5 DPO & IPO

This chapter will analyze the derivation of the DPO algorithm and introduce several interesting variants of DPO.

5.1 Direct Preference Optimization: Your Language Model is Secretly a Reward Model

5.1.1 Overview

Through a rigorous mathematical derivation, Direct Preference Optimization (DPO) reveals that under the classic Bradley-Terry model, the reward maximization objective of PPO can be equivalently transformed into a classification-style loss function that is trained directly on preference data. This transformation means that one can completely bypass the explicit training of a reward model and the subsequent reinforcement learning sampling loop. Instead, a language model can be optimized to align with human preferences directly through a simple, SFT-like loss. DPO not only significantly simplifies the training pipeline but also provides a theoretical foundation for a subsequent series of interesting alignment algorithms.

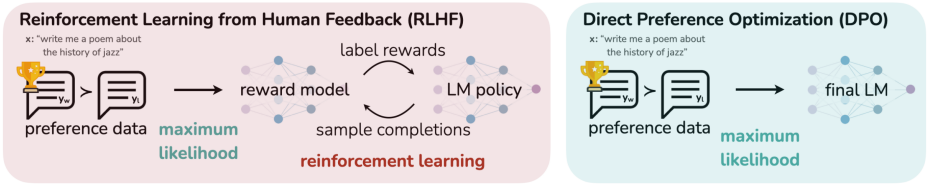

Figure 4: This figure, from the DPO research paper, illustrates the difference between the DPO and RLHF algorithms.

5.1.2 Motivation

In the methodology for aligning LLMs with human preferences, RLHF is undoubtedly the most successful paradigm. If SFT merely teaches an LLM to "learn to talk" through imitation learning, then RLHF enables the LLM to further understand the value judgments behind behaviors, thereby generating content more aligned with human preferences. Despite its power, the RLHF process is far more complex than supervised learning and relies on a complicated online RL sampling loop. This makes the entire process difficult to maintain and expensive. These issues are all related to the reward model. Therefore, the focus of the DPO algorithm is to bypass explicit reward modeling while retaining the core idea of RLHF.

5.1.3 From Bradley-Terry Model to Reward Model Loss

The Bradley-Terry model is a probabilistic model used to infer the outcome of pairwise comparisons between items, teams, or objects. It is defined as the probability that item i is preferred to item j, given a pair drawn from some population:

$$\mathrm{P}(i \succ j) = \frac{p_i}{p_i + p_j} \tag{34}$$

where p_i is a positive real-valued score assigned to item i, and $i \succ j$ can be interpreted as "i is preferred to j," depending on the context. For example, if p_i represents the skill of a sports team, $\mathrm{P}(i \succ j)$ is the probability that team i defeats team j.

A common functional form for p_i is $p_i = e^i$. This parameterization ensures that the score p_i is always positive and allows the probability ratio to be transformed into a sigmoid of a difference, whose logarithmic derivative has favorable properties, fitting perfectly with logistic regression and the reward model loss function:

$$\begin{aligned} \mathrm{P}(i \succ j) &= \frac{e^i}{e^i + e^j} \\ &= \frac{1}{1 + e^{-(i-j)}} = \sigma(i - j) \end{aligned} \tag{35}$$

with the derivative property:

$$\frac{\partial \log \sigma(t)}{\partial t} = \sigma(-t) \tag{36}$$

In reward modeling, the goal is to learn to compare preferences. Since the absolute value of a reward is difficult to quantify, data is organized through preference rankings. The Bradley-Terry model is well-suited for this scenario. Using its sigmoid form, the probability of a preferred response y_1 over a dispreferred one y_2 given a prompt x is:

$$p^*(y_1 \succ y_2 | x) = \frac{\exp(r^*(x, y_1))}{\exp(r^*(x, y_1)) + \exp(r^*(x, y_2))} = \sigma(r^*(x, y_w) - r^*(x, y_l)) \tag{37}$$

where $r^*(x, y_i)$ represents the ground-truth reward for input x and its corresponding output y_i. Through maximum likelihood estimation, we can derive the negative log-likelihood loss for the reward model:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log \sigma \left(r_\phi(x, y_w) - r_\phi(x, y_l) \right) \right] \tag{38}$$

where r_ϕ is the reward model to be trained, and $\mathcal{D} = \{x^{(i)}, y_w^{(i)}, y_l^{(i)}\}$ is the static preference dataset.

5.1.4 From RLHF to DPO

In RLHF, the PPO reward objective is:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}[r(x,y) - \beta\mathbb{D}_{KL}(\pi_\phi(y|x)||\pi_{ref}(y|x))] \tag{39}$$

where $r(x,y)$ is given by the reward model.

The core idea of DPO is to bypass explicit reward modeling. However, the current reward objective (Equation (39)) is tightly coupled with the reward model $r(x,y)$. To decouple them, we need to find an equivalent new objective that does not contain $r(x,y)$. The derivation is as follows:

$$\begin{aligned}
&\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}[r(x,y)] - \beta\mathbb{D}_{\text{KL}}[\pi_\phi(y|x) \,\|\, \pi_{\text{ref}}(y|x)] \\
&= \max_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi_\phi(y|x)} \left[r(x,y) - \beta \log \frac{\pi_\phi(y|x)}{\pi_{\text{ref}}(y|x)} \right] \\
&= \min_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi\phi(y|x)} \left[\log \frac{\pi_\phi(y|x)}{\pi_{\text{ref}}(y|x)} - \frac{1}{\beta} r(x,y) \right] \\
&= \min_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi_\phi(y|x)} \left[\log \frac{\pi_\phi(y|x)}{\pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x,y)\right)} \right] \\
&= \min_{\pi} \mathbb{E}_{x\sim\mathcal{D}}\mathbb{E}_{y\sim\pi_\phi(y|x)} \left[\log \frac{\pi_\phi(y|x)}{\frac{1}{Z(x)}\pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x,y)\right)} - \log Z(x) \right]
\end{aligned} \tag{40}$$

where $Z(x)$ is the partition function:

$$Z(x) = \sum_y \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x,y)\right) \tag{41}$$

In the derivation above, we introduced the partition function $Z(x)$ to normalize the denominator into a valid probability distribution. Since $Z(x)$ only depends on x and π_{ref}, and not on the policy π_θ we are optimizing, we can define the optimal policy as:

$$\pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x,y)\right) \tag{42}$$

This is a valid probability distribution because $\pi^*(y|x) \geq 0$ for all y and $\sum_y \pi^*(y|x) = 1$. Substituting $\pi^*(y|x)$ into Equation (40) and simplifying yields:

$$\begin{aligned}
&\min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}} \left[\sum_y \pi_\theta(y|x) \log \frac{\pi_\theta(y|x)}{\pi^*(y|x)} - \log Z(x) \right] \\
&= \min_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}} \left[\mathbb{D}_{KL}(\pi_\theta(y|x)||\pi^*(y|x)) - \log Z(x) \right]
\end{aligned} \tag{43}$$

The reward optimization objective from Equation (39) has now been transformed into the form above. Since $Z(x)$ does not depend on π_θ, the minimum is achieved by the first KL term. The KL divergence is minimized at 0, which occurs when $\pi_\theta(y|x) = \pi^*(y|x)$. Thus, the optimal policy is:

$$\pi_\theta(y|x) = \pi^*(y|x) = \frac{1}{Z(x)} \pi_{\text{ref}}(y|x) \exp\left(\frac{1}{\beta} r(x,y)\right) \tag{44}$$

The final goal of DPO is to bypass explicit reward modeling. From Equation (44), we can express the reward $r(x,y)$ in terms of the optimal policy π_θ:

$$r(x,y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x) \tag{45}$$

Substituting this expression for the reward into the reward model loss (Equation (38)) yields the DPO loss function:

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right] \tag{46}$$

Ultimately, the DPO loss function depends only on the policy model π_θ and the reference model π_{ref}, bypassing explicit modeling of the reward model r_ϕ. It transforms the complex RL problem into a loss function L_{DPO} that can be optimized directly to train the policy model. Although an explicit reward signal in the traditional sense cannot be directly computed, an optimal policy model π_θ itself has implicitly encoded the reward information. The reward function can be expressed as the log-ratio of the policy and reference model probabilities: $r(x, y) \propto \log \frac{\pi_\theta(y_w|x)}{\pi_{\text{ref}}(y_w|x)}$. This is precisely the meaning behind the title, "Your Language Model is Secretly a Reward Model."

5.1.5 DPO Algorithm Analysis

Intuitive Analysis of the DPO Mechanism The DPO loss function implicitly contains a penalty for the policy π_θ deviating from the reference policy π_{ref}. This penalty is not an independently introduced KL term but is embodied in its core implicit reward function, $\log(\pi_\theta/\pi_{ref})$, which also constrains the policy updates. Thus, DPO inherits the core idea of "constraining policy updates within a trust region" from PPO. Analyzing the gradient of the DPO loss provides further insight:

$$\begin{aligned}\nabla_\theta \mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\beta \mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \Big[& \sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w)) \\ & \cdot (\nabla_\theta \log \pi_\theta(y_w|x) - \nabla_\theta \log \pi_\theta(y_l|x)) \Big]\end{aligned} \tag{47}$$

Here, the weighting term $\sigma(\hat{r}_\theta(x, y_l) - \hat{r}_\theta(x, y_w))$ intuitively reflects the severity of the model's error. If the model can already distinguish y_w and y_l well (i.e., $r_w \gg r_l$), this weight will be very small (close to 0), leading to a gentle gradient update. If the model gets the preference wrong ($r_l > r_w$), the weight will be large (close to 1), resulting in a strong corrective gradient. The term $\nabla_\theta \log \pi_\theta(y_w|x)$ pushes the model parameters towards the maximum likelihood of y_w, while $-\nabla_\theta \log \pi_\theta(y_l|x)$ pushes them away from the likelihood of y_l. This effectively increases the probability of generating y_w and decreases the probability of generating y_l, with the final gradient scaled by β.

Superiority of DPO Compared to Actor-Critic Standard actor-critic algorithms in RLHF can suffer from training instability. We can re-analyze this within the DPO framework. If we were to introduce a term unrelated to the current policy π_θ, such as the DPO partition function $Z(x)$, into the PPO reward objective (Equation (39)), the objective would become:

$$\max_{\pi_\theta} \mathbb{E}_{\pi_\theta(y|x)} \Bigg[\underbrace{r_\phi(x, y) - \beta \log Z(x)}_{f(r_\phi, \pi_{\text{ref}}, \beta)} \underbrace{-\beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)}}_{\text{KL}} \Bigg] \tag{48}$$

Under this objective, the critic (value model) would need to estimate the value of the newly introduced term containing $Z(x)$. Since $Z(x)$ cannot be directly computed, estimating its value with a critic is extremely difficult and introduces bias. Furthermore, in the PPO algorithm, the reward model's output needs to be manually normalized before use. This normalization is essentially a Monte Carlo estimation to establish a baseline for filtering out signals like $Z(x)$. In contrast, DPO's reward re-parameterization uses the reference model π_{ref} as an implicit, non-learnable baseline, eliminating the complex, policy-dependent partition function $Z(x)$ from the optimization objective. This avoids the bias and instability associated with training a Value Model that needs to estimate a complex expectation.

5.2 A General Theoretical Paradigm for Understanding Learning from Human Preferences

5.2.1 Overview

This research [Azar et al., 2024] develops a general preference optimization framework named ΨPO, which theoretically unifies RLHF and DPO. Through this unified lens, it clearly reveals the weak regularization and overfitting problems that the DPO algorithm may suffer from due to the nature of its loss function. Ultimately, while inheriting DPO's core idea of bypassing an explicit reward model, the study proposes the Identity Preference Optimization (IPO) algorithm, derived through a new path within the ΨPO framework, to solve this problem.

Note: In the original paper, τ corresponds to the weight β of the KL divergence penalty term in RLHF and DPO. In this section, we will follow the original paper's notation and use τ.

5.2.2 Two Important Conceptual Distinctions

To analyze preference learning from a more general mathematical perspective, the paper defines expected preferences at both the action and policy levels. This forms the foundation for constructing the general objective function ΨPO:

Given a context x, the expected preference of a completion y over a distribution μ is denoted as $p^*(y \succ \mu|x)$ and expressed as:

$$p^*(y \succ \mu|x) = \mathbb{E}_{y' \sim \mu(.|x)}[p^*(y \succ y'|x)].$$

For any two policies $\pi, \mu \in \Delta_{\mathcal{Y}}^{\mathcal{X}}$ and a context distribution ρ, the total preference of policy π over μ is denoted as $p^*_\rho(\pi \succ \mu|x)$ and expressed as:

$$p^*_\rho(\pi \succ \mu|x) = \mathbb{E}_{\substack{x \sim \rho \\ y \sim \pi(.|x)}}[p^*(y \succ \mu|x)].$$

5.2.3 Review of Key Objectives in RLHF and DPO

The core optimization objective in both RLHF and DPO is based on the Bradley-Terry model of preferences:

$$p(y \succ y'|x) = \sigma(r(x, y) - r(x, y')) \tag{49}$$

The corresponding loss function for the reward model is:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[\log(p(y_w \succ y_l|x))\right] \tag{50}$$

Next is the RLHF optimization objective, which can be simplified as:

$$J(\pi) = \mathbb{E}_\pi[r(x, y)] - \tau D_{KL}(\pi||\pi_{ref}) \tag{51}$$

And the DPO optimization objective is:

$$\min_\pi \mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}} \left[-\log \sigma \left(\tau \log \left(\frac{\pi(y_w|x)}{\pi_{\text{ref}}(y_w|x)} \right) - \tau \log \left(\frac{\pi(y_l|x)}{\pi_{\text{ref}}(y_l|x)} \right) \right) \right] \tag{52}$$

The DPO objective above is expressed as an expectation over the empirical dataset $\mathcal{D}$. To theoretically compare it fairly with the RLHF objective (which is an expectation over the dynamic sampling distribution of policy π), the DPO loss must also be generalized to its population form. In this form, for an infinite domain, $x \sim \rho$ represents any prompt, and $y, y' \sim \mu$ represent responses sampled from an LLM for that prompt. The loss in its population form is:

$$\min_\pi \mathbb{E}_{\substack{x \sim \rho \\ y, y' \sim \mu}} \left[-p^*(y \succ y'|x) \log \sigma \left(\tau \log \left(\frac{\pi(y|x)}{\pi(y'|x)} \right) - \tau \log \left(\frac{\pi_{\text{ref}}(y|x)}{\pi_{\text{ref}}(y'|x)} \right) \right) \right]. \tag{53}$$

The DPO paper has already proven that when the Bradley-Terry model (Eq. (49)) perfectly fits the preference data and the optimal reward function $r(x, y)$ is obtained from Eq. (50), the global optimizers of the RLHF objective (Eq. (51)) and the DPO objective (Eq. (53)) are identical.

5.2.4 The General Preference Optimization Objective: ΨPO

Inspired by the RLHF objective (Eq. (51)), a general objective for preference optimization is proposed:

$$\max_\pi \mathbb{E}_{\substack{x \sim \rho \\ y \sim \pi(.|x) \\ y' \sim \mu(.|x)}} [\Psi(p^*(y \succ y'|x))] - \tau D_{\text{KL}}(\pi \,\|\, \pi_{\text{ref}}). \tag{54}$$

Here, $\Psi : [0, 1] \to \mathbb{R}$ is a general non-decreasing function, τ is the regularization parameter, and $y \sim \pi(\cdot|x)$ and $y' \sim \mu(\cdot|x)$ represent the probability distributions of the policy being optimized and a comparison policy, respectively, for a given x. For instance, in RLHF, μ would be π_{ref}. The next step is to prove that this new objective is equivalent to the RLHF objective.

Since this objective (Eq. (54)) only differs from the RLHF objective (Eq. (51)) in the first term, we will simplify the Ψ term. The reward $r(y)$ is embedded within $p^*(y \succ y')$ (see Eq. (49)). Therefore, by choosing the log-odds function (the inverse of sigmoid), $\Psi(q) = \log(q/(1-q))$, we can extract $r(y)$ from the first term of the objective. Since the RLHF objective only depends on y, we expand this term over the $y' \sim \mu$ dimension and simplify:

$$\begin{aligned}
\mathbb{E}_{y' \sim \mu}[\Psi(p^*(y \succ y'|x))] &= \mathbb{E}_{y' \sim \mu} \left[\Psi \left(\frac{e^{r(y)}}{e^{r(y)} + e^{r(y')}} \right) \right] \\
&= \mathbb{E}_{y' \sim \mu}[\log(e^{r(y)}/e^{r(y')})] \\
&= \mathbb{E}_{y' \sim \mu}[r(y) - r(y')] \\
&= r(y) - \mathbb{E}_{y' \sim \mu}[r(y')].
\end{aligned}$$

When the simplified term is substituted back into the ΨPO objective, the second part, $\mathbb{E}_{y' \sim \mu}[r(y')]$, becomes a constant with respect to the optimization over π and can be ignored. Thus, it is proven that the ΨPO objective is equivalent to

the RLHF objective, and by extension, to the DPO objective. The design of ΨPO indicates that a general optimization objective should consist of a non-decreasing reward term and a KL divergence regularization term. Both RLHF and DPO satisfy this form, although DPO does so implicitly.

Having proven the equivalence of the ΨPO, RLHF, and DPO objectives, their closed-form solutions for the optimal policy are also identical (for detailed derivation, see Section 5.1.4 and the derivation of Eq. (42)):

$$\pi^*(y|x) \propto \pi_{\text{ref}}(y|x) \exp\left(\tau^{-1}\mathbb{E}_{y'\sim\mu(\cdot|x)}[\Psi(p^*(y \succ y'|x))]\right). \tag{55}$$

5.2.5 Analyzing the Weak Regularization and Overfitting of PPO and DPO via ΨPO

Returning to the closed-form solution for the optimal policy (Eq. (55)), consider the case where an action y is always preferred to y', i.e., $p(y \succ y') = 1$. In the Bradley-Terry model, this corresponds to $(r(y) - r(y')) \to +\infty$. Substituting this into Equation (55) to find $\pi^*(y)$ and $\pi^*(y')$ respectively:

$$\begin{aligned}
\pi^*(y) &\propto \pi_{\text{ref}}(y) \exp\left(\tau^{-1}(r(y) - r(y'))\right) \\
&\quad \text{where } \exp\left(\tau^{-1}(r(y) - r(y'))\right) \to +\infty \\
\pi^*(y') &\propto \pi_{\text{ref}}(y') \exp\left(\tau^{-1}(r(y') - r(y))\right) \\
&\quad \text{where } \exp\left(\tau^{-1}(r(y') - r(y))\right) \to 0
\end{aligned}$$

In this scenario, regardless of the value of τ, we will always get $\frac{\pi^*(y')}{\pi^*(y)} = 0$. This means that as preferences become more deterministic, the ability of τ and the τ-weighted KL regularization term to control the policy becomes weaker.

From this analysis, we can draw the following conclusions:

- The DPO algorithm's objective is centered on the Bradley-Terry loss, which aims to learn $p(y_w \succ y_l) = 1$. This requires infinitely increasing the "distance" between y_w and y_l in the policy's log-probability space, i.e., $(r(y_w) - r(y_l)) \to +\infty$. The loss approaches, but never reaches, zero, meaning there is always room for further reduction. This eventually leads to overfitting, causing policy degradation, where the LLM loses diversity and generalization capabilities.
- In the RLHF process, the Bradley-Terry loss is used to train the reward model. Due to factors like the model's own regularization and its finite capacity, the RM is effectively "under-fitted" (not in the traditional sense, but relative to the preference data). As a result, the reward signal received by the policy in the PPO stage is a finite value. This avoids the situation in DPO where the KL regularization constraint is lost due to overfitting the core reward function objective.

5.2.6 Re-analyzing from the Closed-Form Solution of the Optimal Policy

Under the ΨPO framework, the log-odds function $\Psi(q) = \log(q/(1-q))$ chosen by DPO approaches infinity as $q \to 1$, causing the KL constraint to lose its effect. Therefore, a Ψ function that is bounded for $p(y \succ y') \in [0, 1]$ is needed. The simplest non-decreasing function that satisfies this condition is the identity map, i.e., $\Psi(q) = q$. The optimization objective becomes:

$$\max_{\pi}(p^*(y \succ y'|x)) - \tau D_{\text{KL}}(\pi \,\|\, \pi_{\text{ref}}). \tag{56}$$

The standard way to optimize Equation (56) would be to use a reward model in RLHF to provide $r(y) = p(y \succ y')$. However, inspired by the DPO algorithm, bypassing explicit reward modeling and learning directly from preference data is a more appealing approach.

Similar to DPO, we first find the closed-form solution for the optimal policy corresponding to Equation (56). Let $g(y) = \mathbb{E}_{y'\sim\mu}[\Psi(p^*(y \succ y'))]$. The closed-form solution can then be written as:

$$\pi^*(y) \propto \pi_{\text{ref}}(y) \exp\left(\tau^{-1} g(y)\right). \tag{57}$$

As described in Section 5.2.5, for any $y, y' \in \text{supp}(\pi_{\text{ref}})$ (where 'supp' denotes the support set), we have:

$$\frac{\pi^*(y)}{\pi^*(y')} = \frac{\pi_{\text{ref}}(y)}{\pi_{\text{ref}}(y')} \exp(\tau^{-1}(g(y) - g(y'))) \tag{58}$$

Let

$$h^*(y, y') = \log\left(\frac{\pi^*(y)\pi_{\text{ref}}(y')}{\pi^*(y')\pi_{\text{ref}}(y)}\right)$$

By rearranging Equation (58), we get:

$$h^*(y, y') = \tau^{-1}(g(y) - g(y')) \tag{59}$$

This gives us the mathematical relationship that the optimal policy π^* must satisfy. The problem now transforms into finding a policy π. To measure the gap between a target policy π and the optimal policy π^*, we can re-express this as a single optimization problem using the simplest Mean Squared Error (MSE) loss function:

$$L(\pi) = \mathbb{E}_{y,y' \sim \mu}\left[\left(h_\pi(y, y') - \frac{p^*(y \succ \mu) - p^*(y' \succ \mu)}{\tau}\right)^2\right] \tag{60}$$

When the policy is optimal, i.e., $\pi = \pi^*$, we have $L(\pi) = 0$. Therefore, π^* is a global minimum of $L(\pi)$.

5.2.7 Proving the Unique Optimality of the IPO Loss

For this new objective to be effective as a loss function, we need to prove its validity, i.e., that it has an optimal solution and does not get trapped in local minima. Proving the uniqueness of the optimal solution is a prerequisite for ensuring the algorithm's reproducibility. Although $L(\pi)$ has the form of an MSE function, the variables being optimized within it are very complex. The policy π corresponds to a vector of tens of thousands of dimensions, not an independent weight. Moreover, in $h_\pi(y, y')$, $\pi(y)$ and $\pi(y')$ are coupled, making the loss function not a simple parabola with respect to a single variable, but a high-dimensional surface with respect to a high-dimensional vector. Therefore, it is necessary to prove that $L(\pi)$ is a convex function (to guarantee the usability of gradient descent) and that the global minimum corresponds to a unique policy π^* (to ensure a deterministic gradient direction).

Since the form of $L(\pi)$ is rather complex, we transform it into a function of logits over the action space. Let $s(y)$ be the logits such that $\pi_s(y) = \exp(s(y))/\sum_{y' \in J} \exp(s(y'))$. The objective becomes:

$$\mathcal{L}(s) = \mathbb{E}_{y,y' \sim \mu}\left[\left(\frac{p^*(y \succ \mu) - p^*(y' \succ \mu)}{\tau} - (s(y) - s(y')) - \log\left(\frac{\pi_{\text{ref}}(y')}{\pi_{\text{ref}}(y)}\right)\right)^2\right]. \tag{61}$$

Let A be the terms not related to $s(y)$, then:

$$\mathcal{L}(s) = \mathbb{E}_{y,y' \sim \mu}\left[A - (s(y) - s(y'))\right]^2. \tag{62}$$

The optimal policy for the form above satisfies $(s(y) - s(y')) = A$. Expanding Equation (62), and since linear and constant terms do not affect convexity, the object of discussion becomes:

$$Q(s) = \mathbb{E}_{y,y' \sim \mu}\left[(s(y) - s(y'))^2\right] = \sum_{y,y' \in J} \mu(y)\mu(y')(s(y) - s(y'))^2 \tag{63}$$

where $\sum_{y,y' \in J} \mu(y)\mu(y')$ is the expanded form of $\mathbb{E}_{y,y' \sim \mu}$.

Convexity of the Objective Function Equation (63) is a homogeneous quadratic polynomial in the vector s, i.e., a quadratic form. Since it is a weighted sum of non-negative terms $(s(y) - s(y'))^2$ with non-negative weights $\mu(y)\mu(y')$, we can directly conclude that $Q(s) \geq 0$ for all s. Therefore, $Q(s)$ corresponds to a positive semi-definite quadratic form. A positive semi-definite quadratic function is convex, which guarantees that any local minimum is also a global minimum, providing a theoretical foundation for gradient descent.

Strict Convexity and Uniqueness of the Optimal Policy The uniqueness of the optimal solution can be proven by contradiction. Assume that both π' and π^* are optimal policies, then their corresponding losses $L(s')$ and $L(s^*)$ are equal, from which we can derive:

$$\begin{aligned} s^*(y) - s^*(y') &= s'(y) - s'(y') \\ s^*(y) - s'(y) &= s^*(y') - s'(y') \\ &\text{where } y, y' \sim \mu \end{aligned} \tag{64}$$

From the above, all components of the vector $s^* - s'$ must be equal, i.e., $s^* - s' = \lambda e$, where $e = [1, 1, \ldots, 1]$ is a vector of all ones. This means that in the logit space, all global optima form a straight line. Now, we map this conclusion back to the policy space using the softmax function:

$$\pi_{s+\lambda e}(y) = \frac{e^{s(y)+\lambda}}{\sum_{y' \in J} e^{s(y')+\lambda}} \tag{65}$$

$$= \frac{e^\lambda e^{s(y)}}{e^\lambda \sum_{y' \in J} e^{s(y')}} = \pi_s(y). \tag{66}$$

Equation (66) shows that although there are infinitely many optimal solutions in the logit space, they all correspond to the same, unique optimal policy π^*.

5.2.8 Sampled Loss for IPO

The theoretical IPO loss $L(\pi)$ (Equation (60)) contains the unknown true preference probability p^*, making it impossible to use directly for optimization. Therefore, we consider directly sampling the preference outcome $y \succ y'$ to replace the expected preference p^*. This sampled outcome can be represented as a Bernoulli variable $I(y, y')$:

$$I(y, y') = \begin{cases} 1 & \text{with probability } p^*(y \succ y') \\ 0 & \text{with probability } 1 - p^*(y \succ y') \end{cases}$$

Using $I(y, y')$ as an unbiased estimator for $p^*(y \succ \mu) - p^*(y' \succ \mu)$, and substituting it into Equation (60), we obtain a new, theoretical sampled loss:

$$L(\pi) = \mathbb{E}_{y,y' \sim \mu} \left[\left(h_\pi(y, y') - \tau^{-1} I(y, y') \right)^2 \right] \tag{67}$$

This transforms an incomputable theoretical problem based on p^* into an equivalent problem that can be solved by directly sampling preference outcomes (0 or 1).

Next, we need to prove the equivalence of Equation (60) and Equation (67). To prove the equivalence of forms $(a-b)^2$ and $(a-c)^2$, we only need to show that their cross-terms are equivalent, i.e., prove that:

$$\begin{aligned} &\mathbb{E}_{y,y' \sim \mu} \left[h_\pi(y, y') I(y, y') \right] \\ &= \mathbb{E}_{y,y' \sim \mu} \left[h_\pi(y, y') (p^*(y \succ \mu) - p^*(y' \succ \mu)) \right]. \end{aligned}$$

Let $\pi_y = \log(\pi(y))$, $\pi_y^R = \log(\pi_{\text{ref}}(y))$, and $p_y = p^*(y \succ \mu)$. We expand the right-hand side of the equation:[1]

$$\begin{aligned} &\mathbb{E}_{y,y' \sim \mu} \left[h_\pi(y, y') (p^*(y \succ \mu) - p^*(y' \succ \mu)) \right] \\ &= \mathbb{E}_{y,y' \sim \mu} \left[(\pi_y - \pi_{y'} + \pi_{y'}^R - \pi_y^R)(p_y - p_{y'}) \right] \\ &= \mathbb{E}_{y,y' \sim \mu} \left[\pi_y p_y - \pi_y p_{y'} - \pi_{y'} p_y + \pi_{y'} p_{y'} + \pi_{y'}^R p_y - \pi_{y'}^R p_{y'} - \pi_y^R p_y + \pi_y^R p_{y'} \right] \\ &= \mathbb{E}_{y \sim \mu} \left[(2p_y - 1)\pi_y - (2p_y - 1)\pi_y^R \right], \end{aligned}$$

where y and y' are i.i.d., so $\mathbb{E}_{y \sim \mu}[p_y] = \mathbb{E}_{y' \sim \mu}[p_{y'}] = \frac{1}{2}$, and the symmetry of y and y' is used, e.g., $\mathbb{E}_{y,y'}[\pi_{y'} p_y] = \mathbb{E}_{y,y'}[\pi_y p_{y'}]$.

For the left-hand side, we perform the following transformation:

$$\begin{aligned} &\mathbb{E}_{y,y' \sim \mu}[h_\pi(y, y') I(y, y')] \\ &= \mathbb{E}_{y,y' \sim \mu}[(\pi_y - \pi_{y'} + \pi_{y'}^R - \pi_y^R) I(y, y')] \\ &= \mathbb{E}_{y \sim \mu} \left[(\pi_y - \pi_y^R) \mathbb{E}_{y' \sim \mu}[I(y, y') | y] \right] + \mathbb{E}_{y' \sim \mu} \left[(-\pi_{y'} + \pi_{y'}^R) \mathbb{E}_{y \sim \mu}[I(y, y') | y'] \right] \\ &= \mathbb{E}_{y \sim \mu}[(\pi_y - \pi_y^R) p_y] + \mathbb{E}_{y' \sim \mu}[(-\pi_{y'} + \pi_{y'}^R)(1 - p_{y'})] \\ &= \mathbb{E}_{y \sim \mu}[(2p_y - 1)\pi_y - (2p_y - 1)\pi_y^R], \end{aligned}$$

where $\mathbb{E}_{y' \sim \mu}[I(y, y') | y] = p_y$ and $\mathbb{E}_{y \sim \mu}[I(y, y') | y'] = 1 - p_{y'}$.

For an empirical dataset $\mathcal{D}$, each data point can contribute two data points: $(y, y', I(y, y')) = (y_w^i, y_l^i, 1)$ and $(y, y', I(y, y')) = (y_l^i, y_w^i, 0)$. This symmetry is important for stabilizing the variance of the loss. The overall loss takes the form:

$$\begin{aligned} &\frac{1}{2} \mathbb{E}_{(y_w, y_l) \sim \mathcal{D}} \left[(h_\pi(y_w, y_l) - \tau^{-1} \cdot 1)^2 + (h_\pi(y_l, y_w) - \tau^{-1} \cdot 0)^2 \right] \\ &= \frac{1}{2} \mathbb{E}_{(y_w, y_l) \sim \mathcal{D}} \left[(h_\pi(y_w, y_l) - \tau^{-1})^2 + h_\pi(y_l, y_w)^2 \right] \\ &= \frac{1}{2} \mathbb{E}_{(y_w, y_l) \sim \mathcal{D}} \left[(h_\pi(y_w, y_l) - \tau^{-1})^2 + (-h_\pi(y_w, y_l))^2 \right] \end{aligned}$$

The form above, $\frac{1}{2}[(a-c)^2 + a^2]$, is minimized at the same point as $(a - c/2)^2$ (they differ only by a constant term $c^2/4$ that is independent of a). Therefore, the loss above is equivalent to minimizing:

$$\mathbb{E}_{(y_w, y_l) \sim \mathcal{D}} \left[\left(h_\pi(y_w, y_l) - \frac{\tau^{-1}}{2} \right)^2 \right]. \tag{68}$$

[1] In the arXiv version of the IPO paper, there is a typo in this formula. A correction has been made here.

5.2.9 IPO Algorithm Analysis

The most fundamental difference between the IPO and DPO algorithms lies in the design of their loss functions. DPO's logarithmic loss function theoretically reaches its minimum only when the logit difference between y_w and y_l approaches infinity. This forces the model to endlessly increase the gap between the two, leading to overfitting. IPO's Mean Squared Error loss, $\left(h_\pi(y_w, y_l) - \frac{\tau^{-1}}{2}\right)^2$, is completely different. It sets a clear, finite target for the optimization. As soon as the policy's $h_\pi(y_w, y_l)$ reaches the specific value $\frac{\tau^{-1}}{2}$ determined by τ, the loss drops to zero, the gradient vanishes, and optimization stops. The regularization term $\frac{\tau^{-1}}{2}$ is not directly introduced but is derived step-by-step by finding a more suitable expression for the optimal policy's loss and then using a Bernoulli sampling distribution. This "stop when the target is reached" mechanism fundamentally solves DPO's insatiable tendency to overfit, representing an elegant application of optimization theory.

5.3 Other DPO Variants

This section introduces some interesting variants of the DPO algorithm, each of which identifies a shortcoming of DPO and proposes a corresponding improvement.

5.3.1 KTO: Model Alignment as Prospect Theoretic Optimization

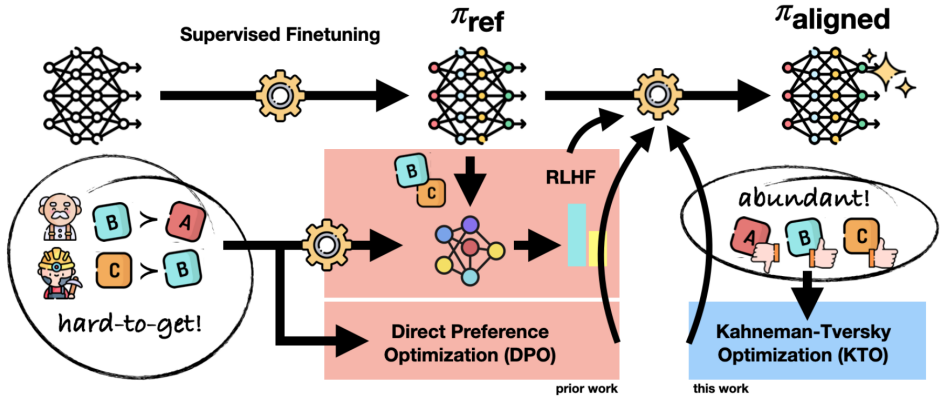

Figure 5: This figure, from v1 of the KTO research paper, illustrates the difference from previous work: while previous methods require paired data, KTO can work by only knowing whether the current data point is desirable or not.

Motivation DPO and other frameworks always require paired preference data, which makes data annotation difficult. Inspired by "prospect theory" from behavioral economics—the idea that humans evaluate gains and losses relative to a reference point rather than in absolute terms—the Khaneman-Tversky Optimization (KTO) [Ethayarajh et al., 2024] algorithm was designed. It does not require paired preference data, only knowledge of whether the current data is desirable or undesirable, to perform alignment. It aims to teach the model to judge whether a response is better or worse than an average response, thereby improving data utilization and reducing the difficulty of data collection.

Optimization Objective

$$L_{\text{KTO}}(\pi_\theta, \pi_{\text{ref}}) = \mathbb{E}_{x,y\sim\mathcal{D}}[w(y)(1 - v_{\text{KTO}}(x, y; \beta))] \tag{69}$$

$$r_{\text{KTO}}(x, y) = \beta \log \frac{\pi_\theta(y|x)}{\pi_{\text{ref}}(y|x)} \tag{70}$$

$$z_{\text{ref}} = \mathbb{E}_{x'\sim\mathcal{D}}[\beta \,\text{KL}(\pi_\theta(y'|x') \,\|\, \pi_{\text{ref}}(y'|x'))] \tag{71}$$

$$v_{\text{KTO}}(x, y; \beta) = \begin{cases} \sigma(r_{\text{KTO}}(x, y) - z_{\text{ref}}) & \text{if } y \sim y_{\text{desirable}}|x \\ \sigma(z_{\text{ref}} - r_{\text{KTO}}(x, y)) & \text{if } y \sim y_{\text{undesirable}}|x \end{cases} \tag{72}$$

$$w(y) = \begin{cases} \lambda_D & \text{if } y \sim y_{\text{desirable}}|x \\ \lambda_U & \text{if } y \sim y_{\text{undesirable}}|x \end{cases} \tag{73}$$

$$\frac{\lambda_D n_D}{\lambda_U n_U} \in \left[1, \frac{3}{2}\right] \tag{74}$$

Analysis of the Objective In Equation (69)[2], observe the term v_{KTO}. Its form $\sigma(A - B)$ is fundamentally similar to DPO's core Bradley-Terry objective (Eq. (35)). The difference is that when y is an undesirable data point, the comparison becomes $p(z_{ref} \succ r_{KTO}(x, y))$. r_{KTO} can be understood as the utility score of y. The key difference is that the other party in the comparison is no longer another real response, but the reference point z_{ref}.

The term $w(y)$ represents the weight of the current sample, primarily to handle data imbalance issues in the training set. It needs to be adjusted according to Equation (74), which is the recommended ratio for λ_D and λ_U from the KTO study. By default, both are 1, and n_D and n_U are the numbers of desirable and undesirable data points, respectively.

The reference point z_{ref} represents the average utility of the current policy π_θ and is pre-computed on a separate, held-out dataset and shared across a minibatch.

The overall design philosophy of KTO is to compare the utility score r_{KTO} of each response y with the average utility z_{ref} of the current batch to determine if y is an outperformer or an underperformer. From another perspective, since z_{ref} is treated as a constant within a single update, the direct driving force for optimization comes from changes in r_{KTO}, meaning the objective is to adjust the probability ratio of π_θ to π_{ref} for a specific response y. Since $w(y)$ is just a fixed weight and not a finite target like in IPO, KTO's loss function also incentivizes the model to infinitely increase the gap between r_{KTO} and z_{ref}, and thus may face similar overfitting issues as DPO.

5.3.2 Contrastive Preference Optimization: Pushing the Boundaries of LLM Performance in Machine Translation

Motivation The DPO algorithm requires inference from a reference model, which introduces additional computational and memory overhead. The goal is to eliminate π_{ref} from the optimization objective through transformations, ultimately resulting in an objective that depends only on π_θ.

Optimization Objective

$$\mathcal{L}(\pi_\theta; U) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\Big[\log\sigma\left(\beta\log\pi_\theta(y_w|x) - \beta\log\pi_\theta(y_l|x)\right)\Big] \tag{75}$$

$$\min_\theta \underbrace{\mathcal{L}(\pi_\theta, U)}_{\mathcal{L}_{\text{prefer}}} - \underbrace{\mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\log\pi_\theta(y_w|x)]}_{\mathcal{L}_{\text{NLL}}} \tag{76}$$

The overall optimization objective for Contrastive Preference Optimization (CPO) [Xu et al., 2024] is Equation (76), which consists of two parts: a preference optimization objective $\mathcal{L}_{\text{prefer}}$ derived from the DPO algorithm, and an SFT regularization term $\mathcal{L}_{\text{NLL}}$.

Analysis of $\mathcal{L}_{\text{prefer}}$ The preference optimization objective $\mathcal{L}_{\text{prefer}}$ is given by Equation (75). This is the core of the CPO algorithm, where U represents a uniform prior distribution. This can be understood as an extremely poor model where the logits for all actions are identical. In this case, the terms $\log(\pi_{\text{ref}}(y_w|x))$ and $\log(\pi_{\text{ref}}(y_l|x))$ in the DPO objective cancel each other out, yielding the new objective form.

[2]The objective function shown here is from v1 of the KTO paper on arXiv, which is more intuitive for understanding.

Proving the Consistency with DPO's Objective When π_{ref} reaches its ideal state, denoted as π_w, it can perfectly align with the data's preference distribution. For any data point (x, y_w, y_l), we have $\pi_w(y_w|x) = 1$ and $0 \leq \pi_w(y_l|x) \leq 1$. Therefore, the DPO objective can be transformed as follows:

$$\begin{aligned}\mathcal{L}(\pi_\theta; \pi_w) &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_w(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_w(y_l|x)}\right)\right] \\ &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\pi_\theta(y_w|x) - \beta\log\pi_w(y_w|x)\right) + \beta\log\pi_w(y_l|x)\right]\end{aligned}$$

Further expanding the sigmoid function:

$$\begin{aligned}\mathcal{L}(\pi_\theta; \pi_w) &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\left(\frac{1}{1+e^{-\beta\log\pi_\theta(y_w|x)+\beta\log\pi_w(y_l|x)}} - \beta\log\pi_w(y_l|x)\right)\right] \\ &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\left(\frac{1}{1+\frac{\pi_\theta(y_w|x)^{-\beta}}{\pi_w(y_l|x)^{-\beta}}}\right)\right] \\ &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log\pi_\theta(y_w|x)^\beta + \log\pi_w(y_l|x)^\beta - \log\left(\pi_\theta(y_w|x)^\beta\cdot\pi_w(y_l|x)^\beta + \pi_\theta(y_l|x)^\beta\right)\right]\end{aligned}$$

In the form above, π_w is a fixed model, and $\log\pi_w(y_l|x)$ does not participate in the gradient update, so it can be considered an irrelevant term, denoted by C. Optimizing $\mathcal{L}(\pi_\theta; \pi_w)$ is equivalent to optimizing:

$$\begin{aligned}\mathcal{L}'(\pi_\theta; \pi_w) &\triangleq \mathcal{L}(\pi_\theta; \pi_w) + \underbrace{\mathbb{E}_{(x,y_l)\sim\mathcal{D}}[\log\pi_w(y_l|x)^\beta]}_{C \text{ in the Theorem}} \\ &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}[\log\pi_\theta(y_w|x)^\beta - \log(\pi_\theta(y_w|x)^\beta\cdot\pi_w(y_l|x)^\beta + \pi_\theta(y_l|x)^\beta)]\end{aligned}$$

Since π_w is an ideal policy that is practically unobtainable, we consider the upper bound of $\mathcal{L}'(\pi_\theta, \pi_w)$ by using the fact that $0 \leq \pi_w(y_l|x) \leq 1$:

$$\begin{aligned}\mathcal{L}'(\pi_\theta; \pi_w) &\leq -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}[\log\pi_\theta(y_w|x)^\beta - \log(\pi_\theta(y_w|x)^\beta\cdot 1 + \pi_\theta(y_l|x)^\beta)] \\ &= -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}[\log\sigma(\beta\log\pi_\theta(y_w|x) - \beta\log\pi_\theta(y_l|x))] \\ &= \mathcal{L}(\pi_\theta; U).\end{aligned}$$

Therefore, $\mathcal{L}(\pi_\theta; U)$ is an upper bound for $\mathcal{L}(\pi_\theta, \pi_w) + C$, where $C = \mathbb{E}_{(x,y_l)\sim\mathcal{D}}[\log\pi_w(y_l|x)^\beta]$. In practice, the objective being optimized is $\mathcal{L}(\pi_\theta; U)$, which indirectly optimizes its lower bound $\mathcal{L}(\pi_\theta; \pi_w)$. This proves the consistency between the $\mathcal{L}_{\text{prefer}}$ objective and the DPO objective.

Analysis of $\mathcal{L}_{\text{NLL}}$ Although the design of $\mathcal{L}_{\text{prefer}}$ is more efficient, eliminating the computation of π_w (i.e., π_{ref}) also removes the implicit KL divergence constraint. This could lead to reward hacking and policy degradation. Therefore, an explicit Behavior Cloning constraint is introduced, which is essentially a KL divergence constraint. It can be understood as requiring that π_θ, during optimization, does not deviate too much from the distribution of y_w under the ideal policy π_w. The objective becomes:

$$\min_\theta \mathcal{L}(\pi_\theta, U) \quad \text{s.t.} \quad \mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\text{KL}(\pi_w(\cdot|x)\|\pi_\theta(\cdot|x))] < \epsilon \tag{77}$$

Using the method of Lagrange multipliers, the constraint can be incorporated into the objective:

$$\min_\theta \mathcal{L}(\pi_\theta, U) + \lambda\cdot\mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\text{KL}(\pi_w(\cdot|x)\|\pi_\theta(\cdot|x))] \tag{78}$$

where λ is a hyperparameter. Setting $\lambda = 1$, expanding the KL divergence, and substituting the ideal properties of π_w:

$$\begin{aligned}\mathcal{L}_{\text{CPO}} &= \mathcal{L}(\pi_\theta, U) + \mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\text{KL}(\pi_w(\cdot|x)\|\pi_\theta(\cdot|x))] \\ &= \mathcal{L}(\pi_\theta, U) + \mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\sum_y \pi_w(y|x)\log\frac{\pi_w(y|x)}{\pi_\theta(y|x)}] \\ &= \mathcal{L}(\pi_\theta, U) + \mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\pi_w(y_w|x)\log\frac{\pi_w(y_w|x)}{\pi_\theta(y_w|x)}] \quad (\text{since } \pi_w(y|x) = 0 \text{ for } y \neq y_w) \\ &= \mathcal{L}(\pi_\theta, U) + \mathbb{E}_{(x,y_w)\sim\mathcal{D}}[1\cdot\log\frac{1}{\pi_\theta(y_w|x)}] \quad (\text{since } \pi_w(y_w|x) = 1) \\ &= \mathcal{L}(\pi_\theta, U) - \mathbb{E}_{(x,y_w)\sim\mathcal{D}}[\log(\pi_\theta(y_w|x))]\end{aligned}$$

This is the final form of the CPO optimization objective.

Analysis of the Objective Looking at the form of the CPO algorithm (Eq. (76)), if we view $\mathcal{L}_{\text{prefer}}$ as an implicit reward signal, similar to DPO, the overall expression resembles the PPO-ptx objective from Section 4.1.4. Both are composed of a reward term and a next-token prediction task. However, the motivation for PPO-ptx is to mitigate the degradation of pre-training performance, while for CPO, it is to prevent reward hacking.

5.3.3 2D-DPO: Scaling Direct Preference Optimization with 2-Dimensional Supervision

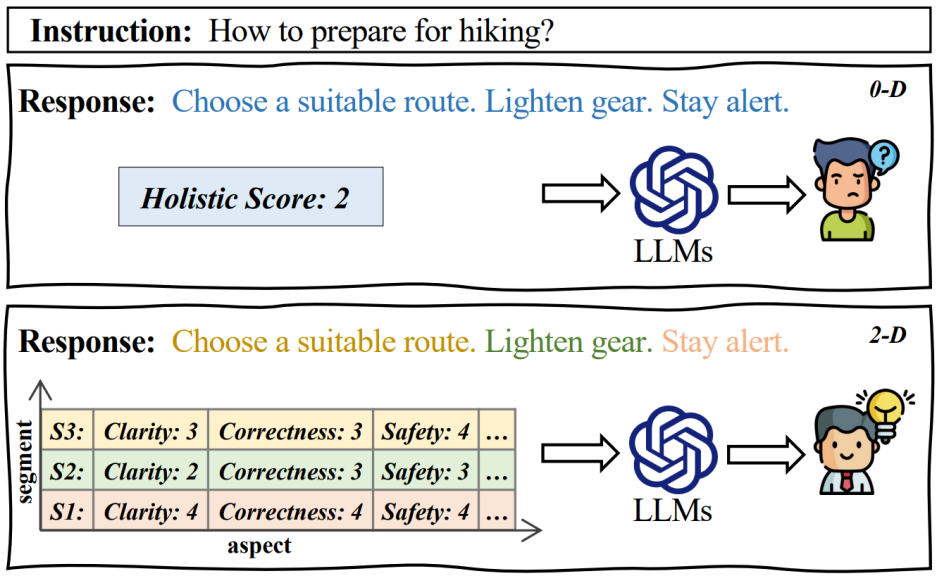

Figure 6: This figure, from the 2D-DPO research paper, illustrates the difference between the 2D-DPO and DPO algorithms.

Motivation Real human preferences are complex. Algorithms like PPO or DPO learn preferences for an entire sentence, which is typically sampled from an SFT model. However, the constituent elements within a sentence contribute differently to the overall preference. A more fine-grained preference alignment should be considered. Therefore, this work proposes splitting the complete sentence into different segments and annotating each segment with multi-dimensional preferences.

Optimization Objective

$$\begin{aligned}\mathcal{L}_{\text{group}}(\pi_\theta, \mathcal{D}) = \\ -\mathbb{E}_{(\tau_w,\tau_l)\sim\mathcal{D}}\Bigg[\sum_{k=0}^{N-1}\log\sigma\Bigg(\beta\sum_{t=n_k}^{n_k+l_k} r_{w,k}\log\frac{\pi_\theta(\mathbf{a}_t^w|\mathbf{s}_t^w)}{\pi_{\text{ref}}(\mathbf{a}_t^w|\mathbf{s}_t^w)} \\ -\beta\sum_{t=n_k}^{n_k+l_k} r_{l,k}\log\frac{\pi_\theta(\mathbf{a}_t^l|\mathbf{s}_t^l)}{\pi_{\text{ref}}(\mathbf{a}_t^l|\mathbf{s}_t^l)}\Bigg)\Bigg]\end{aligned} \tag{79}$$

Analysis of the Objective The optimization objective of the 2D-DPO [Li et al., 2024] algorithm is extremely similar in form to that of DPO and can be seen as a generalization of the DPO loss at the segment level. The core idea is to decompose the implicit reward calculation for the entire sequence in DPO into a calculation for each corresponding segment pair k. The most fundamental difference from DPO lies in the preference utility function within the sigmoid, which becomes a weighted sum of the preference utility functions for that segment. The weights, $r_{w,k}$ and $r_{l,k}$, represent the reward scores for that segment across different dimensions (e.g., helpfulness, harmlessness). In this way, 2D-DPO no longer makes a vague good/bad judgment on the entire sentence but instead precisely delivers the preference signal to the specific segments that caused the preference, achieving a more fine-grained model alignment.

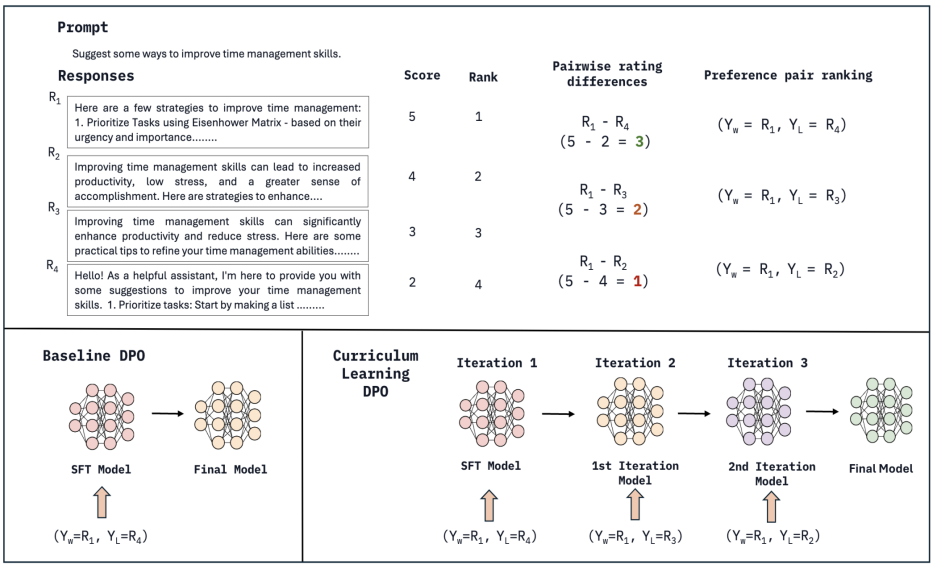

Figure 7: This figure, from the Curry-DPO research paper, illustrates the iterative curriculum learning paradigm proposed by the algorithm.

5.3.4 Curry-DPO: Enhancing Alignment using Curriculum Learning & Ranked Preferences

Motivation In large-scale alignment, a single prompt often has multiple ranked responses. For example, in RLHF, each data point might have between 4 and 9 responses. However, most preference learning methods still operate in a pairwise manner. Therefore, Curry-DPO [Pattnaik et al., 2024] proposes a curriculum learning paradigm for preference learning, which prioritizes learning from preference pairs that are far apart in ranking before moving on to pairs that are closer in preference.

Optimization Objective

$$\mathcal{L}(\pi_\theta^{i+1};\pi_\theta^i) = -\mathbb{E}_{(x,y_w^{i+1},y_l^{i+1})\sim\mathcal{D}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta^{i+1}(y_w^{i+1}|x)}{\pi_\theta^i(y_w^{i+1}|x)} - \beta\log\frac{\pi_\theta^{i+1}(y_l^{i+1}|x)}{\pi_\theta^i(y_l^{i+1}|x)}\right)\right] \tag{80}$$

Analysis of the Objective Curry-DPO can be seen as a more advanced training strategy for DPO. It achieves better alignment results than static DPO training through a data scheduling approach that moves from easy to hard examples and through model iteration. The "easy-to-hard" curriculum is implemented by first training on pairs with large preference gaps (e.g., rank 1 vs. rank 9), which are easier for the model to learn. As training progresses, it gradually introduces pairs with smaller preference gaps (e.g., rank 1 vs. rank 2). The model from the previous stage, π_θ^i, is used as the reference model for the current stage, π_θ^{i+1}, creating a self-improving loop.

6 GRPO

This chapter introduces the Group Relative Policy Optimization (GRPO) algorithm and related technical details of DeepSeek-R1 [Guo et al., 2025]. GRPO was proposed in the research work of DeepSeek, primarily to enhance the mathematical reasoning capabilities of models. Therefore, we will first introduce the GRPO algorithm directly, and then discuss the research work on DeepSeek-R1.

6.1 Motivation

The role of the value model in the PPO algorithm is to introduce a baseline, which, in conjunction with the advantage function, helps the reward signal during training achieve a balance between bias and variance. However, the value

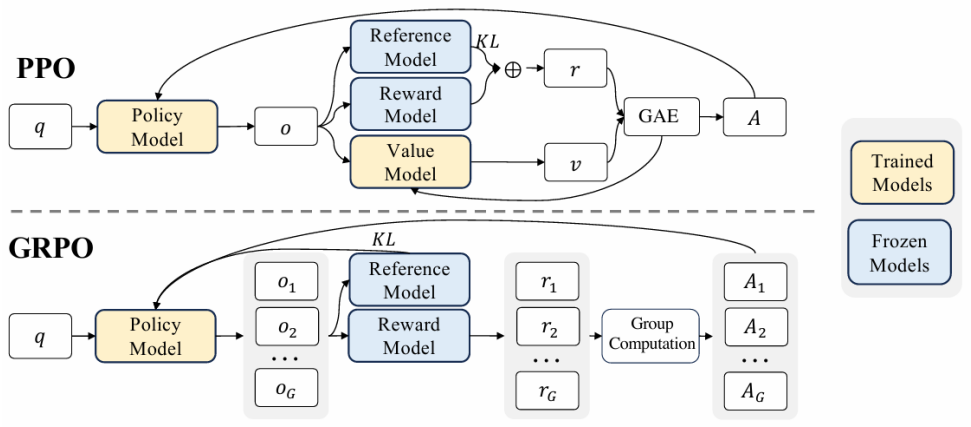

Figure 8: This figure, from the DeepSeek-Math research paper, illustrates the high-level processes of PPO and GRPO. Compared to PPO, GRPO's process abandons the value model; its baseline is estimated from group scores.

model not only introduces memory and computational burdens due to its large parameter size, but also complicates the training of the value function for each token, as only the final token receives a score from the reward model in the RL process.

To address the issues with PPO's value model, Group Relative Policy Optimization (GRPO) was proposed. It uses the average reward from multiple samples as a baseline to optimize the policy.

6.2 From PPO to GRPO

The core idea of the GRPO algorithm is to use the average reward of multiple intra-group samples as a baseline, replacing the learnable Value Model in PPO. However, its optimization objective is still primarily based on PPO's $L^{CLIP}(\theta)$ (Equation (10)). To address the issue of reward sparsity, the PPO objective needs to be rewritten from a token-level perspective:

$$\begin{aligned}\mathcal{J}_{\text{PPO}}(\theta) = \mathbb{E}_{q\sim P(Q),o\sim\pi_{\theta_{\text{old}}}(O|q)} \frac{1}{|o|}\sum_{t=1}^{|o|}\Bigg[\min\Bigg(&\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{\text{old}}}(o_t|q,o_{<t})}A_t,\\ &\text{clip}\left(\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\theta_{\text{old}}}(o_t|q,o_{<t})},1-\epsilon,1+\epsilon\right)A_t\Bigg)\Bigg],\\ \text{where}\quad r_t = r_\phi(q,o_{\le t}) &- \beta\log\frac{\pi_\theta(o_t|q,o_{<t})}{\pi_{\text{ref}}(o_t|q,o_{<t})}.\end{aligned} \tag{81}$$

While PPO samples once from $\pi_{\theta_{old}}$ for each query q within a batch/epoch for a single learning step, GRPO performs multiple samplings to obtain a group of outputs $\{o_1, o_2, \ldots, o_G\}$. The optimization objective is transformed into the following form:

$$\begin{aligned}\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(O|q)}\Bigg[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\Bigg(\min\Bigg[&\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})}\hat{A}_{i,t},\\ &\text{clip}\left(\frac{\pi_\theta(o_{i,t}|q,o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t}|q,o_{i,<t})},1-\epsilon,1+\epsilon\right)\hat{A}_{i,t}\Bigg]\\ &-\beta\mathbb{D}_{\text{KL}}[\pi_\theta\|\pi_{\text{ref}}]\Bigg)\Bigg],\\ \text{where}\quad \mathbb{D}_{\text{KL}}[\pi_\theta\|\pi_{\text{ref}}] = &\frac{\pi_{\text{ref}}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} - \log\frac{\pi_{\text{ref}}(o_{i,t}|q,o_{i,<t})}{\pi_\theta(o_{i,t}|q,o_{i,<t})} - 1.\end{aligned} \tag{82}$$

Here, $\hat{A}_{i,t}$ is the advantage calculated from the intra-group relative reward, which will be explained in the subsequent section. Additionally, the GRPO algorithm places the KL regularization term between the policy and reference directly into the optimization objective, avoiding complex calculations for the advantage $\hat{A}_{i,t}$. Furthermore, the form of the KL divergence is replaced with an unbiased estimator that has lower variance.

6.3 The Advantage Function of GRPO

The advantage function in GRPO is defined in two ways, corresponding to two types of RL supervision: outcome-based supervision and process-based supervision.

Outcome-based Supervision For each query q, a group of outputs $\{o_1, o_2, \ldots, o_G\}$ is sampled. A reward model is used to score these outputs, yielding a corresponding group of rewards $r = \{r_1, r_2, \ldots, r_G\}$. Then, the reward for each output o_i is normalized within the group. Since the KL regularization term has been moved to the final optimization objective, the advantage function is simply an identity mapping, and all tokens are assigned the standardized reward, i.e., $\hat{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$. The policy is then optimized by maximizing Equation (82).

Process-based Supervision Outcome-based supervision, which only provides a reward at the final output, may perform poorly and be inefficient for training on mathematical tasks. Therefore, process-based supervision was introduced. For each query q and a group of sampled outputs $\{o_1, o_2, \ldots, o_G\}$, a process reward model scores each step of o_i, resulting in a reward sequence:

$$\mathbf{R} = \left\{ \{r_1^{\text{index}(1)}, \ldots, r_1^{\text{index}(K_1)}\}, \ldots, \{r_G^{\text{index}(1)}, \ldots, r_G^{\text{index}(K_G)}\} \right\}$$

where 'index(j)' denotes the j-th token, and K_i is the total number of tokens in the current output. Similarly, the reward sequence is normalized within the group, i.e., $\tilde{r}_i^{\text{index}(j)} = \frac{r_i^{\text{index}(j)} - \text{mean}(\mathbf{R})}{\text{std}(\mathbf{R})}$. The token-level advantage under process-based supervision is calculated as $\hat{A}_{i,t} = \sum_{\text{index}(j) \geq t} \tilde{r}_i^{\text{index}(j)}$. The policy is then optimized by maximizing Equation (82).

In the practice of DeepSeek-Math, process-based supervision was found to be superior to outcome-based supervision.

6.4 The KL Regularization Term

GRPO's regularization term is derived from a blog post [Schulman, 2020] that analyzes three different KL divergence estimators. Following the blog's notation, let $r = \frac{\pi_{\text{ref}}}{\pi_\theta}$. In PPO, the regularization term is denoted as $k_1 = -\log r$, while in GRPO, it is $k_3 = r - \log r - 1$. There is another estimator, $k_2 = \frac{(\log r)^2}{2}$. Let $q = \pi_\theta$ and $p = \pi_{\text{ref}}$, so $r = p/q$.

The variance of KL divergence estimation mainly comes from two regions: the tail and the center. The center region is where $r = 1$, meaning q and p are close, which is the common case. The tail regions are where $r \to \infty$ and $r \to 0$, meaning q deviates extremely from p, which is the uncommon case. Therefore, variance analysis primarily focuses on the center region. This section will analyze the bias and variance of these three KL estimators.

Why KL Divergence Needs Estimation The definition of KL divergence is $\mathbb{D}_{\text{KL}} = \sum_x q(x) \left[\log \frac{q(x)}{p(x)}\right] = \mathbb{E}_{x\sim q}\left[\log \frac{q(x)}{p(x)}\right] = -\mathbb{E}_{x\sim q}[\log r]$. The expectation in this definition cannot be calculated exactly because it requires summing over all possible outputs x. Therefore, it must be estimated using Monte Carlo sampling from q.

$k_1 = -\log r$: Unbiased, High-Variance Estimator The expected value of the k_1 estimator is exactly equal to the true KL divergence, making k_1 a naive unbiased estimator. As a single-sample estimator, its derivative is non-zero at the center region ($r = 1$), leading to high variance.

$k_2 = \frac{(\log r)^2}{2}$: Biased, Low-Variance Estimator The expected value of the k_2 estimator can be written as:

$$\begin{aligned} \mathbb{E}[k_2] &= \mathbb{E}\left[\frac{(\log r)^2}{2}\right] \\ &= \frac{\mathbb{E}[(\log r)^2]}{2} \\ &= \frac{\text{Var}(\log r) + (\mathbb{E}[\log r])^2}{2} \end{aligned} \tag{83}$$

From this form, it is clear that the k_2 estimator is biased, except in the trivial case where the variance is zero and $\mathbb{D}_{\text{KL}} = 0, 2$. The derivative of the k_2 estimator is zero at $r = 1$, meaning it is a low-variance estimator in most scenarios.

$k_3 = r - \log r - 1$**: Unbiased, Low-Variance Estimator** The expected value of the k_3 estimator can be transformed as follows:

$$\begin{aligned}\mathbb{E}_{x\sim q}[k_3] &= \mathbb{E}_{x\sim q}\left[r - 1 - \log r\right] \\ &= \mathbb{E}_{x\sim q}\left[r\right] - 1 - \mathbb{E}_{x\sim q}\left[\log r\right] \\ &= \sum_x q(x)\frac{p(x)}{q(x)} - 1 + \mathbb{D}_{\text{KL}}(q\|p) \\ &= \sum_x p(x) - 1 + \mathbb{D}_{\text{KL}}(q\|p) \\ &= 1 - 1 + \mathbb{D}_{\text{KL}}(q\|p) = \mathbb{D}_{\text{KL}}(q\|p)\end{aligned} \tag{84}$$

From its final expected form, the k_3 estimator is unbiased. Its derivative at $r = 1$ is also zero, making k_3 an unbiased, low-variance estimator.

$\mathbb{D}_{\text{KL}}(q\|p)$ **vs.** $\mathbb{D}_{\text{KL}}(p\|q)$ In KL divergence, $\mathbb{D}_{\text{KL}}(p\|q)$ corresponds to fitting distribution p to distribution q, penalizing regions where p fails to cover q. LLM alignment training is analogous to rejection sampling. The SFT model is already a distribution that performs well in natural language. Through reward signals, the model explores within the existing distribution of π_{ref}, attempting to make the distribution sharper and biased towards preferred outputs. Therefore, alignment training uses the form $\mathbb{D}_{\text{KL}}(q\|p)$ (i.e., $\mathbb{D}_{\text{KL}}(\pi_\theta\|\pi_{ref})$). Since p is fixed, this means q fits some expectation within the region of p, penalizing regions in q not covered by p. This form of KL divergence is known as Reverse KL; the other is Forward KL. This also highlights the important property of KL divergence's asymmetry.

6.5 Analysis of the GRPO Objective

From the form of its optimization objective, the core of the GRPO algorithm is fundamentally the same as PPO: using the policy gradient theorem to perform gradient ascent on an advantage signal. However, GRPO places the KL divergence term directly into the final optimization objective. In terms of the advantage function, GRPO uses the group-average reward to replace the value network, changing the object of comparison from an estimate of future returns to the group average. For outcome-based supervision, treating a whole sentence as a single unit where each token contributes equally is a rather coarse training method. Process-based supervision, while requiring higher-quality data and thus additional data collection costs, has been experimentally proven to be effective for this new advantage estimation. If PPO tells the policy "how much better/worse you are than my expectation," then GRPO tells the policy "how much better or worse the current action is, relative to the average of its peers within the group."

Although GRPO is demonstrated to improve the reasoning ability of models, its essence is still alignment. The goal simply shifts from aligning with the subjective, abstract concept of human preference to aligning with objective, verifiable ground truth in specific tasks through autonomous exploration of reasoning strategies. Therefore, GRPO can be seen as a fact-oriented alignment algorithm.

6.6 DeepSeek-R1-Zero

DeepSeek-R1-Zero focuses on RL training without any initial SFT data, with a key interest in the self-evolution of the LLM through pure reinforcement learning.

6.6.1 RL Algorithm Used

Continuing the philosophy from the DeepSeek-Math work, the GRPO algorithm is used to optimize the policy, with the objective function given by Equation (82). To further mitigate reward hacking and improve training efficiency, neither process-based nor outcome-based supervision models were used. Instead, the reward signal is composed of an accuracy reward and a format reward. The accuracy reward is provided by a rule-based accuracy reward model, which evaluates whether the LLM's answer is correct. The format reward is given by a format reward model, which enforces that the model's thought process is enclosed within `<think>` and `</think>` tags. However, further technical details—such as how the accuracy reward, which can be seen as a more extreme form of outcome-based supervision, avoids reward hacking; how the completeness of the thought process is ensured; and how the model is practically applied after RL training—are not publicly known.

6.6.2 Training Template & RL Philosophy

The training template for DeepSeek-R1-Zero is shown in Table 1. It requires the model to first generate a reasoning process and then provide the final answer. The researchers state that they deliberately limited the constraints to structural

A conversation between User and Assistant. The user asks a question, and the Assistant solves it. The assistant first thinks about the reasoning process in the mind and then provides the user with the answer. The reasoning process and answer are enclosed within `<think>...</think>` and `<answer>...</answer>` tags, respectively, i.e., `<think>` reasoning process here `</think><answer>` answer here `</answer>`. User: prompt. Assistant:

Table 1: The template for DeepSeek-R1-Zero. The prompt is replaced with a specific reasoning question during training.

formats rather than imposing Content-Specific Biases. They list two common examples of using content-specific constraints:

Mandating Reflective Reasoning This is a common advanced thinking strategy that requires the model to repeatedly review, check, and correct its own thought process or output. For example, forcing the training template to include phrases like "Let me check my steps."

Promoting Particular Problem-Solving Strategies This requires the model to use a specific strategy when solving a problem, such as forcing it to first define variables, then set up equations, and finally solve them when faced with a word problem.

Biases & The Expectation for RL Forcing reflection can be redundant and inefficient for simple problems, and a promoted specific strategy may not always be the best one. Therefore, a philosophy of **minimally constrained reinforcement learning** was adopted. It only defines the framework, not the content, with the expectation that the LLM will learn to reflect and check on its own when faced with genuinely complex problems, actively explore new solutions, and independently discover superior strategies. This self-evolution, driven by intrinsic need rather than external enforcement, is what may truly lead to higher-level reasoning abilities and allows for better observation of the LLM's self-evolution under pure reinforcement learning.

6.6.3 Other Details

Self-Evolution The study mentions that during training, DeepSeek-R1-Zero's "thinking time" continuously improved, and the token length of its reasoning process also significantly increased. Furthermore, as thinking time increased, complex behaviors spontaneously emerged, such as the reflective behaviors mentioned above and the exploration of better problem-solving strategies.

Aha Moment As shown in Table 2, in an intermediate version during training, DeepSeek-R1-Zero learned to allocate more thinking time to a problem by re-evaluating its initial approach. This once again emphasizes that instead of explicitly teaching an LLM how to solve problems, it is better to provide it with appropriate rewards and let it autonomously explore better solution strategies.

Drawbacks DeepSeek-R1-Zero suffered from issues like poor readability and language mixing, which led to the subsequent research on DeepSeek-R1, which uses a cold start data stage.

6.7 DeepSeek-R1

The development of DeepSeek-R1 consists of four stages: a cold start stage, inference-oriented RL, rejection sampling & supervised fine-tuning, and a secondary RL for preference alignment.

6.7.1 Cold Start

To avoid the instability of the base model during the early cold start phase of RL training, zero-shot and long Chain-of-Thought (CoT) prompting was used to have DeepSeek-R1-Zero generate detailed answers, reflections, and verifications. Data was collected in a highly readable format and post-processed by human annotators to fine-tune DeepSeek-V3-Base, which served as the starting point for RL training. This approach improves the readability of the output from the very beginning of the cold start phase and provides a foundation for iterative model training.

6.7.2 Inference-Oriented RL

This stage uses the same RL training process as DeepSeek-R1-Zero, focusing on enhancing reasoning abilities, especially in reasoning-intensive tasks like coding, math, and logic. During this process, CoT often exhibited language mixing,

Question: If $a > 1$, then the sum of the real solutions of $\sqrt{a - \sqrt{a + x}} = x$ is equal to

Response: `<think>`
To solve the equation $\sqrt{a - \sqrt{a + x}} = x$, let's start by squaring both ...

$$(\sqrt{a - \sqrt{a + x}})^2 = x^2 \implies a - \sqrt{a + x} = x^2.$$

Rearrange to isolate the inner square root term:

$$(a - x^2)^2 = a + x \implies a^2 - 2ax^2 + (x^2)^2 = a + x \implies x^4 - 2ax^2 - x + (a^2 - a) = 0$$

...
Wait, wait. Wait. That's an aha moment I can flag here.
Let's reevaluate this step-by-step to identify if the correct sum can be ...
We started with the equation:

$$\sqrt{a - \sqrt{a + x}} = x$$

First, let's square both sides:

$$a - \sqrt{a + x} = x^2 \implies \sqrt{a + x} = a - x^2$$

Next, I could square both sides again, treating the equation: ...
...

Table 2: An interesting "Aha moment" occurred in an intermediate version of DeepSeek-R1-Zero. The model learned to rethink using an anthropomorphic tone. This was an aha moment for us as well, allowing us to witness the power and beauty of reinforcement learning.

particularly when the prompts in the RL stage were multi-lingual. To mitigate this, a language consistency reward was introduced. Ablation studies showed that the language consistency reward caused a slight decrease in model performance but was more consistent with human preferences and easier to read. Therefore, the final reward signal was composed of the accuracy reward and the language consistency reward. (The paper does not mention the format reward, likely because the model from the cold start stage had already been fine-tuned on DeepSeek-R1-Zero's data and was thus assumed to meet the template format requirements).

6.7.3 Rejection Sampling & Supervised Fine-Tuning

When the RL stage approached convergence, checkpoints were used to collect the next round of SFT data. Unlike the cold start data, which focused mainly on reasoning, this stage incorporated data from other domains to enhance the model's general-purpose task capabilities. For reasoning data, rejection sampling was used: multiple responses were sampled from the checkpoints for each prompt, and only correct and highly readable responses were collected. For non-reasoning data, including writing, translation, self-awareness, and factual Q&A, data was still collected from DeepSeek-V3. The overall process is illustrated in Figure 9.

6.7.4 Secondary RL

This stage aims to further align with human preferences, improving the model's helpfulness and harmlessness, while refining its reasoning capabilities. For reasoning data, the same method as DeepSeek-R1-Zero was used, with rule-based rewards guiding the model's optimization. For non-reasoning data, a traditional reward model was built using DeepSeek-V3. For helpfulness, the focus was on the final summary to minimize interference from the underlying reasoning process. For harmlessness, the entire output was evaluated to identify and mitigate any potential risks, biases, or harmful content that might arise during generation.

6.7.5 Other Details

Distillation & RL Using data from DeepSeek-R1, several models were distilled via SFT. The distilled models surpassed the non-reasoning models in all aspects. The GRPO strategy was also applied to Qwen-32B-Base. The results showed that the performance of the resulting DeepSeek-R1-Zero-Qwen-32B was comparable to Qwen-32B-Preview but inferior to DeepSeek-R1-Distill-Qwen-32B.

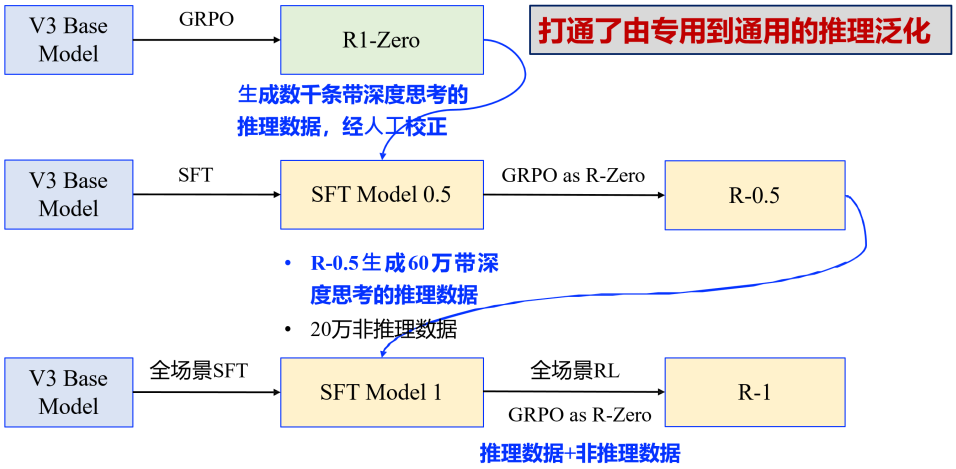

Figure 9: This figure, from a presentation by Prof. Jiajun Zhang of the Institute of Automation, Chinese Academy of Sciences, shows the practical workflow of the complete GRPO algorithm in DeepSeek-R1.

From this, it can be concluded that for complex abilities like reasoning, letting smaller models explore on their own may have its limitations. Learning the thought patterns of a more powerful model through distillation is a more effective and economical path. This suggests that top-tier reasoning abilities may require larger-scale models or higher-quality data to emerge.

7 Conclusion and Discussion

7.1 Review of the Technical Trajectory

The central narrative of this paper has been the evolution of Reinforcement Learning for Large Language Models (RL4LLM) along two distinct paths under the goal of alignment. The first path, pioneered by RLHF and refined by algorithms like DPO, is oriented towards aligning with subjective human preferences. The second path, represented by the GRPO algorithm, is oriented towards aligning with objective, fact-based reasoning capabilities.

Starting with the classic RLHF algorithm, we dissected its core algorithm, PPO, and its key component, the Generalized Advantage Estimator (GAE), analyzing both the power of online RL for preference alignment and its practical challenges. To address these challenges, the DPO algorithm, through an elegant mathematical transformation, bypassed the explicit modeling of a reward model and converted the complex RL optimization problem into a supervised learning-like problem, greatly simplifying the training paradigm. Subsequent algorithms like IPO further addressed DPO's potential overfitting issues by re-deriving the optimization objective, while other DPO-based works have proposed additional perspectives on preference alignment.

In the pursuit of enhancing model reasoning abilities, the GRPO algorithm was proposed as a successor to PPO, using intra-group relative ranking to replace the value model. This has provided a new approach for reasoning-intensive tasks.

7.2 Comparison of Core Paradigms

A comparison of the core paradigms of the main algorithms mentioned in this paper is shown in Table 3.

7.3 Open Questions

Although RL4LLM technology is developing rapidly, several open questions warrant further exploration:

Data Dependence Current preference alignment methods, whether PPO or DPO, are highly dependent on large-scale, high-quality human preference data. While techniques like RLAIF can alleviate this dependence, they do not fundamentally solve the problem of where preferences originate. Therefore, designing alignment paradigms that can self-guide and heuristically learn human values from more primitive, lower-cost data sources to achieve truly efficient unsupervised or weakly supervised alignment remains a key long-term challenge.

Dimension	PPO (RLHF)	DPO/IPO	GRPO
Learning Paradigm	Online	Offline	Online
Core Idea	Maximize expected reward	Maximize pairwise preference likelihood	Intra-group relative optimization
Model Requirements	4 models (policy, value, reward, ref)	2 models (policy, ref)	2 models (policy, ref) + Rule-based RM
Main Advantages	Strong exploration capability	Low cost, stable training	Suitable for reasoning-intensive tasks
Main Challenges	Training instability, hyperparameter sensitivity	Weak exploration, relies on fixed preference data	Relies on rule-based rewards, limited generality

Table 3: A comparison of the core paradigms of mainstream RL4LLM algorithms.

Paradigm Fusion DPO and its variants are known for their training stability, but this comes at the cost of sacrificing PPO's powerful online exploration capabilities. Therefore, designing a hybrid algorithm that can leverage DPO's stability for rapidly fitting the preference distribution in the early stages of training, while introducing PPO-style exploration mechanisms in later stages to discover new strategies beyond the existing data, will be crucial for achieving a higher ceiling of alignment.

Reward Exploitability Reward hacking is an inherent problem in RLHF. Even under GRPO's objective outcome supervision, a model might achieve a correct result through guessing or a flawed reasoning process, thereby deceiving the reward system. Thus, designing reward signals that can evaluate the quality of the **reasoning process**, not just the final outcome, without introducing costly human process supervision, is central to improving the reliability of model reasoning.

Generality across Scenarios DeepSeek's research demonstrated the effectiveness of using different RL strategies for reasoning versus general scenarios. This raises a deeper question: do we ultimately need a "Swiss Army knife"—a single, unified alignment objective that can handle all tasks—or a "toolbox" of specialized objectives tailored for different scenarios (e.g., creative writing, factual Q&A, code generation)? Exploring the boundaries of different alignment techniques and defining the most suitable RL paradigm for each scenario will be a critical step in RL4LLM research.

RL for Small Language Models (RL4SLM) While it seems that powerful reasoning pathways need to be discovered by larger-scale models, SFT-based distillation merely brings Small Language Models (SLMs) back to imitation learning. A more promising direction is to explore a new role for reinforcement learning in the collaboration between large and small models. For example, a powerful LLM could act as a reward model or an environment to train an SLM via RL to explore and learn the LLM's "thinking strategies," rather than just imitating its outputs. RL4SLM may be a crucial pathway to achieving low-cost, high-performance, specialized models.

References

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

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

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.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015a.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. Learning to summarize with human feedback. *Advances in neural information processing systems*, 33:3008–3021, 2020.

Richard S. Sutton. Learning to predict by the methods of temporal differences. *Mach. Learn.*, 3(1):9–44, August 1988. ISSN 0885-6125. doi:10.1023/A:1022633531479. URL https://doi.org/10.1023/A:1022633531479.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015b.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Remi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics*, pages 4447–4455. PMLR, 2024.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*, 2024.

Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. Contrastive preference optimization: Pushing the boundaries of llm performance in machine translation. *arXiv preprint arXiv:2401.08417*, 2024.

Shilong Li, Yancheng He, Hui Huang, Xingyuan Bu, Jiaheng Liu, Hangyu Guo, Weixun Wang, Jihao Gu, Wenbo Su, and Bo Zheng. 2d-dpo: Scaling direct preference optimization with 2-dimensional supervision. *arXiv preprint arXiv:2410.19720*, 2024.

Pulkit Pattnaik, Rishabh Maheshwary, Kelechi Ogueji, Vikas Yadav, and Sathwik Tejaswi Madhusudhan. Curry-dpo: Enhancing alignment using curriculum learning & ranked preferences. *arXiv preprint arXiv:2403.07230*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

John Schulman. Approximating kl divergence, 2020. URL http://joschu.net/blog/kl-approx.html.