A Tutorial on Reinforcement Learning for Large Language Models: RLHF and Beyond

Daniel R. Jiang

August 2026

Abstract

Large language models (LLMs) require *post-training*, which takes a foundation model and trains it to follow instructions, behave safely, and perform well across downstream use cases. *Reinforcement learning (RL) post-training* has become one of the most common post-training approaches, most prominently *RL from human feedback* (RLHF), where the reward reflects human preference judgments. In this tutorial, we formulate RL post-training as a sequential decision problem (a Markov decision process), covering both *single-turn* settings, where the model produces one response to a prompt, and *multi-turn* settings, where it interacts with a user or environment over multiple rounds. We also distinguish reward signals by how directly they can be measured or inferred (verifiable, observable, or latent). We then discuss reward overoptimization, a phenomenon in which the policy exploits errors in a learned reward model, motivating a Kullback-Leibler (KL)-regularized objective that keeps the policy close to a reference model while still optimizing reward. From this objective, we give precise step-by-step derivations of four central policy-optimization algorithms (DPO, REINFORCE, PPO, and GRPO), explicitly distinguishing exact derivations from practical approximations and heuristics. The tutorial also covers deployment paradigms, with a focus on *batch-online* deployment, in which a policy's interaction data is periodically collected and used for offline retraining and subsequent redeployment. For multi-turn settings, we cover an extension of GRPO to full trajectories collected from a training environment, and also present a batch-online method that performs approximate policy iteration from logged trajectories. Finally, we discuss open research directions in RL post-training that may benefit from a broad range of research perspectives.

Contents

1 Introduction

Modern large language model (LLM) training is often organized into two broad phases. In the first phase, called *pre-training*, a model learns general language patterns and world knowledge from large-scale unlabeled text data. The pre-training objective asks the model to predict the next token in naturally occurring text, and the result is a *foundation model* (Bommasani et al., 2021): a broad-capability model that can be adapted to many tasks, but one that does not yet reliably follow instructions, refuse unsafe requests, or perform well in downstream use cases. A subsequent phase of *post-training* adapts it accordingly (Ouyang et al., 2022; Lambert, 2026).

The most prominent family of post-training techniques trains the model's parameters with reinforcement learning (RL) to maximize a reward signal, which can come from various sources. We call this general approach *RL post-training*. More specifically, *RL from human feedback* (RLHF) derives the reward from human preference judgments, typically via a learned reward model (Christiano et al., 2017; Ouyang et al., 2022). *RL with verifiable rewards* (RLVR) instead uses a *verifier*, such as a unit test or an answer checker, to compute rewards directly from the model's responses (Lambert et al., 2024). Other applications train against observable performance metrics, such as click-through or conversion rates (Jiang et al., 2025). This tutorial provides a self-contained introduction to RL post-training for LLMs, spanning RLHF, RLVR, and related methods. We assume the reader has some basic familiarity with optimization, machine learning, and RL, but no prior background in natural language processing, LLMs, or LLM-specific RL algorithms.

RLHF was the first of these approaches to emerge, with origins that predate LLMs. Christiano et al. (2017) proposed training a reward model from pairwise human comparisons and then optimizing a policy against it using standard RL. Later, Ziegler et al. (2019) and Stiennon et al. (2020) applied this idea to text summarization, showing that optimizing for human preferences produced better summaries than supervised learning alone. The turning point was InstructGPT (Ouyang et al., 2022), which organized RLHF into three distinct steps: supervised fine-tuning (SFT), reward model training on preference comparisons, and policy optimization using the *Proximal Policy Optimization* (PPO) algorithm of Schulman et al. (2017). Applied to a 1.3B-parameter model, this pipeline produced outputs that human raters preferred over those of the much larger 175B-parameter GPT-3. It became the standard recipe and was the foundation of ChatGPT, released in late 2022 (OpenAI, 2022).

Although PPO was highly effective, its memory and compute demands motivated the community to develop simpler alternatives. Rafailov et al. (2023) introduced *Direct Preference Optimization* (DPO), a method that bypasses training a separate reward model and optimizes the policy directly from pairwise preferences. Shao et al. (2024) introduced *Group Relative Policy Optimization* (GRPO), which replaces PPO's learned value-function baseline with a group average over several sampled responses to the same prompt. GRPO was later used in the training of DeepSeek-R1-Zero and elicited strong reasoning behaviors without a supervised learning step (Guo et al., 2025).

RL post-training can be viewed through the broader lens of sequential decision making under uncertainty, connecting it to classical themes in reinforcement learning and approximate dynamic programming. Lambert (2026) provides a detailed and comprehensive treatment of RLHF and post-training. Relative to it and other surveys (Casper et al., 2023; Kaufmann et al., 2025; Zheng et al., 2023b; Wang et al., 2024), we emphasize five organizing themes:

1. *Consolidating multiple formulations into a unified Markov decision process (MDP) view.* Section 4 treats the single-turn (token-level and response-level) and multi-turn settings with common MDP notation, showing how moving the agent-environment boundary between the token- and response-level formulations changes the state space, action space, and transition model.

2. *Organizing types and sources of reward signals.* Section 5 discusses various types of reward signals and organizes them into *verifiable*, *observable*, and *latent* rewards, ordered by how directly they can be measured.

3. *Viewing reward models through a model-based RL lens.* Section 5 shows how we can view the learned reward models used in most RLHF pipelines through the lens of *model-based RL* (Sutton, 1991), which we believe is a useful perspective for future work because it delineates the true environment from a learned model used only for policy optimization.

4. *Deriving RL algorithms from a shared optimization objective.* Sections 6 and 7 introduce reward overoptimization and the Kullback–Leibler (KL)-regularized objective, and give step-by-step derivations of DPO, REINFORCE, PPO, and GRPO from this common objective, explicitly distinguishing exact derivations from practical approximations and heuristics.

5. *Formalizing batch-online deployment.* Section 8 formalizes the *batch-online* deployment pattern and connects it to classical batch-mode RL, and Section 9 exploits batch-online deployment

to give a principled multi-turn RL approach that requires no interactive training environment (Jiang et al., 2026).

2 Illustrative Examples

We now introduce five examples that span the settings where RL is applied to LLMs, and return to them throughout the tutorial. Table 1 summarizes the examples along several dimensions used later in the paper. The "Feedback type" column, formalized in Section 5, indicates whether the reward is verifiable, observable as a deployment outcome, or latent (requiring deliberate elicitation). The "Source of feedback" column specifies what supplies the reward, such as a human user, aggregate user behavior, a verifier, or a tool system. The "Reasoning traces" column indicates whether the model generates intermediate text before the final output, such as mathematical derivations, scratch work, or a "chain-of-thought" (Wei et al., 2022).

Example 1 (Creative writing assistant)**.** *Consider a user who asks an LLM to write a short story about a robot learning to paint. The model receives the writing prompt (the input x) and produces a creative response y (Lee et al., 2024b). The user may internally judge the quality of y, but no feedback is observed during normal use. To train a reward model that serves as a proxy for human judgment, practitioners elicit pairwise preferences by showing candidate response pairs to annotators and asking which is better (Ouyang et al., 2022).*

Example 2 (Generative ads)**.** *Consider a small business that wants to create ad text for a product but has little experience with online advertising. An LLM can generate improved variations of the business's original ad text (Jiang et al., 2025). Given a prompt x containing the original ad text and product metadata, the model produces a new version of the ad text y, whose quality is measured by an aggregate metric such as the click-through rate, computed from click outcomes recorded when the ad is shown to real users. Unlike the creative writing setting above, this reward is* observable*, and no deliberate elicitation step is needed, though deployment in a real environment is required.*

Example 3 (Math reasoning)**.** *Consider a student who asks an LLM to solve the equation $n^2 - 3n + 2 = 0$. The model receives the math problem as the prompt x and produces a response $y = (y_c, y_a)$, a chain-of-thought reasoning trace y_c followed by a final answer y_a. The reward is binary, taking value one if y_a matches a ground-truth answer and zero otherwise. Unlike the generative ads example, this reward can be easily computed offline (e.g., via simple string matching). Although this*

Table 1: Five illustrative examples spanning settings where RL can be applied to LLMs.

Example	Single-turn	Multi-turn	Feedback type	Source of feedback	Reasoning traces
Creative writing	✓		Latent	Human user	
Generative ads	✓		Observable	Many users	
Math reasoning	✓		Verifiable	Ground-truth answer	✓
Conversational agent		✓	Observable	Human user	
Coding agent		✓	Verifiable	Test suite	✓

is a single-turn problem, the policy must learn, from this sparse terminal reward alone, that careful intermediate reasoning leads to better answers (Guo et al., 2025; Shao et al., 2024).

Example 4 (Conversational agent)**.** *Consider a user who interacts with an LLM assistant over multiple turns in a goal-oriented conversation, such as finding the right product to buy (Jiang et al., 2026), resolving a technical issue (Brynjolfsson et al., 2025), or working through a problem with a tutor (Chung et al., 2026; Nam et al., 2025). Success is determined at the end of the conversation by an observable outcome, such as whether the customer completed a purchase or the support ticket was resolved (reward is one if so, zero otherwise). This creates a long-horizon credit-assignment problem, since each response matters through its downstream effect on a user's behavior.*

Example 5 (Coding agent with tool use)**.** *Consider a software engineer using an LLM coding agent to solve a coding task. The agent uses various "tools" (e.g., a code editor, terminal commands, or API calls) over multiple steps (FAIR CodeGen team et al., 2025). The reward is determined by running the final code through a suite of unit tests (reward is one if all tests pass, zero otherwise). Like math reasoning, this reward can be computed offline (e.g., in an execution sandbox), whereas conversational rewards often require observing outcomes in a deployed interaction.*

3 Background

Before introducing the RL algorithms used in post-training, we establish the foundational concepts needed to formalize the problem. This section covers autoregressive generation in LLMs, transformer properties relevant to RL, and the basics of Markov decision processes.

3.1 Large language models

LLMs generate text one *token* at a time. A token is an indivisible unit of text (a word, subword, or punctuation mark) drawn from a fixed vocabulary $\mathcal{V}$ with $|\mathcal{V}|$ typically in the tens or hundreds of thousands (Grattafiori et al., 2024). The mapping from text to tokens is determined by a *tokenizer*,

most commonly based on byte pair encoding (Sennrich et al., 2016).[1] For example, the sentence "Reinforcement learning is a useful tool." could be tokenized as

$$[\text{“Re”, “in”, “forcement”, “␣learning”, “␣is”, “␣a”, “␣useful”, “␣tool”, “.”}],$$

where ␣ denotes a space character that is part of the token and "Reinforcement" is split into three subword tokens. Given a prompt x (itself a sequence of tokens), the model generates a response $y = (y_1, y_2, \ldots, y_T)$ by repeatedly sampling the next token conditioned on the prompt and all tokens generated so far. We write $y_{<t} = (y_1, \ldots, y_{t-1})$ for the prefix before token t, with $y_{<1}$ empty. Generation stops when y_T is the special *end-of-sequence* token `<eos>` $\in \mathcal{V}$:

$$y_t \sim \pi_\theta(\cdot \mid x, y_{<t}), \qquad t = 1, 2, \ldots, T.$$

Here $\pi_\theta(\cdot \mid x, y_{<t})$ is a probability distribution over $\mathcal{V}$, parameterized by θ. This process, called *autoregressive generation*, produces each token conditioned on all previous ones, from left to right. The probability of a response y given prompt x decomposes as follows:

$$\pi_\theta(y \mid x) = \prod_{t=1}^{T} \pi_\theta(y_t \mid x, y_{<t}). \tag{1}$$

3.2 Brief overview of transformers

Since this tutorial focuses on RL, we present only a high-level summary of the *transformer* architecture, referring readers to Vaswani et al. (2017) for a full treatment. Given an input sequence of tokens, a transformer first looks up each token i in a learned *embedding table* (a matrix that maps every token to a vector) to obtain an initial hidden state $\mathbf{h}_i \in \mathbb{R}^d$, then repeatedly updates these states with *self-attention* layers.

In a self-attention layer, each token i forms a query vector, a key vector, and a value vector: $\mathbf{q}_i = \mathbf{W}_Q \mathbf{h}_i$, $\mathbf{k}_i = \mathbf{W}_K \mathbf{h}_i$, and $\mathbf{v}_i = \mathbf{W}_V \mathbf{h}_i$. The query can be interpreted as what token i is "looking for," the key as what it "offers" to other tokens, and the value as the "information copied" when it receives attention. Token i *attends* to token j by comparing its query $\mathbf{q}_i$ to token j's key $\mathbf{k}_j$ via a

[1] Subword methods like byte pair encoding keep common words as single tokens while splitting rare words into reusable pieces, balancing vocabulary size against sequence length and reducing out-of-vocabulary failures.

scaled dot product, a score measuring how relevant token j is to token i. These scores are normalized with a softmax to produce *attention weights* α_{ij}, and the output at token i is the resulting weighted sum of values, $\sum_j \alpha_{ij}\mathbf{v}_j$. Since generation is left to right, token i is only allowed to attend to earlier tokens $j \le i$, so the weights are computed as

$$\alpha_{ij} = \frac{\exp(\mathbf{q}_i^\top \mathbf{k}_j/\sqrt{d_k})}{\sum_{j' \le i} \exp(\mathbf{q}_i^\top \mathbf{k}_{j'}/\sqrt{d_k})}, \qquad j \le i,$$

where d_k is the dimension of the query and key vectors. We set $\alpha_{ij} = 0$ for $j > i$, so the sum $\sum_j \alpha_{ij}\mathbf{v}_j$ effectively runs only over $j \le i$. In practice, multiple attention heads operate in parallel on different projections, and their outputs are concatenated.

The model stacks multiple layers of self-attention and feedforward networks, and then applies a linear projection to vocabulary logits and a softmax to produce the next-token distribution $\pi_\theta(y_t \mid x, y_{<t})$. Three properties of this architecture are particularly relevant to RL post-training.

1. *Autoregressive policy.* A transformer implements the conditional distributions $\pi_\theta(y_t \mid x, y_{<t})$ in (1), mapping the current history $(x, y_{<t})$ to a distribution over the vocabulary $\mathcal{V}$. This makes it natural to cast LLM generation as sequential decision making, which we do in Section 4.

2. *Differentiable log-probabilities.* Although token samples are discrete, the log-probability assigned to any fixed token sequence is differentiable with respect to the model parameters, so gradients such as $\nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})$ can be computed by backpropagation.[2] This enables gradient-based RL for LLM policies.

3. *Efficient scoring of responses.* For a fixed response $y = (y_1, \ldots, y_T)$, all T terms of $\log \pi_\theta(y \mid x) = \sum_{t=1}^{T} \log \pi_\theta(y_t \mid x, y_{<t})$ come from a single forward pass, rather than T sequential ones. This is because self-attention scores every pair of positions at once, and a properly defined causal mask[3] "blocks" the pairs that look ahead, so each position's prediction is valid. As we will see in Sections 7.1 and 7.3, the DPO algorithm, reference-policy comparisons for the KL penalty, and PPO importance ratios all rely on this fixed-response scoring.

[2]The log-probability gradient $\nabla_\theta \log \pi_\theta(a \mid s)$ plays a central role in policy-gradient methods for RL (Williams, 1992). Differentiable log-probabilities are what make these methods applicable to LLMs.
[3]A causal mask sets $\alpha_{ij} = 0$ for every future position $j > i$ before the softmax, so position i's output depends only on $j \le i$.

3.3 Pre-training and supervised fine-tuning

In the *pre-training* stage (Radford et al., 2018; Brown et al., 2020), an LLM is trained on a large corpus of unlabeled text (trillions of tokens from the web, books, and code). Each document is treated as a single token sequence, and the objective is to predict every token from its predecessors. In *supervised fine-tuning* (SFT), usually the first stage of post-training, the model is further trained on a smaller, curated dataset $\mathcal{D}_{\text{SFT}} = \{(x_i, y_i)\}_{i=1}^{N}$, where y_i is the target response to prompt x_i. A supervised objective trains the model to generate responses similar to y_i when presented with prompts similar to x_i. Many conventional post-training pipelines begin with SFT on curated (prompt, response) pairs (Ouyang et al., 2022). While SFT teaches the model to imitate demonstrated responses, the subsequent RL stage allows the policy to explore candidate outputs and *optimize directly against feedback signals*. The rest of this tutorial focuses on this RL stage.

3.4 Reinforcement learning preliminaries

3.4.1 Markov decision processes

In reinforcement learning (RL), we study how an *agent* (a decision maker) learns to make sequential decisions by interacting with an *environment*. The standard formalization is a *Markov decision process* (MDP). As a preview, in the post-training setting of this tutorial, the agent is the LLM π_θ, and actions correspond to either individual tokens or complete responses. While we introduce MDPs here in general terms, Section 4 instantiates them for LLM generation.

Definition 1. *An MDP is a tuple $(\mathcal{S}, \mathcal{A}, P, P_0, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $P(s' \mid s, a)$ is the transition kernel (the distribution over next state s' given current state s and action a), P_0 is the initial state distribution, $r(s, a) = \mathbb{E}[R_t \mid s_t = s, a_t = a]$ is the expected immediate reward, where the realized reward R_t may be stochastic, and $\gamma \in [0, 1]$ is the discount factor, which downweights future rewards relative to immediate ones. We consider episodic rather than infinite-horizon MDPs; each episode starts from an initial state and ends upon reaching a terminal state.*

An agent's behavior in an MDP is specified by a stochastic *policy* $\pi(a \mid s)$, which maps states to probability distributions over actions. Given a policy π, the *value function* $V^\pi(s) = \mathbb{E}_\pi[\sum_{t=1}^{T} \gamma^{t-1} r(s_t, a_t) \mid s_1 = s]$ is the expected return from state s, where T is the (random) episode length. The *action-value function* $Q^\pi(s, a) = r(s, a) + \gamma\, \mathbb{E}_{s' \sim P(\cdot|s,a)}[V^\pi(s')]$ is the expected return from taking action a in state s and then following π. The *advantage function* $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$ measures how much better

action a is compared to the average action under π (and has zero mean under π). The goal is to find a policy that maximizes the expected return $J(\pi) = \mathbb{E}_{s_1 \sim P_0}[V^\pi(s_1)]$.

The gridworld of Example 6 is a common introductory MDP whose states, actions, transitions, and rewards can easily be visualized; see Figure 1. For further background on MDPs, see Puterman (1994), Sutton and Barto (2018), and Bertsekas (2019).

Example 6 (Gridworld MDP)**.** *The gridworld environment is a simple MDP. Suppose $\mathcal{S}$ is the set of grid cells, $\mathcal{A}$ is the four movement directions $\{\uparrow, \downarrow, \leftarrow, \rightarrow\}$, and P transitions the agent in the chosen direction unless blocked by a gray cell. P_0 places the agent in the blue starting cell. The green goal cell and red trap cell are terminal states. Entering either at step t ends the episode and yields reward $+1$ or -1, respectively, which contributes $\gamma^{t-1} \cdot (+1)$ or $\gamma^{t-1} \cdot (-1)$ to the discounted return.*

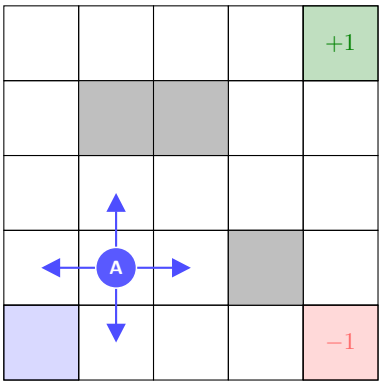

Figure 1: The gridworld MDP of Example 6.

3.4.2 The agent-environment interface

The *agent-environment interface* (Sutton and Barto, 2018) is the modeling choice of where to draw the boundary between agent and environment. It specifies what the agent chooses at each step (an action in $\mathcal{A}$) and what the environment returns before the agent acts again (the next state and reward). This choice determines $\mathcal{S}$, $\mathcal{A}$, P, and r, and therefore the resulting MDP itself, so the same real-world task can sometimes be modeled as different MDPs depending on where the boundary is drawn. In the gridworld of Example 6, a given sequence of movement directions always produces the same path. We can therefore model each direction as an action, with the environment returning the next cell and reward after each move, or the *complete path* as one action, with the environment returning the terminal cell and reward at the end. Both MDPs describe the same task but expose different decisions to the agent.[4]

[4]This equivalence relies on each movement direction having a deterministic outcome. If movement were stochastic, the same sequence of directions could produce different paths. The sequence-level formulation would prevent the agent from adapting later moves to earlier outcomes, making it a different decision problem.

4 Problem Formulation

We now specialize MDPs to the case of LLM post-training, beginning with the simpler single-turn setting (Section 4.1) and then extending to the multi-turn setting (Section 4.2).

4.1 The single-turn setting

In the single-turn setting, the LLM receives a prompt x and generates a single response y, after which a terminal reward R_T is observed. Consider the ad text generation example (Example 2) for a running shoe. The LLM might generate, token by token, "Discover," "the," "perfect," "running," "shoe." After the last token, the full ad is deployed and receives a reward based on user engagement. To give the reader some conceptual grounding (which will be useful to keep in mind throughout this section), we present a direct analogy to the gridworld environment of Example 6:

Just as a gridworld agent sequentially selects among four directions to reach the goal cell, an LLM sequentially selects among tens of thousands of tokens to reach a high-reward response.

This sequential token-by-token process defines an MDP.

Definition 2 (Single-turn, token-level MDP)**.** *Given a prompt x, the* token-level MDP *defines the state at step t as $s_t = (x, y_{<t})$, the prompt concatenated with all tokens generated so far, with initial state $s_1 = x$. The action at step t is the next token $y_t \in \mathcal{V}$. Transitions are deterministic, with $s_{t+1} = (s_t, y_t)$. The realized reward is $R_t = 0$ at all intermediate steps, with terminal reward R_T observed after the complete response $y = (y_1, \ldots, y_T)$, where $\mathbb{E}[R_T \mid x, y] = r(x, y)$. We typically set $\gamma = 1$. The policy is the LLM, $\pi_\theta(y_t \mid s_t) = \pi_\theta(y_t \mid x, y_{<t})$. The objective is to maximize the expected return:*

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\mathbb{E}_{y_t\sim\pi_\theta(\cdot|s_t)}\left[\sum_{t=1}^{T} r(s_t, y_t)\right]\right], \quad \text{where } r(s_t, y_t) = \begin{cases} 0 & t < T, \\ r(x, y) & t = T. \end{cases} \tag{2}$$

The environment *consists of the sparse reward R_T and the (known) token-concatenation dynamics. Even though intermediate tokens receive no reward, they influence future tokens, which in turn influence the terminal reward. For example, in math reasoning (Example 3), the model must learn which reasoning traces tend to lead to correct answers (Guo et al., 2025). See Figure 2 for a visualization.*

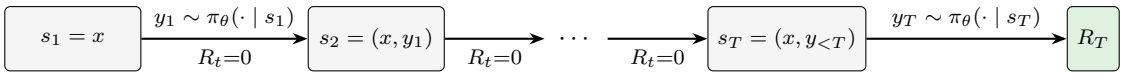

Figure 2: The token-level MDP. Each state is the prompt plus tokens generated so far. Actions are individual tokens from the vocabulary $\mathcal{V}$. Transitions are deterministic (concatenation). Realized rewards are zero at intermediate steps, and R_T is observed at the terminal step.

Just as the gridworld can treat a *complete path* rather than each move as an action, an alternative LLM formulation treats a *complete response* rather than each token as an action (Ahmadian et al., 2024).

Definition 3 (Single-turn, response-level MDP)**.** *Alternatively, the* response-level *(or* bandit*) formulation takes the state to be $s = x$ (the prompt), the action to be the entire response $y \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of finite token sequences ending in* `<eos>`*, and the reward to be $r(x, y)$. This is a contextual bandit, or a one-step MDP with no sequential structure. The* environment *is just the reward function r, with no transition function P to specify since the episode ends after a single action. The objective is:*

$$J(\theta) = \mathbb{E}_{x\sim\mathcal{D}}\big[\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[r(x, y)\big]\big]\,. \tag{3}$$

Remark 4 (Agent-environment interface in the two formulations)**.** *Definitions 2 and 3 model the same task with the agent-environment boundary drawn at different granularities. Because partial responses are not valid states in the response-level formulation, the two MDPs have different state spaces in addition to different action spaces. As we will see in Section 7, algorithms that assign credit to individual tokens along the way rely on the token-level MDP's intermediate states, whereas algorithms that only ever compare or score complete responses fit naturally into the response-level formulation, with no need for those intermediate states.*

Remark 5 (Equivalence of the two single-turn formulations)**.** *The token-level objective (2) and the response-level objective (3) assign the same value to any fixed policy in the single-turn setting. Token-level generation samples $y_1, \ldots, y_T$, and the environment only concatenates these tokens into a response y, so both formulations induce the same distribution over completed responses and thus the same distribution over terminal rewards (Ahmadian et al., 2024).*

4.2 The multi-turn setting

Many practical applications involve multi-turn interactions. Consider the sales agent from Example 4. The customer asks about a product, the LLM responds, the customer follows up, and this continues until the customer finds the right product or leaves. The same framework applies to LLM agents that interact with external tools (executing code, calling APIs, browsing the web), where the environment responses are determined by the tool outputs rather than a human user. Unlike the single-turn case, where the LLM controls the entire generation process, here the *environment dynamics are external to the LLM and may be stochastic or unknown.*

The natural formulation is at the response level. Each action is a complete response, and the environment (e.g., a user) responds with the next message. We write the response at turn h as the action $a_h = (y_{h,1}, \ldots, y_{h,T_h})$, where T_h is the (random) number of tokens in the response; see Figure 3.

Definition 6 (Multi-turn, response-level MDP)**.** *In a multi-turn MDP, the state at turn h is the full conversation history $s_h = (x_1, a_1, x_2, a_2, \ldots, x_h)$, where x_i is the environment's message at turn i and a_i is the agent's response.*[5] *The initial state is $s_1 = x_1$. The action at turn h is the agent's complete response a_h. After the agent sends a_h, the environment produces $x_{h+1} \sim P(\cdot \mid s_h, a_h)$, where P is generally unknown. The realized rewards may be per-turn or sparse. In goal-oriented settings they are typically sparse, with $R_h = 0$ for $h < H$ and terminal reward R_H, though dense per-turn rewards are also possible (e.g., a small reward for each successful tool call). The objective is to maximize the expected discounted return:*

$$J(\theta) = \mathbb{E}_{x_1 \sim \mathcal{D}} \left[\mathbb{E}_{\substack{a_h \sim \pi_\theta(\cdot \mid s_h) \\ x_{h+1} \sim P(\cdot \mid s_h, a_h)}} \left[\sum_{h=1}^{H} \gamma^{h-1} \, r(s_h, a_h) \right] \right],$$

where $\gamma \in [0, 1]$ is the discount factor and H is the (random) number of turns. Comparing with the single-turn objectives (2) and (3), here the sum is over turns *(complete responses) rather than tokens, and the expectation includes the stochastic environment transitions P.*

[5] For simplicity, we treat the complete conversation history as a fully observed Markov state. In practice, the underlying environment may be partially observable. In the sales-agent setting (Example 4), the customer's intent remains latent, whereas in a coding agent (Example 5), relevant parts of the codebase may lie outside the context window (Kausik et al., 2026). These settings are naturally modeled as a *partially observed MDP* (POMDP), which we leave outside the scope of this tutorial.

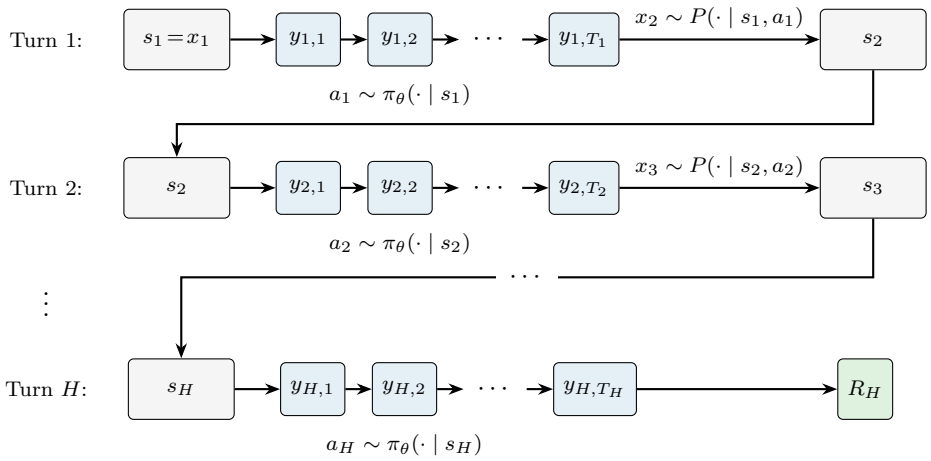

Figure 3: The multi-turn response-level MDP. Each turn, the agent (LLM π_θ) generates a complete response $a_h = (y_{h,1}, \ldots, y_{h,T_h})$ token by token, then the environment produces the next user message via the transition P. The terminal reward R_H is received at turn H.

5 Feedback and Reward Modeling

Before optimizing a policy, we need to understand where the reward signal comes from. Some rewards are immediately computable, some depend on real users' behaviors, and many important rewards are *latent*, meaning no reward is revealed during normal system operation and must be inferred from deliberate elicitation. Table 1 from Section 2 captures this variation in the "Feedback type" column.

5.1 Verifiable rewards

A *verifiable reward* is computed exactly by a known function, as in the RLVR setting introduced in Section 1 (Lambert et al., 2024). In math reasoning (Example 3), the LLM's answer is compared to a ground-truth answer, producing a binary reward. For coding agents (Example 5), running the test suite also produces a binary reward. This verifiability was central to a pivotal moment for RL in LLM reasoning, when DeepSeek-R1-Zero showed that RL with rule-based verifiable rewards could produce reasoning behavior without supervised reasoning traces (Shao et al., 2024; Guo et al., 2025). One caveat to keep in mind is that verifiability does not necessarily mean the specified criterion fully captures the intended task or cannot be exploited (e.g., a coding agent may pass unit tests by hard-coding the test cases).

Remark 7 (Connection to planning and approximate dynamic programming)**.** *In the single-turn setting with a verifiable reward, the token-concatenation dynamics are known and the reward can be computed exactly. This makes the problem closer to planning or* approximate dynamic programming *(Bertsekas and Tsitsiklis, 1996; Powell, 2011; Munos, 2003; Scherrer et al., 2015; Bertsekas, 2019),*

though policy optimization remains computationally difficult because the response space is enormous and the policy is optimized from sampled model outputs. We mention this because classical RL usually emphasizes unknown environment dynamics (Sutton and Barto, 2018), but this line has blurred in modern practice, where the same methods are called "RL" whether or not the dynamics are known.

5.2 Observable rewards

In many practical settings the reward can only be obtained by *observing an outcome in the environment*. This is the setting closest to the classical RL setup (Sutton and Barto, 2018). *Observable rewards* arise as a byproduct of normal system operation and no deliberate elicitation step is needed (see the next section for a discussion of *latent rewards*). In the generative ads setting (Example 2), after deploying an ad, it receives an aggregate click-through rate based on user engagement (Jiang et al., 2025). In the conversational agent setting (Example 4), whether a customer makes a purchase ($R_H = 1$) or not ($R_H = 0$) is a directly observable outcome, but only after the full conversation has played out with a real user (Jiang et al., 2026). More broadly, any LLM system with a measurable downstream outcome (student test scores for a tutoring agent, ticket resolution rates for a support bot, or user retention for a chat assistant) produces an observable reward.

5.3 Latent rewards, preferences, and reward modeling

When the reward is *latent*, no reward is revealed during normal system operation, so it must be *inferred from a deliberate elicitation step*. In Example 1 on creative writing, a reader may find a story engaging or dull, but this judgment is never recorded unless we explicitly ask for it. The same applies to general-purpose chat assistants, summarization, and instruction following, where quality is a subjective preference that does not surface as a measurable outcome during ordinary use.

One approach to elicit the underlying reward is to ask annotators for absolute scores, but assigning a consistent numeric score to a subjective quality is difficult, and the resulting scores are often noisy and hard to calibrate across annotators. A more reliable method uses *pairwise preference comparisons*. An annotator is shown two responses $y^{(a)}$ and $y^{(b)}$ to the same prompt x and asked which is better. The foundational work applying this idea to deep RL is Christiano et al. (2017), who demonstrated that human preferences could be used to train RL agents without access to a reward function.[6] An

[6]Preference-based feedback also has a long history in the Bayesian learning and bandits literatures (Chu and Ghahramani, 2005; Yue et al., 2012).

important practical point is that preference elicitation is not the same as collecting user feedback from normal product use (although some products do solicit comparisons directly from users, such as a chatbot occasionally asking which of two responses is better, but this remains the exception). Most preference data is instead collected through a controlled process involving trained *human labelers*, often given detailed rubrics (Stiennon et al., 2020; Ouyang et al., 2022).

To recover the latent reward from pairwise comparisons, we need a structural assumption relating rewards to observed preferences. The standard is the *Bradley-Terry model* (Bradley and Terry, 1952):[7]

$$P(y^{(a)} \succ y^{(b)} \mid x) = \sigma\big(r(x, y^{(a)}) - r(x, y^{(b)})\big), \tag{4}$$

where $\sigma(z) = 1/(1 + e^{-z})$ is the sigmoid function. Here, the preference probabilities are determined by differences in an underlying scalar (expected) reward. Given a preference dataset $\mathcal{D}_{\text{pref}} = \{(x, y_w, y_l)\}$ where $y_w \succ y_l$ (this denotes that y_w is preferred to y_l), we can fit a parametric *reward model* $r_\phi(x, y)$, typically a pre-trained LLM with a scalar output head, by maximum likelihood under the Bradley-Terry assumption (Christiano et al., 2017; Ouyang et al., 2022):

$$\mathcal{L}_{\text{RM}}(\phi) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_{\text{pref}}}\Big[\log \sigma\big(r_\phi(x, y_w) - r_\phi(x, y_l)\big)\Big].$$

The trained r_ϕ serves as a proxy for the latent true reward in downstream RL optimization.

In addition, we note that reward models are not limited to the latent-reward case. In settings with observable rewards, one can fit r_ϕ directly to scalar outcome data via standard regression (as opposed to using a preference dataset): for example, in the case of Example 2, one can train a reward model to predict click-through rate from (x, y) pairs using historical ad performance data (Jiang et al., 2025). The resulting r_ϕ then serves as a fast, deployable proxy for a reward that would otherwise require real-world interaction.

5.4 RL from AI feedback (RLAIF) and LLM-as-a-judge

Human preference labels are expensive to collect. As an alternative, *RL from AI feedback* (RLAIF) replaces some or all of them with judgments from an LLM (Lee et al., 2024a). The model performing

[7] Other structural assumptions beyond Bradley-Terry also exist; for example, *ranking models* (e.g., Plackett-Luce) extend it to full orderings over $k \geq 3$ responses, allowing for more information per annotation.

the evaluation is often called an *LLM-as-a-judge*, or simply an *LLM judge* (Zheng et al., 2023a). In an early example, Bai et al. (2022) use a model to critique and revise responses according to written principles (the "constitution"), and then use an LLM to compare response pairs to generate preferences for reward-model training. Other examples include DeepSeek-V3, which used the model's own judgments as feedback on subjective prompts (DeepSeek-AI, 2024); Kimi K2, which used a model to rank responses against written criteria (Kimi Team et al., 2025); MAI-Thinking-1, which combined LLM judgments with human feedback and automatic checks during RL (The Microsoft AI Team, 2026); and Tulu 3, which used GPT-4o to choose the better response in synthetic pairs (Lambert et al., 2024). However, LLM judges can introduce systematic errors into training, including *position bias* (favoring the first response) or *verbosity bias* (favoring overly long responses) (Zheng et al., 2023a), so their judgments should be validated carefully before they are used as training labels.

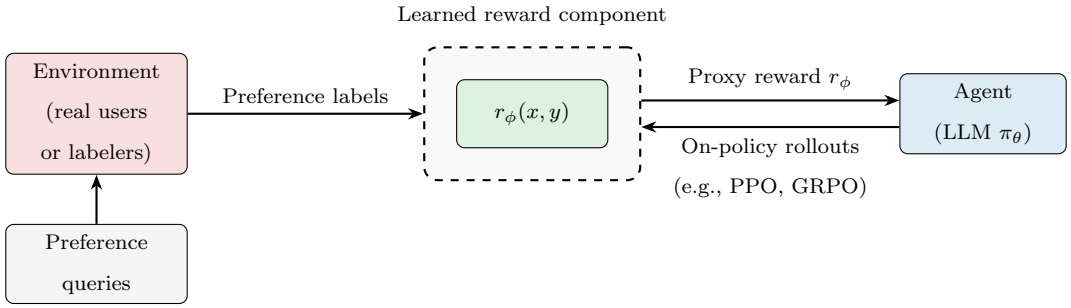

Figure 4: Reward modeling viewed through a model-based RL lens. Preference queries are sent to users or labelers to elicit preference labels, which are used to fit a reward proxy r_ϕ, the learned reward component of an environment model. During post-training, the policy generates on-policy rollouts that are scored by r_ϕ rather than by real users.

5.5 Viewing reward modeling through a model-based RL lens

RL algorithms are often divided into *model-free* methods, which learn a policy or value function directly from interaction, and *model-based* methods, which first learn a model of the environment and then use it for policy optimization (Sutton and Barto, 2018). Depending on which parts of the environment are known, model-based RL methods may learn only rewards, only transition dynamics (Chua et al., 2018; Janner et al., 2019), or rewards and dynamics jointly (Sutton, 1991; Hafner et al., 2019, 2020; Schrittwieser et al., 2020). Single-turn RLHF falls into the first case because its transition dynamics are known while its rewards are not. As shown in Figure 4, the reward model r_ϕ approximates the unknown reward function and, together with the known transition dynamics, forms a complete model of the environment that supports policy optimization with on-policy rollouts.

We believe that viewing reward modeling as model-based RL is an important conceptual perspective because it clearly delineates the true environment from the learned reward model that is only used as a proxy during policy optimization. This distinction becomes especially useful across repeated deployments of an LLM because each round of newly collected data can be used to refit the reward model and retrain the LLM, as discussed in Section 8.2. In the broader RL setting, other authors have recently argued for a similar distinction between *solving a simulator* and *using a simulator as a proxy* (Vandergrift et al., 2026).

6 The KL-Regularized Objective and Reward Overoptimization

This section discusses reward overoptimization (Section 6.1), which motivates the KL-regularized objective (Section 6.2) underlying the RL algorithms that we will introduce in Section 7.

6.1 Reward overoptimization

As we have emphasized in the previous section, the learned reward r_ϕ, trained from a finite set of human comparisons, is only a *proxy* for the true expected reward r, so it is most reliable on responses similar to those in its training data. As the policy changes, it may produce responses on which r_ϕ is less accurate, and optimization over many possible responses can find and amplify even small errors by favoring responses whose quality is overestimated by r_ϕ. *Reward overoptimization* occurs when this process keeps raising r_ϕ after r has flattened or declined (Gao et al., 2023; Coste et al., 2024; Moskovitz et al., 2024). For example, if better answers in the training data tend to be longer, r_ϕ may learn to reward verbose responses, so continued optimization can produce longer but incorrect answers that raise r_ϕ while lowering r.

Gao et al. (2023) measure overoptimization by using a large "gold" reward model as a substitute for human judgment, training smaller proxy models from its labels, and optimizing the proxies through best-of-n sampling[8] or RL. Under both methods of optimization, the gold reward first rises and then falls while the proxy reward keeps increasing, as illustrated in Figure 5(b). Larger proxy models and more reward-model data tend to reduce this discrepancy.

In practice, an independent quality measure on held-out prompts, such as human evaluation or a separate reward model, reveals overoptimization when proxy reward continues to rise after the measure

[8] *Best-of-n* sampling generates n candidate responses for a prompt and returns the one with the highest reward-model score. This is a simple way to optimize the reward without changing the policy weights.

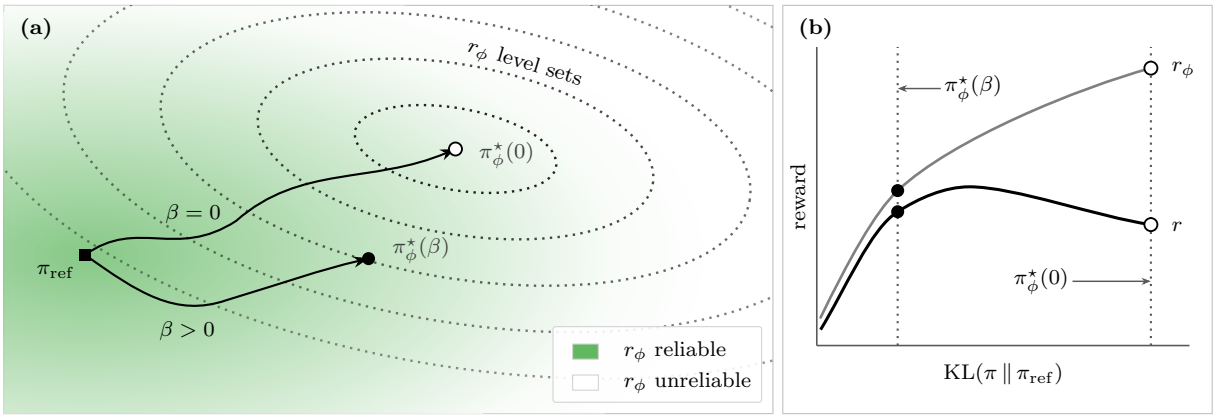

Figure 5: Reward overoptimization and KL regularization. For each $\beta \geq 0$, let $\pi^\star_\phi(\beta) \in \arg\max_\pi J_{\text{KL},\phi}(\pi)$, with $J_{\text{KL},\phi}$ evaluated at KL coefficient β in (5). Panel (a) compares the unregularized policy $\pi^\star_\phi(0)$ with $\pi^\star_\phi(\beta)$ for $\beta > 0$. Filled circles denote $\pi^\star_\phi(\beta)$, open circles denote $\pi^\star_\phi(0)$, and the square marks π_{ref}. Panel (b) shows the overoptimization pattern reported by Gao et al. (2023), where the proxy reward r_ϕ (gray) continues to increase while the true reward r (black) eventually decreases, so the unregularized policy attains higher proxy reward but lower true reward.

flattens or declines. Simple statistics such as response length help identify what the policy is exploiting, while reward-model ensembles and constrained optimization reduce overoptimization directly (Coste et al., 2024; Eisenstein et al., 2024; Moskovitz et al., 2024). These methods are beyond the scope of this tutorial, so we focus on the standard KL-regularized objective, which penalizes deviations from the reference model.

6.2 The KL-regularized objective

We denote the fixed reference model by π_{ref} and call it the *reference policy*, which is typically the SFT model. The distance between π_θ and π_{ref} is measured by the *Kullback–Leibler (KL) divergence*, a standard measure of how different two probability distributions are. Depending on how the reward is obtained, policy optimization maximizes one of two KL-regularized objectives:[9]

$$\begin{aligned} J_{\text{KL}}(\pi) &= \mathbb{E}_{x\sim\mathcal{D}}\Big[\mathbb{E}_{y\sim\pi(\cdot|x)}\big[r(x,y)\big] - \beta\,\text{KL}\big(\pi(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\big)\Big]\,, \\ J_{\text{KL},\phi}(\pi) &= \mathbb{E}_{x\sim\mathcal{D}}\Big[\mathbb{E}_{y\sim\pi(\cdot|x)}\big[r_\phi(x,y)\big] - \beta\,\text{KL}\big(\pi(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\big)\Big]\,. \end{aligned} \tag{5}$$

The objective J_{KL} uses the environment reward r, whether it is latent, observed, or computed by a verifier, while $J_{\text{KL},\phi}$ uses the learned proxy r_ϕ. The coefficient β controls the strength of the KL penalty,

[9]The "KL" in (5) is the exact divergence $\text{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)) = \mathbb{E}_{y\sim\pi_\theta}[\log(\pi_\theta(y \mid x)/\pi_{\text{ref}}(y \mid x))]$, which averages over all responses. Implementations approximate it from sampled responses, commonly using the log-ratio $\log(\pi_\theta(y \mid x)/\pi_{\text{ref}}(y \mid x))$ or Schulman's nonnegative k_3 estimator $\pi_{\text{ref}}(y \mid x)/\pi_\theta(y \mid x) - \log\big(\pi_{\text{ref}}(y \mid x)/\pi_\theta(y \mid x)\big) - 1$. The KL direction, estimator, and normalization can each affect training behavior (Schulman, 2020; Tang and Munos, 2025).

which keeps the policy near π_{ref}, where the reward model is most reliable (Ziegler et al., 2019; Stiennon et al., 2020; Ouyang et al., 2022). A smaller β permits more optimization but gives the policy more opportunity to exploit reward-model errors, while a larger β reduces this risk but can limit improvement. The policy π_{ref} is fixed in both objectives, and the reward-model parameter vector ϕ is also fixed while optimizing $J_{\text{KL},\phi}$. Evaluating the objectives at π_θ, setting $\beta = 0$ in $J_{\text{KL}}(\pi_\theta)$ recovers the response-level objective in (3), while setting $\beta = 0$ in $J_{\text{KL},\phi}(\pi_\theta)$ means we simply optimize expected proxy reward.

Figure 5 connects this tradeoff to reward overoptimization. In Panel (a), the green shading marks the region where r_ϕ is most reliable, and the dotted contours are level sets of r_ϕ, with higher proxy reward toward their center. Starting from π_{ref}, optimization with $\beta = 0$ reaches the open marker $\pi^\star_\phi(0)$ in a less reliable region, whereas $\beta > 0$ ends at the filled marker $\pi^\star_\phi(\beta)$ closer to π_{ref}. Panel (b) plots reward against the KL divergence from π_{ref}: the gray curve is the proxy reward r_ϕ, the black curve is the true reward r, and the vertical dotted lines locate the two policies. The figure shows that moving from $\pi^\star_\phi(\beta)$ to $\pi^\star_\phi(0)$ (removing KL regularization) raises the proxy reward but *lowers* the true reward, illustrating how KL regularization is expected to help limit overoptimization.

7 Single-Turn Policy Optimization

The algorithms in this section target either J_{KL} or $J_{\text{KL},\phi}$ in (5), two versions of the same KL-regularized objective that differ only in whether the reward is r or r_ϕ. DPO derives the optimizer of J_{KL} directly from preferences, while we present REINFORCE, PPO, and GRPO using $J_{\text{KL},\phi}$. In the verifiable setting where the environment reward r is directly available, the only required change is to replace $J_{\text{KL},\phi}$ with J_{KL}. Table 2 gives a preview and summary of the four algorithms we cover.

Table 2: Comparison of four single-turn optimization algorithms. "MDP formulation" describes the level at which the update is written. GRPO is listed as "mixed" because it computes response-level rewards and a group baseline, then applies the resulting advantage through token-level likelihood ratios. "Reward signal" distinguishes pairwise preferences from scalar rewards (which may come from a learned reward model, verifier, or observed metric). "Baseline" identifies the quantity used to center rewards or returns in policy-gradient methods. "Multiple updates" means that the same rollout batch is used for several optimization steps via an off-policy correction ratio.

Method	MDP formulation	On-policy rollouts	Reward signal	Baseline	Multiple updates
DPO	Response level	No	Preferences	N/A	N/A
REINFORCE	Response level	Yes	Scalar reward	Optional scalar	No
PPO	Token level	Yes	Scalar reward	Value model	Yes
GRPO	Mixed	Yes	Scalar reward	Group average	Yes

7.1 DPO

DPO (Rafailov et al., 2023) was introduced after PPO-based RLHF (Ouyang et al., 2022), but we present it first because it follows most directly from the KL-regularized objective. DPO avoids training a separate reward model by optimizing the policy *directly* on an offline preference dataset $\mathcal{D}_{\text{pref}}$ of triples (x, y_w, y_l), where y_w is preferred to y_l for prompt x (Section 5.3). This matches the creative-writing setting of Example 1, where a human annotator can often compare two stories written for the same prompt even when assigning stable numeric scores would be difficult.

Following Rafailov et al. (2023), we derive DPO from J_{KL} in (5) using three objects: the latent reward r that determines preferences, an optimal policy $\pi^* \in \arg\max_\pi J_{\text{KL}}(\pi)$, and the parameterized policy π_θ that we train. DPO first solves for π^* and then uses that solution to replace r with policy log-probabilities, so the final loss depends only on observed preferences, π_θ, and π_{ref}. First, for $\beta > 0$, the closed-form solution is

$$\pi^*(y \mid x) = \frac{1}{Z(x)}\, \pi_{\text{ref}}(y \mid x)\, \exp\Big(\frac{r(x,y)}{\beta}\Big), \tag{6}$$

where $Z(x) = \sum_{y'} \pi_{\text{ref}}(y' \mid x) \exp(r(x,y')/\beta)$ is a normalizing partition function. The optimal policy π^* "tilts" π_{ref} toward high-reward responses, with β controlling the temperature. Second, we can invert (6) to express r as a function of π^*. Rearranging and taking the log gives

$$r(x,y) = \beta \log \frac{\pi^*(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x). \tag{7}$$

This identity applies to the policy that is optimal for the reward under consideration, saying that if we knew π^*, we could recover r up to the prompt-dependent constant $\beta \log Z(x)$. Third, we can substitute (7) into the Bradley-Terry model (4). The model depends on r only through the difference $r(x, y_w) - r(x, y_l)$, so $\beta \log Z(x)$ cancels:

$$P(y_w \succ y_l \mid x) = \sigma\left(\beta \log \frac{\pi^*(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi^*(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right).$$

This expresses the preference probability entirely in terms of π^*, with no explicit reward function. Of course, π^*, being our target, is unknown. The final step replaces π^* with the parameterized policy π_θ. For each pair (x, y_w, y_l), the sigmoid below is the probability $P_\theta(y_w \succ y_l \mid x)$ implied by π_θ for the

observed preference. Maximum likelihood chooses θ to make the observed preferences collectively as probable as possible, or equivalently, to minimize their average negative log-probability:

$$\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}_{\text{pref}}}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta\log\frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]. \tag{8}$$

The log-ratio $r_\theta(x, y) = \beta\log\big(\pi_\theta(y \mid x)/\pi_{\text{ref}}(y \mid x)\big)$ acts as DPO's *implicit reward*, although DPO does not train a separate reward model. Under a correct Bradley-Terry model (i.e., observed preferences satisfy the Bradley-Terry assumption), sufficient coverage, and an expressive policy, minimizing the DPO loss recovers the reward differences needed for the KL-regularized optimal policy. This means the implicit reward is identified only up to a prompt-dependent constant, which importantly does not change the preference probabilities or the optimal policy (Rafailov et al., 2023, Sec. 5).

7.1.1 Practical limitations and extensions of DPO

DPO differs from the other methods that we will cover (REINFORCE, PPO, GRPO) in that it is fully "offline" in the sense that it does not sample responses from the current policy during training. In general, offline alignment methods have been reported to underperform methods that use on-policy samples (Tang et al., 2024; Ivison et al., 2024; Song et al., 2024).

One intuition for DPO is that it should increase the likelihood of preferred responses while decreasing the likelihood of rejected ones, but Pal et al. (2024) show that the DPO loss can actually *decrease* the absolute likelihood of preferred examples while improving their likelihood relative to rejected responses. Razin et al. (2024) characterize a related phenomenon, *likelihood displacement*, in which probability mass shifts away from preferred responses toward unintended ones, and Asadi et al. (2025) propose constraints that limit this displacement.

DPO can also amplify small length imbalances in preference data, producing responses longer than those in the training pairs (Park et al., 2024). DPO also sums token-level log-ratios over each response, so longer responses contribute more terms to the implicit reward (Lu et al., 2024). Park et al. (2024) add a length-difference term to the DPO logit, while Lu et al. (2024) compute both response log-ratios using the same number of token-level terms, uniformly subsampling the longer response. Simple Preference Optimization (SimPO) addresses this bias by replacing summed sequence log-probability with average log-probability plus a target margin (Meng et al., 2024).

Other DPO variants change the loss or feedback type. Identity Policy Optimization (IPO) replaces DPO's logistic loss with a squared objective (Gheshlaghi Azar et al., 2024). Kahneman-Tversky Optimization (KTO) instead optimizes a prospect-theoretic utility relative to a reference point using binary desirable/undesirable labels on individual responses, avoiding the need for matched preference pairs while remaining competitive with pairwise methods (Ethayarajh et al., 2024).

Takeaway. DPO is a response-level method that trains directly on preference pairs rather than scalar rewards or on-policy rollouts. It is simple and stable because it requires neither a separate reward model nor a rollout loop during optimization. However, several studies find that offline alignment methods can underperform methods trained with on-policy rollouts (Tang et al., 2024; Ivison et al., 2024; Song et al., 2024). Extensions of vanilla DPO address likelihood displacement and length biases.

7.2 REINFORCE

We now turn to policy-gradient methods, the simplest of which is *REINFORCE* (Williams, 1992). Following Ahmadian et al. (2024), we use the response-level form, which treats the complete response y as one action in the bandit formulation of Definition 3. Recall the learned-reward objective $J_{\mathrm{KL},\phi}$ from (5). Previous work has often folded the KL penalty directly into the RL reward. Notice that if we define the sampled log-ratio $\ell_\theta(x,y) = \log(\pi_\theta(y \mid x)/\pi_{\mathrm{ref}}(y \mid x))$, then since $\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[\ell_\theta(x,y)] = \mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x))$, we have

$$J_{\mathrm{KL},\phi}(\pi_\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\big[\underbrace{r_\phi(x,y) - \beta\ell_\theta(x,y)}_{\tilde{r}_{\mathrm{KL},\theta}(x,y)}\big]. \tag{9}$$

Other authors have referred to the quantity inside the expectation as the *KL-shaped reward* $\tilde{r}_{\mathrm{KL},\theta}(x,y)$ (Ouyang et al., 2022; Ahmadian et al., 2024). The main idea is that existing RL algorithms (e.g., REINFORCE) can be applied directly to this "redefined" reward.

Remark 8 (Issue with policy-dependent reward)**.** *Although defining $\tilde{r}_{\mathrm{KL},\theta}(x,y)$ is convenient, it should not be viewed conceptually as an environment reward in the standard RL sense. For a fixed prompt-response pair (x,y), the "reward" $\tilde{r}_{\mathrm{KL},\theta}(x,y)$ changes with the policy parameters θ, whereas standard RL formulations do not allow the environment reward to depend on the policy parameters.*

Because ℓ_θ depends on θ, the exact gradient must account for both the changing sampling probabilities and the explicit derivative of ℓ_θ. We therefore cannot apply the standard policy-gradient theorem by treating the KL-shaped reward as fixed. Alternatively, we can explicitly freeze a rollout snapshot θ^-, define the KL-shaped return using π_{θ^-}, and optimize the resulting first-order local surrogate. We will discuss these approaches in the following subsections.

To our knowledge, prior work on LLM post-training has not explicitly discussed this conceptual issue. However, we note that GRPO (Section 7.4) avoids the conceptual issue entirely by placing the KL term outside the MDP as an additional loss penalty. Separate but related work studies how alternative KL formulations and estimators affect RL post-training (Liu et al., 2025a; Tang and Munos, 2025).

7.2.1 Exact gradient of the KL-regularized objective

The conceptual discussion in Remark 8 does not prevent us from computing the gradient of (9) directly.

$$
\begin{aligned}
&\nabla_\theta J_{\mathrm{KL},\phi}(\pi_\theta) \\
&= \nabla_\theta \, \mathbb{E}_{x\sim\mathcal{D}}\big[\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[r_\phi(x,y) - \beta\ell_\theta(x,y)]\big] \\
&= \mathbb{E}_{x\sim\mathcal{D}}\left[\nabla_\theta \sum_y \pi_\theta(y \mid x)\big(r_\phi(x,y) - \beta\ell_\theta(x,y)\big)\right] \\
&= \mathbb{E}_{x\sim\mathcal{D}}\left[\sum_y \nabla_\theta\pi_\theta(y \mid x)\big(r_\phi(x,y) - \beta\ell_\theta(x,y)\big) - \beta\sum_y \pi_\theta(y \mid x)\nabla_\theta\ell_\theta(x,y)\right] \\
&= \mathbb{E}_{x\sim\mathcal{D}}\big[\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[\big(r_\phi(x,y) - \beta\ell_\theta(x,y)\big)\nabla_\theta \log \pi_\theta(y \mid x)\big] - \beta\,\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[\nabla_\theta\ell_\theta(x,y)]\big] \\
&= \mathbb{E}_{x\sim\mathcal{D}}\big[\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}\big[\big(r_\phi(x,y) - \beta\ell_\theta(x,y)\big)\nabla_\theta \log \pi_\theta(y \mid x)\big] - \beta\,\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[\nabla_\theta \log \pi_\theta(y \mid x)]\big] .
\end{aligned}
$$

where the third equality is the product rule, the fourth equality uses the log-derivative trick[10] on the sampling probabilities, and the last equality uses $\nabla_\theta\ell_\theta(x,y) = \nabla_\theta \log \pi_\theta(y \mid x)$ because π_{ref} is fixed. Finally, applying the same log-derivative trick with $f \equiv 1$ shows that $\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[\nabla_\theta \log \pi_\theta(y \mid x)] = 0$ for every x. The second expectation above therefore vanishes, giving the exact on-policy gradient.

$$
\nabla_\theta J_{\mathrm{KL},\phi}(\pi_\theta) = \mathbb{E}_{x\sim\mathcal{D}}\big[\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[\tilde{r}_{\mathrm{KL},\theta}(x,y)\nabla_\theta \log \pi_\theta(y \mid x)]\big] . \tag{10}
$$

[10]The identity starts by differentiating the log probability: $\nabla_\theta \log \pi_\theta(y \mid x) = \nabla_\theta\pi_\theta(y \mid x)/\pi_\theta(y \mid x)$. Multiplying by $\pi_\theta(y \mid x)$ gives $\nabla_\theta\pi_\theta(y \mid x) = \pi_\theta(y \mid x)\nabla_\theta \log \pi_\theta(y \mid x)$. Hence, for fixed f, $\nabla_\theta\mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[f(x,y)] = \sum_y \nabla_\theta\pi_\theta(y \mid x)f(x,y) = \sum_y \pi_\theta(y \mid x)f(x,y)\nabla_\theta \log \pi_\theta(y \mid x) = \mathbb{E}_{y\sim\pi_\theta(\cdot|x)}[f(x,y)\nabla_\theta \log \pi_\theta(y \mid x)]$ (Williams, 1992).

Although the return is defined at the response level, the autoregressive factorization gives $\nabla_\theta \log \pi_\theta(y \mid x) = \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(y_t \mid x, y_{<t})$, so the same scalar return weights the gradient contribution from every token in the response.

Note that Equation (10) gives the exact gradient, which implementations estimate by averaging $\tilde{r}_{\mathrm{KL},\theta}(x,y)\nabla_\theta \log \pi_\theta(y \mid x)$ over a rollout batch. This matches the gradient obtained by treating $\tilde{r}_{\mathrm{KL},\theta}$ as fixed only because $\mathbb{E}[\nabla_\theta \ell_\theta(x,y)] = 0$. For a single rollout, $\nabla_\theta \ell_\theta(x,y)$ is generally nonzero, so full sample-level differentiation adds $-\beta \nabla_\theta \ell_\theta(x,y)$. This term may be omitted without bias because its expectation is zero.[11]

7.2.2 Detached KL-shaped reward view

Regardless of how we arrive at the on-policy gradient in (10), implementations usually compute the KL-shaped return on the rollout batch and detach it before taking the policy gradient.[12] This is equivalent to optimizing a surrogate objective formed by holding the rollout KL-shaped reward fixed, rather than optimizing $J_{\mathrm{KL},\phi}(\pi_\theta)$ directly for all θ. To formalize this convention, fix $\theta^- \leftarrow \theta$ and define the detached KL-shaped reward

$$\tilde{r}_{\mathrm{KL},\theta^-}(x,y) = r_\phi(x,y) - \beta \ell_{\theta^-}(x,y) = r_\phi(x,y) - \beta \log \frac{\pi_{\theta^-}(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)}.$$

The corresponding surrogate is $\tilde{J}^{-}_{\mathrm{KL},\phi}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}[\tilde{r}_{\mathrm{KL},\theta^-}(x,y)]$, which is a local approximation at the rollout parameter θ^- and not globally the same as $J_{\mathrm{KL},\phi}(\pi_\theta)$. Locally, however, it agrees with the original objective to first order.[13] Since θ^- is fixed, the detached KL-shaped reward is now an ordinary scalar reward for the purpose of the update. Applying the log-derivative trick gives the same REINFORCE equation as (10), except with $\tilde{r}_{\mathrm{KL},\theta}$ replaced by $\tilde{r}_{\mathrm{KL},\theta^-}$.

7.2.3 Discussion

REINFORCE is fully *on-policy*: each rollout batch is used for one update at the policy that generated it, then discarded. At the update point, we have $\theta^- = \theta$, so the surrogate gradient coincides with (10).

[11] Simply differentiating the sampled log-ratio does not give the full KL gradient because automatic differentiation treats the sampled response y as fixed and misses how θ changes its sampling probability.

[12] Here "detach" means that the quantity is treated as fixed scalar data during automatic differentiation. Gradients are taken through $\log \pi_\theta(y \mid x)$, but not through the return multiplying it.

[13] Indeed, $J_{\mathrm{KL},\phi}(\pi_\theta)$-$\tilde{J}^{-}_{\mathrm{KL},\phi}(\theta) = -\beta\, \mathbb{E}_{x\sim\mathcal{D}}[\mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\theta^-}(\cdot \mid x))]$, whose value and gradient are zero at $\theta = \theta^-$.

REINFORCE then samples a new batch from the updated policy. Thus, in the frozen-snapshot view above, the notation of θ^- is mainly a formalization of the existing implementation, but both derivations lead to the same algorithm. The distinction between the frozen surrogate and the original objective becomes more important in algorithms that allow multiple optimization steps per rollout batch; we discuss this more in Section 7.3 on PPO.

One downside of fully on-policy algorithms is that for large LLMs, generating responses is often the expensive part of training, and REINFORCE uses each generated batch only once. REINFORCE also has high variance because the same noisy scalar return multiplies the log-probability gradient for the entire response. A standard fix subtracts a prompt-dependent *baseline*, replacing $\tilde{r}_{\mathrm{KL},\theta^-}(x, y)$ with $\tilde{r}_{\mathrm{KL},\theta^-}(x, y) - b(x)$.[14] The baseline can be a running average, a learned predictor, or another prompt-level estimate. In *REINFORCE leave-one-out (RLOO)*, we sample K independent responses $y_1, \ldots, y_K$ for the same prompt and let $y_{-i} = (y_j)_{j \neq i}$. The random baseline $\hat{b}_i(x; y_{-i}) = (K-1)^{-1} \sum_{j \neq i} \tilde{r}_{\mathrm{KL},\theta^-}(x, y_j)$ excludes y_i, so, conditional on x, it is independent of the response being scored and preserves an unbiased policy gradient. The update for y_i uses $\tilde{r}_{\mathrm{KL},\theta^-}(x, y_i) - \hat{b}_i(x; y_{-i})$, comparing each response with the others sampled for the same prompt (Ahmadian et al., 2024).

Takeaway. REINFORCE applies the vanilla policy-gradient algorithm to LLMs using response-level rollouts and scalar returns, which makes it easy to implement. When the return includes the policy-dependent KL term, we can differentiate the objective exactly or freeze the term at the rollout policy; both treatments give the same expected gradient at the on-policy update point. Two downsides of REINFORCE are that it does not reuse previously collected rollouts and has high variance. Common variance-reduction extensions use a prompt-dependent baseline, and RLOO estimates it from other responses sampled for the same prompt.

7.3 PPO (starting from REINFORCE)

PPO (Schulman et al., 2017) differs from REINFORCE primarily in that it performs *multiple optimization steps* per rollout, addressing the efficiency issue of REINFORCE. Throughout this section, we take the detached KL-shaped reward view from Section 7.2, where the KL-shaped reward

[14] Any $b(x)$ independent of the response leaves the expected policy gradient unchanged, since $\mathbb{E}_{y \sim \pi_\theta(\cdot|x)}[b(x) \nabla_\theta \log \pi_\theta(y \mid x)] = b(x) \nabla_\theta \sum_y \pi_\theta(y \mid x) = 0$; see Sutton and Barto (2018).

is computed on the rollout batch and held fixed during the update. We also consider $\pi_{\theta_{\text{old}}}$ to be a fixed rollout policy that generated the current batch, and therefore suppress the dependence of certain quantities on θ_{old} for ease of notation. Here π_{ref} is the fixed reference model, whereas $\pi_{\theta_{\text{old}}}$ is the rollout policy refreshed after each iteration. We now walk the reader through the derivation of the PPO objective, starting from $J_{\text{KL},\phi}(\pi_\theta)$ in (5).

We first write this return in the token-level MDP of Definition 2. Let $s_t = (x, y_{<t})$ and $a_t = y_t$. By factoring the response-level log-ratio over tokens and placing the reward-model score at the terminal token, we obtain:

$$\tilde{r}_{\text{KL},t}(s_t, a_t) = \begin{cases} -\beta \log \dfrac{\pi_{\theta_{\text{old}}}(a_t \mid s_t)}{\pi_{\text{ref}}(a_t \mid s_t)} & t < T, \\ r_\phi(x, y) - \beta \log \dfrac{\pi_{\theta_{\text{old}}}(a_T \mid s_T)}{\pi_{\text{ref}}(a_T \mid s_T)} & t = T. \end{cases}$$

Observe that $\sum_{t=1}^{T} \tilde{r}_{\text{KL},t}(s_t, a_t) = r_\phi(x, y) - \beta \log(\pi_{\theta_{\text{old}}}(y \mid x)/\pi_{\text{ref}}(y \mid x))$, so we have simply decomposed the response-level reward into token-level rewards. The token-level objective is

$$\tilde{J}_{\text{KL},\phi}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T} \tilde{r}_{\text{KL},t}(s_t, a_t)\right], \tag{11}$$

which is a local surrogate for $J_{\text{KL},\phi}(\pi_\theta)$ around θ_{old}[15], since $\tilde{r}_{\text{KL},t}(s_t, a_t)$ is defined for a fixed θ_{old}. As in the REINFORCE derivation, it agrees with $J_{\text{KL},\phi}(\pi_\theta)$ to first order at the rollout policy. PPO optimizes this surrogate for several steps on the same batch.

By the performance-difference lemma[16] of Kakade and Langford (2002), stated in a convenient form in equation 1 of Schulman et al. (2015) and proved in their Appendix A, the objective in (11) can be written as a sum of advantages under $\pi_{\theta_{\text{old}}}$. Define $A_t(x, y_{\le t}) \equiv A^{\pi_{\theta_{\text{old}}}}(s_t, a_t) = Q^{\pi_{\theta_{\text{old}}}}(s_t, a_t) - V^{\pi_{\theta_{\text{old}}}}(s_t)$ (defined in the token-level MDP whose rewards $\tilde{r}_{\text{KL},t}$ include the KL term). Then

$$\tilde{J}_{\text{KL},\phi}(\theta) = \tilde{J}_{\text{KL},\phi}(\theta_{\text{old}}) + \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|x)}\left[\sum_{t=1}^{T} A_t(x, y_{\le t})\right]. \tag{12}$$

[15] Section 7.2 wrote the corresponding surrogate as $\tilde{J}^{-}_{\text{KL},\phi}$ to make its dependence on the frozen rollout parameter θ^- explicit. Here we suppress the analogous dependence on θ_{old} for simplicity of notation.

[16] The lemma says that improvement over the rollout policy equals the expected sum of rollout-policy advantages along responses sampled from the new policy.

Because $\tilde{J}_{\mathrm{KL},\phi}(\theta_{\mathrm{old}})$ does not depend on θ, we use $\doteq$ below to denote equality up to this additive constant. Also, we use the notation $A_t(x, y_{\le t})$ to emphasize that the advantage is for the whole state-action prefix. Equation (12) is still an expectation under π_θ, while the rollout batch was sampled from $\pi_{\theta_{\mathrm{old}}}$. We can apply an importance correction for each advantage term individually:

$$\tilde{J}_{\mathrm{KL},\phi}(\theta) \doteq \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_{\theta_{\mathrm{old}}}(\cdot|x)}\left[\sum_{t=1}^{T} \frac{\pi_\theta(y_{\le t} \mid x)}{\pi_{\theta_{\mathrm{old}}}(y_{\le t} \mid x)} A_t(x, y_{\le t})\right]. \tag{13}$$

Zhang et al. (2026) call the factor $\pi_\theta(y_{\le t} \mid x)/\pi_{\theta_{\mathrm{old}}}(y_{\le t} \mid x)$ the *cumulative token importance sampling ratio*. It is the right correction because $A_t(x, y_{\le t})$ depends on the whole prefix $y_{\le t}$, which was sampled from $\pi_{\theta_{\mathrm{old}}}$, so we reweight by that prefix's likelihood ratio. By the autoregressive factorization (1), this is the product of the per-token ratios up to t. Note that this is different from a ratio that corrects only for the last token, the standard approximation that readers may be familiar with (see below).

Starting from (13), which is impractical to optimize because its exact advantages are unknown and its cumulative ratios can have high variance, we make four practical changes:

- *Estimated advantage.* The exact advantage $A_t(x, y_{\le t})$ is unknown, so we replace it with an estimate $\hat{A}(s_t, a_t)$, usually computed by generalized advantage estimation (GAE) (Schulman et al., 2016) from a learned *value function* $V_\psi(s_t)$ that predicts the return after a partial response. Writing $s_t = (x, y_{<t})$, with $V_\psi(s_{T+1}) = 0$ at the terminal state, GAE forms per-token temporal-difference errors δ_t and sets

$$\hat{A}(s_t, a_t) = \sum_{\ell=t}^{T} \lambda^{\ell-t}\delta_\ell, \qquad \delta_\ell = \tilde{r}_{\mathrm{KL},\ell} + V_\psi(s_{\ell+1}) - V_\psi(s_\ell), \tag{14}$$

  where $\gamma = 1$ as in Definition 2, and $\lambda \in [0, 1]$ trades off variance against reliance on the value function.[17] We are presenting (14) here primarily to convey the idea that $\hat{A}(s_t, a_t)$ compares realized rewards against value-function predictions, but note that GAE is a subtle and complex topic in its own right, and we refer the reader to Schulman et al. (2016) for a full treatment.

- *Local importance ratio.* PPO replaces the cumulative ratio $\pi_\theta(y_{\le t} \mid x)/\pi_{\theta_{\mathrm{old}}}(y_{\le t} \mid x)$ in (13) with the single local token ratio $\rho_t(\theta) = \pi_\theta(y_t \mid x, y_{<t})/\pi_{\theta_{\mathrm{old}}}(y_t \mid x, y_{<t})$, keeping only the factor for

[17] Smaller λ leans on the value predictions and lowers variance when V_ψ is accurate; larger λ uses more realized reward and reduces bias from bootstrapping on an imperfect value function. The value function V_ψ is trained alongside the policy on the same rollout batch by regression toward the return estimates $\hat{A}(s_t, a_t) + V_\psi(s_t)$.

token t and dropping the earlier factors in the product. The cumulative product contains up to T factors and becomes noisy for long responses, whereas the local ratio can be more stable. For further details and another importance-ratio strategy, see the discussion in Zhang et al. (2026).

- *Clipping.* PPO clips the (local) importance ratio to $[1-\epsilon, 1+\epsilon]$ and takes the *minimum* of the clipped and unclipped terms, making the objective a pessimistic (lower) bound on the unclipped one. This removes any incentive to push $\rho_t(\theta)$ beyond the interval: when $\hat{A}(s_t, a_t) > 0$ the term stops improving once $\rho_t(\theta)$ reaches $1+\epsilon$, and when $\hat{A}(s_t, a_t) < 0$ it stops once $\rho_t(\theta)$ reaches $1-\epsilon$, which discourages large favorable changes.

- *Length averaging.* We divide the summed token terms for each response by its length before averaging across the batch, giving each response equal weight. This normalization is an implementation choice helpful in the setting of variable-length LLM responses. The original PPO objective instead averages over all sampled time steps, so longer responses receive more weight (Schulman et al., 2017).

Together, these changes transform (13) into the clipped surrogate

$$J_{\mathrm{PPO}}^{\mathrm{policy}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, y\sim\pi_{\theta_{\mathrm{old}}}(\cdot|x)}\left[\frac{1}{T}\sum_{t=1}^{T}\min\Big(\rho_t(\theta)\hat{A}(s_t,a_t),\ \mathrm{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}(s_t,a_t)\Big)\right]. \tag{15}$$

For a fixed rollout batch, the sampled tokens, rewards, and estimated advantages are treated as data; only the ratios depend on θ. PPO can take several optimization steps before refreshing $\pi_{\theta_{\mathrm{old}}}$ and sampling new responses. Note that Equation (15) only shows the policy surrogate, but PPO also fits V_ψ with a value regression loss, whose details are omitted here.

Takeaway. PPO uses the token-level formulation and combines a value function with a clipped importance ratio between the current and rollout-generating policies, letting it reuse each rollout batch for several gradient steps rather than discarding it after one update. It is flexible and works with learned reward models or verifiers, but it is the most infrastructure-heavy algorithm discussed here, requiring careful tuning of value-function fitting, KL penalties, and clipping.

7.4 GRPO (starting from PPO)

As in the PPO presentation above, we introduce GRPO using $J_{\mathrm{KL},\phi}(\pi_\theta)$ under a proxy reward model r_ϕ. However, note that GRPO is often used with verifiable rewards, and when these rewards come directly from the environment, the correct corresponding objective is actually $J_{\mathrm{KL}}(\pi_\theta)$, with r replacing r_ϕ. GRPO (Shao et al., 2024) builds upon PPO's surrogate (15) with two main innovations.

- *Group-mean advantage.* Instead of a learned value function V_ψ, GRPO samples a group of G responses $y_1, \ldots, y_G \sim \pi_{\theta_{\text{old}}}(\cdot \mid x)$ per prompt, scores each response with $r_\phi(x, y_i)$, and uses the group's mean as a prompt-specific baseline. Every token in response i receives the shared "advantage"[18]

$$\hat{A}_i = \frac{r_\phi(x, y_i) - \text{mean}(\{r_\phi(x, y_j)\}_{j=1}^G)}{\text{std}(\{r_\phi(x, y_j)\}_{j=1}^G)}.$$[19]

- *Separate KL loss.* Rather than folding the KL penalty into the reward, GRPO computes the advantage from r_ϕ alone and adds an explicit per-token KL penalty to the loss, comparing π_θ with the fixed reference policy π_{ref}, usually using Schulman's nonnegative k_3 estimator (Schulman, 2020) at π_θ (see Footnote 9 for a caveat):

$$\text{KL}_{i,t} = \frac{\pi_{\text{ref}}(y_{i,t} \mid x, y_{i,<t})}{\pi_\theta(y_{i,t} \mid x, y_{i,<t})} - \log \frac{\pi_{\text{ref}}(y_{i,t} \mid x, y_{i,<t})}{\pi_\theta(y_{i,t} \mid x, y_{i,<t})} - 1. \tag{16}$$

  With a verifiable reward, the KL penalty may be less necessary because there is no estimation error from a separately learned reward model, although verifier misspecification and reward hacking can remain. Some implementations omit the penalty (Yu et al., 2025); however, a positive penalty can help preserve capabilities of the model that the reward does not measure.

[18] Strictly speaking, $\hat{A}_i$ is not an advantage estimate. The group mean is not a valid baseline (it depends on the action being scored, since $r_\phi(x, y_i)$ appears in it; RLOO instead excludes y_i), and dividing by the group standard deviation rescales the estimate, so $\hat{A}_i$ no longer estimates $Q^\pi - V^\pi$. We nonetheless call it an advantage, following common usage.

[19] If all responses in the group receive the same reward, as can happen when all sampled solutions are correct or all are incorrect, the within-group standard deviation is zero. Implementations therefore need a small numerical stabilizer, filtering rule, or resampling/dynamic-sampling scheme.

Writing $\rho_{i,t} = \pi_\theta(y_{i,t} \mid x, y_{i,<t}) / \pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t})$, these two changes give the GRPO objective

$$J_{\text{GRPO}}(\theta) = \mathbb{E}_{x\sim\mathcal{D},\, \{y_i\}_{i=1}^{G}\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \left[\frac{1}{G}\sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \Big(\min\big(\rho_{i,t}\hat{A}_i,\ \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i\big) - \beta\, \text{KL}_{i,t} \Big) \right], \tag{17}$$

which is essentially (15) averaged over the group, with the group-based $\hat{A}_i$ replacing the value-function advantage and the KL penalty moved into the loss. GRPO is especially natural when many responses to the same prompt can be scored cheaply, as in math reasoning (Example 3), where a verifier can grade a whole group.

7.4.1 Practical limitations and extensions of GRPO

The details of the GRPO objective, especially its normalization and importance ratio terms, have a substantial effect on training. In the Dr. GRPO paper, Liu et al. (2025b) point out that the two normalization choices also reweight prompts and responses: dividing centered rewards by the within-group reward standard deviation when computing $\hat{A}_i$ introduces a question-level difficulty bias, while dividing each response's summed token loss by its length through $1/|y_i|$ introduces a response-level length bias.[20] Dr. GRPO proposes to remove both terms, using unstandardized centered rewards and a fixed length normalizer (a global constant) (Liu et al., 2025b).

Decoupled Clip and Dynamic Sampling Policy Optimization (DAPO) addresses the response-length bias induced by the $1/|y_i|$ normalization by replacing the *response-level averaging* in (17) (averaging token terms within each response, then averaging the G response means equally) with *token-level averaging* (averaging all tokens in the group together) (Yu et al., 2025). Writing $\ell_{i,t}(\theta) = \min(\rho_{i,t}\hat{A}_i, \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i) - \beta\, \text{KL}_{i,t}$ for the per-token GRPO term inside (17), the change is

$$\underbrace{\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\ell_{i,t}(\theta)}_{\text{response-level averaging}} \quad\longrightarrow\quad \underbrace{\frac{1}{\sum_{i=1}^{G}|y_i|}\sum_{i=1}^{G}\sum_{t=1}^{|y_i|}\ell_{i,t}(\theta)}_{\text{token-level averaging}}.$$

This weights each generated token equally, which is useful when training reasoning models whose response lengths vary substantially. DAPO also uses dynamic sampling to avoid all-correct or

[20] The group standard deviation gives larger updates to prompts with low reward variation, such as groups that are almost all correct or almost all incorrect. The $1/|y_i|$ factor gives longer responses smaller per-token updates, since every token in response i carries the same advantage $\hat{A}_i$; this favors brief correct answers and penalizes long incorrect answers too weakly.

all-incorrect prompt groups and decouples the clipping bounds, raising the upper bound to preserve exploration. As of July 2026, Hugging Face's Transformers Reinforcement Learning (TRL) library supports the original GRPO, DAPO, and Dr. GRPO objective normalizations, with the DAPO normalization as the current default (Hugging Face, 2026).

Beyond the averaging scheme, Group Sequence Policy Optimization (GSPO) changes the importance ratio itself, using a length-normalized ratio that compares the probabilities assigned to the complete response by the current and rollout policies; this matches the reward signal, which scores the response as a whole (Zheng et al., 2025). GSPO then clips and optimizes this ratio at the sequence level rather than clipping separate token-level ratios. Zheng et al. (2025) report improved training for Qwen3 models using GSPO.

Takeaway. GRPO keeps the PPO idea of clipped token-level updates but replaces the learned value function with a group-relative baseline. It is widely used for current LLM post-training in verifiable-reward reasoning tasks because it avoids training a value model while still allowing multiple optimization steps per rollout batch. In practice, one needs to consider details like length normalization and within-group reward variation in order to achieve good performance (see Dr. GRPO and DAPO).

8 Deployment

When instantiated with a learned reward model or offline verifier, the algorithms in Section 7 run entirely "inside the computer," without collecting feedback from deployed users. This section distinguishes on-policy optimization from online learning through real interactions, and then introduces batch-online deployment, where each deployed policy collects the data used to train the next policy.

8.1 Ambiguity of the term "online"

In the LLM post-training community, "online" often refers to algorithms that sample fresh responses from the current policy and then score them, with PPO being the standard example (Schulman et al., 2017). These algorithms use *on-policy rollouts*. By contrast, "offline" algorithms such as DPO train on a fixed preference dataset and do not sample new responses from the current policy during optimization (Rafailov et al., 2023).

In this tutorial, we use "online RL" from a deployment perspective to mean learning that incorporates new feedback from the environment in which the policy is deployed. PPO or GRPO training against a

learned reward model r_ϕ uses fresh on-policy rollouts, but scoring is done by the learned reward model rather than real users, so the deployment environment[21] remains outside the training loop (Figure 4). Environment feedback enters only after the policy is deployed.

Takeaway. We advocate describing algorithms that generate fresh samples from the current policy as using "on-policy rollouts," as we do throughout this tutorial, and reserving "online RL" for training that incorporates new feedback from the deployment environment.

8.2 The batch-online deployment paradigm

In *fully offline* deployment, a policy is trained once on fixed data and never updated, which is stable but not adaptable. In *fully online* deployment, the policy updates continuously from live feedback, making it "online" RL in the sense of Section 8.1. This adapts quickly, but deploying every update is often impractical and unsafe because policy quality may vary during training.

Batch-online training (Jiang et al., 2026) is a common production pattern in which the current policy is deployed, a batch of interaction data is collected (e.g., over an A/B test lasting days or weeks), and the policy is retrained offline on that data before being redeployed (see Figure 6). By using large batches to slow the update cadence, batch-online training retains some of the stability of offline training while still letting the system adapt. This framework matches the update pattern studied in batch-mode RL and approximate dynamic programming, where a fixed batch of data is used to update a policy or value function offline. Classical examples include *fitted Q-iteration*, *fitted value iteration*, *approximate value iteration*, and *approximate policy iteration* (Bertsekas and Tsitsiklis, 1996; de Farias and Van Roy, 2000; Gordon, 1999; Powell, 2011; Munos, 2003, 2005; Ernst et al., 2005; Szepesvári and Munos, 2005; Van Roy, 2006; Munos, 2007; Lange et al., 2012; Chen and Jiang, 2019; Munos and Szepesvári, 2008; Farahmand et al., 2010).

9 Multi-Turn Policy Optimization

In the single-turn setting (see Section 7), transitions are deterministic, so the reward is the only component of the MDP that must be learned or specified. The multi-turn setting (see Section 4.2) is harder because the environment dynamics are external to the model and may be stochastic or

[21] A verifiable reward (Section 5.1) may or may not coincide with the deployment environment. If it does, on-policy optimization against it constitutes online RL in this sense.

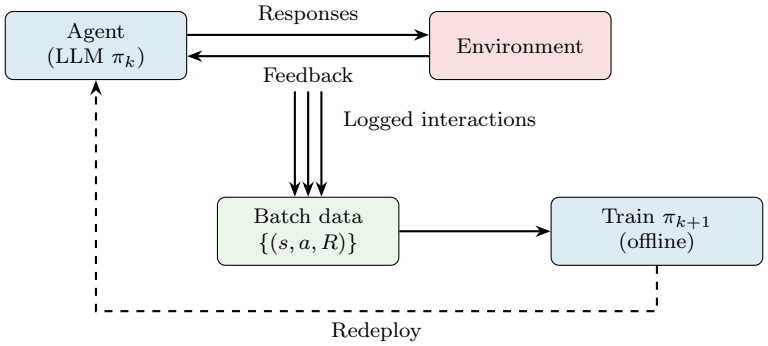

Figure 6: Batch-online training. At iteration k, the current policy π_k is deployed to interact with real users. Interaction data (prompts, responses, and observed rewards) is collected over the deployment period. A new policy π_{k+1} is then retrained offline on this data and redeployed for the next iteration.

unknown, so they must be handled either through interaction with a training environment or through logged trajectories, both of which we discuss in this section.

9.1 Multi-turn GRPO

A natural extension of single-turn GRPO applies policy-gradient updates to complete multi-turn trajectories. In the response-level MDP of Definition 6, the policy generates a complete response a_h at each turn, after which the environment returns the next message x_{h+1}. We consider undiscounted sparse rewards, with zero reward at intermediate turns and a single terminal reward. Variants of this approach have been applied to coding agents (FAIR CodeGen team et al., 2025), web agents (Wei et al., 2025b), and reasoning agents (Wang et al., 2025).

Following FAIR CodeGen team et al. (2025), the rollout policy $\pi_{\theta_{\text{old}}}$ interacts with the environment to generate a group of trajectories $\tau_1, \ldots, \tau_G$ from each initial user message x_1. Each trajectory alternates between agent responses and environment messages and receives terminal reward R_i. We tokenize trajectory i as z_i, with tokens $z_{i,t}$ indexed by $t = 1, \ldots, |z_i|$, define $M_{i,t} = 1$ for agent-generated tokens and $M_{i,t} = 0$ otherwise, and let $L_i = \sum_{t=1}^{|z_i|} M_{i,t}$ denote the number of agent tokens. The group baseline $\bar{R} = G^{-1} \sum_{j=1}^{G} R_j$ gives the trajectory-level advantage $\hat{A}_i = R_i - \bar{R}$.[22] For $M_{i,t} = 1$, write $\rho_{i,t} = \pi_\theta(z_{i,t} \mid z_{i,<t}) / \pi_{\theta_{\text{old}}}(z_{i,t} \mid z_{i,<t})$ and define the per-token term

$$\ell_{i,t}(\theta) = \min\big(\rho_{i,t}\hat{A}_i,\ \text{clip}(\rho_{i,t}, 1-\epsilon, 1+\epsilon)\hat{A}_i\big) - \beta\, \text{KL}_{i,t}.$$

[22] FAIR CodeGen team et al. (2025) instead set $\hat{A}_i = R_i - \mu$ using the length-weighted baseline $\mu = L^{-1} \sum_{i=1}^{G} L_i R_i$, where $L = \sum_i L_i$. This centers the advantages under token weighting because $\sum_i L_i \hat{A}_i = 0$. Their implementation also drops standard-deviation normalization, divides by a fixed maximum token budget rather than trajectory length, and omits the KL penalty.

The quantity $\mathrm{KL}_{i,t}$ is the per-token KL penalty in (16), evaluated at agent-generated token $z_{i,t}$. The masked GRPO objective[23] is

$$J_{\text{MT-GRPO}}(\theta) = \mathbb{E}_{x_1 \sim \mathcal{D},\, \{\tau_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{old}}}(\cdot|x_1)} \left[\frac{1}{G} \sum_{i=1}^{G} \frac{1}{L_i} \sum_{t=1}^{|z_i|} M_{i,t}\, \ell_{i,t}(\theta) \right].$$

The objective applies one trajectory-level advantage to all agent tokens, while environment tokens remain in the prefix but are excluded from the policy-ratio and KL terms.

Remark 9 (Baseline computation in multi-turn GRPO)**.** *There is one subtle point in multi-turn GRPO's baseline computation. In single-turn GRPO, all G responses start from the same prompt x, so $\bar{R}$ plays the role of a prompt-level baseline $V(x)$ and $\hat{A}_i$ approximates $Q(x, y_i) - V(x)$ (with the caveats of Section 7.4). In multi-turn GRPO, the trajectories share only the initial message x_1. After the environment responds, later actions are taken from different states $s_h^{(i)}$. The trajectory-level baseline still depends only on x_1, so it remains a sensible group baseline, but the resulting "advantage" is no longer an estimate of a state-dependent quantity $Q(s_h^{(i)}, a_h^{(i)}) - V(s_h^{(i)})$. In fact, it assigns the same value to every turn of the trajectory.*

Branching-based GRPO variants form local comparison groups by sampling multiple continuations from intermediate turns, allowing actions to be compared under a shared prefix (Wei et al., 2025a; Ji et al., 2026; Samanta et al., 2026).[24] Group-in-Group Policy Optimization (GiGPO) and Generalized Advantage Grouped Policy Optimization (GAGPO) avoid explicit branching by running ordinary trajectory groups and then grouping identical environment states that occur naturally in the collected batch, which works when repeated states are available (Feng et al., 2025; Zhu et al., 2026).

Remark 10 (Value functions for multi-turn RL)**.** *Alternatively, value functions can provide finer credit assignment through separate advantage estimates at each turn. ArCHer combines an utterance-level critic with a token-level actor, while Turn-PPO applies critic-based PPO to agentic LLMs and reports greater robustness than GRPO on long-horizon tasks (Zhou et al., 2024; Li et al., 2025b). Despite*

[23] The derivation parallels the PPO and GRPO derivations in Section 7.3, except that the trajectory law also contains environment transitions and policy terms occur only at agent-generated positions. Since P is independent of θ, its terms vanish when differentiating the log trajectory probability and cancel from the exact cumulative importance ratio; multi-turn GRPO then applies the same local-ratio approximation at the remaining agent-token positions.

[24] Note that Samanta et al. (2026) operate in a single-turn RLVR setting, where the reasoning trace is broken into multiple discrete "thoughts." This becomes similar to a multi-turn problem, except that the environment does not produce responses between turns. Their algorithm performs resets to intermediate reasoning steps, resamples a group, and applies GRPO with the fresh group at the intermediate state.

the recent shift toward GRPO and other methods that avoid value-function estimation, the GLM-5.2 release post describes the use of critic-based PPO in the model's training procedure (Z.ai, 2026). For simplicity, this subsection uses GRPO to introduce the main ideas of multi-turn RL and omits a full treatment of value-function methods, although Section 9.3 presents an approach that fits a Q-function from Monte Carlo returns.

Remark 11 (Preference-based multi-turn RL)**.** *When feedback is comparative rather than scalar, multi-turn methods can learn from preferences between complete trajectories. Existing approaches formulate preference-based RL over full conversations or adapt DPO using occupancy-measure regularization and trajectory-length normalization (Shani et al., 2024; Shi et al., 2024).*

9.2 User simulators and training environments

The previous subsection treated multi-turn rollouts as given, but what does producing a multi-turn rollout actually entail? In single-turn RL post-training, the policy can generate a full response by simply sampling tokens from the model. In multi-turn RL post-training, however, an *external environment* must supply observations between model responses to form a full rollout. In the conversational setting of Example 4, that observation is the user's next message. In the coding-agent setting of Example 5, it could be the output of a terminal command or API call.

For many applications, this environment has to be built before RL can begin, and current work does so in three broad ways. The first is *user simulators*, in which an LLM produces the user's next message in dialogue or customer-facing interactions (Hu et al., 2023; Luo et al., 2024; Shaikh et al., 2025; Wu et al., 2026; Tennenholtz et al., 2026). This allows tool-agent benchmarks such as τ-bench and τ^2-bench to evaluate complete interactions without real users (Yao et al., 2024; Barres et al., 2025), although simulated personas can introduce bias and require careful validation (Li et al., 2025a). The second approach uses *real executable environments*, where coding agents modify codebases and run tests, web agents use browsers, and computer-use agents operate an operating system, as in SWE-bench, WebArena, and OSWorld (Jimenez et al., 2024; Zhou et al., 2023; Xie et al., 2024). The third approach uses *synthetic tool-use tasks* that automatically generate tool specifications, tasks, and rubrics (Qin et al., 2023; Kimi Team et al., 2025; Castellani et al., 2025). Kimi K2, for example, generates tools, tasks, and trajectories, then filters them with rubrics and real sandboxes (Kimi Team et al., 2025).

9.3 Batch-online multi-turn RL from logged trajectories

For many bespoke applications, reliable simulators or executable environments are unavailable, leaving only *logged trajectories* from production. The conversational-agent setting in Example 4 is typical because deployment produces conversations with observed outcomes but no reliable model of how users would react to different responses. Jiang et al. (2026) adapt *approximate policy iteration*[25] to this setting by reusing the batch-online deployment loop described in Section 8. Each deployment round supplies new trajectories for policy evaluation, after which policy improvement runs offline against the fitted Q-function. The resulting algorithm, *Iterative GRPO*, therefore requires no separate training environment and runs the following evaluation/improvement loop in each round k:

- *Data collection.* The system deploys π_k and collects a batch of real multi-turn trajectories with observed outcomes (say, over a week of data collection).
- *Value fitting.* For a trajectory of length H with realized terminal reward R_H, the method assigns each logged turn h the return-to-go $G_h = \gamma^{H-h} R_H$, with discount factor $\gamma \in [0, 1]$. The resulting dataset $\{(s_h, a_h, G_h)\}$ is used to fit Q^{π_k} with standard reward modeling methods.
- *Policy improvement.* The method holds Q^{π_k} fixed, treats each logged conversation state as a single-turn prompt, and computes an approximate policy-improvement step.[26] As in the single-turn setting, this step uses the KL-regularized form in (5), with $Q^{\pi_k}(s, a)$ replacing the single-turn scalar reward:

$$\pi_{k+1}(\cdot \mid s) \;\approx\; \arg\max_{\mu} \left[\mathbb{E}_{a\sim\mu}[Q^{\pi_k}(s, a)] - \beta\, \mathrm{KL}(\mu \,\|\, \pi_{\mathrm{ref}}(\cdot \mid s))\right]. \tag{18}$$

  The updated policy is then redeployed.

After redeployment, the same loop repeats with a new batch of real trajectories. The remaining question is: how do we implement the policy-improvement step in (18), which is an optimization problem over responses? Jiang et al. (2026) make the following observation.

[25] Policy iteration alternates between evaluating the current policy and improving it using the resulting value estimates. In the exact finite-MDP case, the policy-improvement theorem guarantees that the next policy is no worse than the previous one at every state, and repeated exact policy iteration converges to an optimal policy. See Sutton and Barto (2018) and Bertsekas and Tsitsiklis (1996) for full treatments of these algorithms.

[26] Note that Q^{π_k} is fit only on data covered by π_k, so it can be unreliable for the actions that policy improvement searches over. Jiang et al. (2026) note this limitation and discuss exploration or KL-restrained updates to mitigate it.

Observation (Single-turn GRPO as an approximate policy-improvement operator). In (18), the action a is a complete response. Optimizing over a is therefore a single-turn RL problem at the *token level*, with the Q-function replacing the reward model. The fitted Q-function already summarizes the future interaction and eventual outcome after a, so policy improvement only needs to generate a and does not wait for another environment response. Figure 3 illustrates the token-level generation within each multi-turn action. This is an *approximate* policy-improvement step because it fits Q^{π_k} from limited-coverage data and uses the KL-regularized GRPO surrogate in (17) rather than an exact greedy update.

This observation reduces multi-turn training to two off-the-shelf single-turn RL post-training components. A regression step fits the Q-function from logged returns, and a single-turn optimizer, such as GRPO, improves the policy against the fitted Q-function. The method therefore does not require a user simulator or an interactive training environment, and instead leverages batch-online deployment for policy improvement.

10 Open Research Directions

One challenge in RLHF is that human feedback is difficult to obtain at scale. Constitutional AI (Bai et al., 2022) and self-improvement methods (Chen et al., 2024; Samanta et al., 2025; Zuo et al., 2025; Wang et al., 2023) generate synthetic preference data or reward signals using LLMs themselves, enabling large-scale training without human annotators. Important open questions include understanding how to best combine AI-generated and human preferences, when AI feedback degrades in quality, and how feedback quality changes as it is iteratively recycled through multiple rounds of training.

Related to the above, most current RLHF methods are passive in that they optimize on data collected by the current or previous policy without deliberately seeking informative interactions. Connecting RLHF to the rich literature on exploration, bandits, and active learning has only recently begun (Dwaracherla et al., 2024; Asghari et al., 2026). Designing exploration strategies for the RLHF setting that can efficiently and continuously collect human feedback remains an open challenge. More broadly, this connects to continual and lifelong learning, where the agent must adapt to shifts in the environment without forgetting previously learned behavior.

As LLMs evolve from single-turn assistants into agents that browse the web, write code, and conduct extended conversations, multi-turn RL is becoming central (see Section 9). Key challenges include credit assignment over long horizons, safe exploration during deployment, partial observability (Ma et al., 2024), context management (Rajasekaran et al., 2025), and harness design (Lee et al., 2026). A further challenge is to build high-fidelity models of the deployment environment, such as user simulators (see Section 9.2), that remain consistent across multi-turn interactions (Laban et al., 2025), and to determine how they should be updated as the policy and deployment data evolve.

Acknowledgments

We thank four anonymous reviewers, the volume editors, and David Alderson, the series editor, for their helpful comments and in-depth reviews. We also thank Mina Lee for multiple rounds of careful proofreading and thoughtful comments that significantly improved this tutorial.

References

A. Ahmadian, C. Cremer, M. Gallé, M. Fadaee, J. Kreutzer, O. Pietquin, A. Üstün, and S. Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, 2024.

K. Asadi, J. Han, I. Pipano, X. Xu, D. Perrault-Joncas, S. Sabach, K. Bouyarmane, and M. Ghavamzadeh. C2-DPO: Constrained controlled direct preference optimization. *arXiv preprint arXiv:2502.17507*, 2025.

S. M. Asghari, C. Chute, V. Dwaracherla, X. Lu, M. Jafarnia, V. Minden, Z. Wen, and B. Van Roy. Efficient exploration at scale. *arXiv preprint arXiv:2603.17378*, 2026.

Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

V. Barres, H. Dong, S. Ray, X. Si, and K. Narasimhan. τ^2-bench: Evaluating conversational agents in a dual-control environment. *arXiv preprint arXiv:2506.07982*, 2025.

D. P. Bertsekas. *Reinforcement Learning and Optimal Control*. Athena Scientific, 2019.

D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

R. A. Bradley and M. E. Terry. Rank analysis of incomplete block designs: I. The method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

E. Brynjolfsson, D. Li, and L. Raymond. Generative AI at work. *The Quarterly Journal of Economics*, 140(2):889–942, 2025.

S. Casper, X. Davies, C. Shi, T. K. Gilbert, J. Scheurer, J. Rando, R. Freedman, T. Korbak, D. Lindner, P. Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.

T. Castellani, N. Ye, D. Mittal, T. Yen, E. Koukoumidis, W. Zeng, and H. Namkoong. SynthTools: A framework for scaling synthetic tools for agent development. *arXiv preprint arXiv:2511.09572*, 2025.

J. Chen and N. Jiang. Information-theoretic considerations in batch reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1042–1051. PMLR, 2019.

Z. Chen, Y. Deng, H. Yuan, K. Ji, and Q. Gu. Self-play fine-tuning converts weak language models to strong language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 6621–6642. PMLR, 2024.

P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30, pages 4299–4307, 2017.

W. Chu and Z. Ghahramani. Preference learning with Gaussian processes. In *International Conference on Machine Learning*, pages 137–144, 2005.

K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. *Advances in Neural Information Processing Systems*, 31: 4754–4765, 2018.

A. T.-H. Chung, B. Zhang, L.-C. Kung, H. Bastani, and O. Bastani. Effective personalized AI tutors via LLM-guided reinforcement learning. *SSRN preprint 6423358*, 2026.

T. Coste, U. Anwar, R. Kirk, and D. Krueger. Reward model ensembles help mitigate overoptimization. In *International Conference on Learning Representations*, 2024.

D. P. de Farias and B. Van Roy. On the existence of fixed points for approximate value iteration and temporal-difference learning. *Journal of Optimization Theory and Applications*, 105(3):589–608, 2000.

DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

V. Dwaracherla, S. M. Asghari, B. Hao, and B. Van Roy. Efficient exploration for LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12215–12227. PMLR, 2024.

J. Eisenstein, C. Nagpal, A. Agarwal, A. Beirami, A. D'Amour, D. Dvijotham, A. Fisch, K. Heller, S. Pfohl, D. Ramachandran, et al. Helping or herding? Reward model ensembles mitigate but do not eliminate reward hacking. In *Conference on Language Modeling*, 2024.

D. Ernst, P. Geurts, and L. Wehenkel. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

K. Ethayarajh, W. Xu, N. Muennighoff, D. Jurafsky, and D. Kiela. Model alignment as prospect theoretic optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 12634–12651. PMLR, 2024.

FAIR CodeGen team, J. Copet, Q. Carbonneaux, G. Cohen, J. Gehring, J. Kahn, J. Kossen, F. Kreuk, E. McMilin, M. Meyer, et al. CWM: An open-weights LLM for research on code generation with world models. *arXiv preprint arXiv:2510.02387*, 2025.

A. M. Farahmand, R. Munos, and C. Szepesvári. Error propagation for approximate policy and value iteration. In *Advances in Neural Information Processing Systems*, volume 23, pages 568–576, 2010.

L. Feng, Z. Xue, T. Liu, and B. An. Group-in-group policy optimization for LLM agent training. *arXiv preprint arXiv:2505.10978*, 2025.

L. Gao, J. Schulman, and J. Hilton. Scaling laws for reward model overoptimization. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 10835–10866. PMLR, 2023.

M. Gheshlaghi Azar, Z. D. Guo, B. Piot, R. Munos, M. Rowland, M. Valko, and D. Calandriello. A general theoretical paradigm to understand learning from human preferences. In *Proceedings of the*

27th International Conference on Artificial Intelligence and Statistics, volume 238 of *Proceedings of Machine Learning Research*, pages 4447–4455. PMLR, 2024.

G. J. Gordon. *Approximate Solutions to Markov Decision Processes*. PhD thesis, Carnegie Mellon University, 1999.

A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

D. Guo, D. Yang, H. Zhang, J. Song, P. Wang, Q. Zhu, R. Xu, R. Zhang, S. Ma, X. Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645:633–638, 2025.

D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In *International Conference on Machine Learning*, pages 2555–2565. PMLR, 2019.

D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi. Dream to control: Learning behaviors by latent imagination. In *International Conference on Learning Representations*, 2020.

Z. Hu, Y. Feng, A. T. Luu, B. Hooi, and A. Lipani. Unlocking the potential of user feedback: Leveraging large language model as user simulator to enhance dialogue system. *arXiv preprint arXiv:2306.09821*, 2023.

Hugging Face. GRPO trainer. https://huggingface.co/docs/trl/grpo_trainer, 2026. Accessed July 24, 2026.

H. Ivison, Y. Wang, J. Liu, Z. Wu, V. Pyatkin, N. Lambert, N. A. Smith, Y. Choi, and H. Hajishirzi. Unpacking DPO and PPO: Disentangling best practices for learning from preference feedback. *arXiv preprint arXiv:2406.09279*, 2024.

M. Janner, J. Fu, M. Zhang, and S. Levine. When to trust your model: Model-based policy optimization. *Advances in Neural Information Processing Systems*, 32:12498–12509, 2019.

Y. Ji, Z. Ma, Y. Wang, G. Chen, X. Chu, and L. Wu. Tree search for LLM agent reinforcement learning. In *International Conference on Learning Representations*, 2026.

D. R. Jiang, A. Nikulkov, Y.-C. Chen, Y. Bai, and Z. Zhu. Improving generative ad text on Facebook using reinforcement learning. *arXiv preprint arXiv:2507.21983*, 2025.

D. R. Jiang, A. Samanta, J. Bhandari, A. Yang, R. Munos, and T. Lu. Aligning LLMs toward multi-turn conversational outcomes using iterative RLHF. In *Lifelong Agents Workshop at ICLR*, 2026.

C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

S. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML)*, pages 267–274, 2002.

T. Kaufmann, P. Weng, V. Bengs, and E. Hüllermeier. A survey of reinforcement learning from human feedback. *Transactions on Machine Learning Research*, 2025.

C. Kausik, A. Swaminathan, and N. Kallus. The context gathering decision process: A POMDP framework for agentic search. *arXiv preprint arXiv:2605.07042*, 2026.

Kimi Team, Y. Bai, Y. Bao, Y. Charles, C. Chen, G. Chen, H. Chen, H. Chen, J. Chen, N. Chen, R. Chen, Y. Chen, et al. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

P. Laban, H. Hayashi, Y. Zhou, and J. Neville. LLMs get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*, 2025.

N. Lambert. *Reinforcement Learning from Human Feedback*. Manning Publications, 2026.

N. Lambert, J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

S. Lange, T. Gabel, and M. Riedmiller. Batch reinforcement learning. In *Reinforcement Learning*, volume 12 of *Adaptation, Learning, and Optimization*, pages 45–73. Springer, 2012.

H. Lee, S. Phatale, H. Mansoor, T. Mesnard, J. Ferret, K. R. Lu, C. Bishop, E. Hall, V. Carbune, A. Rastogi, and S. Prakash. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback

with AI feedback. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 26874–26901. PMLR, 2024a.

M. Lee, K. I. Gero, J. J. Y. Chung, S. Buckingham Shum, V. Raheja, H. Shen, S. Venugopalan, T. Wambsganss, D. Zhou, E. A. Alghamdi, T. August, A. Bhat, M. Z. Choksi, S. Dutta, J. L. Guo, M. N. Hoque, Y. Kim, S. Knight, S. P. Neshaei, A. Sergeyuk, A. Shibani, D. Shrivastava, L. Shroff, J. Stark, S. Sterman, S. Wang, A. Bosselut, D. Buschek, J. C. Chang, S. Chen, M. Kreminski, J. Park, R. Pea, E. H. Rho, S. Z. Shen, and P. Siangliulue. A design space for intelligent and interactive writing assistants. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. ACM, 2024b.

Y. Lee, R. Nair, Q. Zhang, K. Lee, O. Khattab, and C. Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

A. Li, H. Chen, H. Namkoong, and T. Peng. LLM generated persona is a promise with a catch. *arXiv preprint arXiv:2503.16527*, 2025a.

J. Li, P. Zhou, R. Meng, M. P. Vadera, L. Li, and Y. Li. Turn-PPO: Turn-level advantage estimation with PPO for improved multi-turn RL in agentic LLMs. *arXiv preprint arXiv:2512.17008*, 2025b.

K. Liu, J. K. Liu, M. Chen, and Y. Liu. Rethinking KL regularization in RLHF: From value estimation to gradient optimization. *arXiv preprint arXiv:2510.01555*, 2025a.

Z. Liu, C. Chen, W. Li, P. Qi, T. Pang, C. Du, W. S. Lee, and M. Lin. Understanding R1-Zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025b.

J. Lu, J. Li, S. An, M. Zhao, Y. He, D. Yin, and X. Sun. Eliminating biased length reliance of direct preference optimization via down-sampled KL divergence. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1047–1067. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.emnlp-main.60.

X. Luo, Z. Tang, J. Wang, and X. Zhang. DuetSim: Building user simulator with dual large language models for task-oriented dialogues. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 5414–5424, 2024.

C. Ma, J. Zhang, Z. Zhu, C. Yang, Y. Yang, Y. Jin, Z. Lan, L. Kong, and J. He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pages 74325–74362, 2024.

Y. Meng, M. Xia, and D. Chen. SimPO: Simple preference optimization with a reference-free reward. In *Advances in Neural Information Processing Systems*, volume 37, pages 124198–124235, 2024.

T. Moskovitz, A. K. Singh, D. Strouse, T. Sandholm, R. Salakhutdinov, A. D. Dragan, and S. McAleer. Confronting reward model overoptimization with constrained RLHF. In *International Conference on Learning Representations*, 2024.

R. Munos. Error bounds for approximate policy iteration. In *International Conference on Machine Learning*, pages 560–567, 2003.

R. Munos. Error bounds for approximate value iteration. In *AAAI Conference on Artificial Intelligence*, pages 1006–1011, 2005.

R. Munos. Performance bounds in L_p-norm for approximate value iteration. *SIAM Journal on Control and Optimization*, 46(2):541–561, 2007.

R. Munos and C. Szepesvári. Finite-time bounds for fitted value iteration. *Journal of Machine Learning Research*, 9:815–857, 2008.

H. Nam, O. Gottesman, A. Zhang, D. Foster, E. Brunskill, and L. Ungar. Efficient RL for optimizing conversation level outcomes with an LLM-based tutor. *arXiv preprint arXiv:2507.16252*, 2025.

OpenAI. Introducing ChatGPT. https://openai.com/index/chatgpt/, 2022. Blog post, November 30, 2022.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022.

A. Pal, D. Karkhanis, S. Dooley, M. Roberts, S. Naidu, and C. White. Smaug: Fixing failure modes of preference optimisation with DPO-positive. *arXiv preprint arXiv:2402.13228*, 2024.

R. Park, R. Rafailov, S. Ermon, and C. Finn. Disentangling length from quality in direct preference optimization. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 4998–5017. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-acl.297.

W. B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley & Sons, 2nd edition, 2011.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.

Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, et al. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*, 2023.

A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. Technical report, OpenAI, 2018.

R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023.

P. Rajasekaran, E. Dixon, C. Ryan, and J. Hadfield. Effective context engineering for AI agents. Anthropic Engineering Blog, 2025. https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

N. Razin, S. Malladi, A. Bhaskar, D. Chen, S. Arora, and B. Hanin. Unintentional unalignment: Likelihood displacement in direct preference optimization. *arXiv preprint arXiv:2410.08847*, 2024.

A. Samanta, A. Magesh, R. Wu, A. Jain, Y. Yu, D. Jiang, B. Vidolov, P. Sajda, Y. Efroni, and K. Hassani. Self-improvement of language models by post-training on multi-agent debate. *arXiv preprint arXiv:2509.15172*, 2025.

A. Samanta, A. Magesh, A. Jain, Y. Yu, D. Jiang, K. Asadi, K. Hassani, P. Sajda, J. Bhandari, and Y. Efroni. Credit assignment with resets in language model reasoning. *arXiv preprint arXiv:2605.25507*, 2026.

B. Scherrer, M. Ghavamzadeh, V. Gabillon, B. Lesner, and M. Geist. Approximate modified policy iteration and its application to the game of Tetris. *Journal of Machine Learning Research*, 16: 1629–1676, 2015.

J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.

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

J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations*, 2016.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pages 1715–1725, 2016.

O. Shaikh, S. Sapkota, S. Rizvi, E. Horvitz, J. S. Park, D. Yang, and M. S. Bernstein. Creating general user models from computer use. *arXiv preprint arXiv:2505.10831*, 2025.

L. Shani, A. Rosenberg, A. Cassel, O. Lang, D. Calandriello, A. Zipori, H. Noga, O. Keller, B. Piot, I. Szpektor, A. Hassidim, Y. Matias, and R. Munos. Multi-turn reinforcement learning with preference human feedback. In *Advances in Neural Information Processing Systems*, volume 37, pages 118953–118993, 2024.

Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

W. Shi, M. Yuan, J. Wu, Q. Wang, and F. Feng. Direct multi-turn preference optimization for language agents. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 2312–2324, 2024.

Y. Song, G. Swamy, A. Singh, J. A. Bagnell, and W. Sun. The importance of online data: Understanding preference fine-tuning via coverage. *arXiv preprint arXiv:2406.01462*, 2024.

N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems*, volume 33, pages 3008–3021, 2020.

R. S. Sutton. Dyna, an integrated architecture for learning, planning, and reacting. *ACM SIGART Bulletin*, 2(4):160–163, 1991.

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018.

C. Szepesvári and R. Munos. Finite time bounds for sampling based fitted value iteration. In *International Conference on Machine Learning*, pages 880–887, 2005.

Y. Tang and R. Munos. On a few pitfalls in KL divergence gradient estimation for RL. *arXiv preprint arXiv:2506.09477*, 2025.

Y. Tang, D. Z. Guo, Z. Zheng, D. Calandriello, Y. Cao, E. Tarassov, R. Munos, B. Á. Pires, M. Valko, Y. Cheng, and W. Dabney. Understanding the performance gap between online and offline alignment algorithms. *arXiv preprint arXiv:2405.08448*, 2024.

G. Tennenholtz, O. Meshi, A. Globerson, U. Shalit, J. Jeong, and C. Boutilier. Controllable user simulation. *arXiv preprint arXiv:2605.11519*, 2026.

The Microsoft AI Team. MAI-Thinking-1: Building a hill-climbing machine. Technical report, Microsoft AI, 2026. URL https://microsoft.ai/pdf/mai-thinking-1.pdf.

B. Van Roy. Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, 31(2):234–244, 2006.

M. Vandergrift, E. Elelimy, and M. White. Position: RL researchers need to distinguish between solving simulators and using simulators as a proxy. In *Proceedings of the 43rd International Conference on Machine Learning*, volume 306 of *Proceedings of Machine Learning Research*, 2026.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

Z. Wang, K. Ramnath, B. Bi, S. K. Pentyala, S. Chaudhuri, S. Mehrotra, Z. J. Zhu, X.-B. Mao, S. Asur, and N. C. Cheng. Reinforcement learning for LLM post-training: A survey. *arXiv preprint arXiv:2407.16216*, 2024.

Z. Wang, K. Wang, Q. Wang, P. Zhang, L. Li, Z. Yang, X. Jin, K. Yu, M. N. Nguyen, L. Liu, E. Gottlieb, Y. Lu, K. Cho, J. Wu, L. Fei-Fei, L. Wang, Y. Choi, and M. Li. RAGEN: Understanding self-evolution in LLM agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

Q. Wei, S. Zeng, C. Li, W. Brown, O. Frunza, W. Deng, A. Schneider, Y. Nevmyvaka, Y. K. Zhao, A. Garcia, and M. Hong. Reinforcing multi-turn reasoning in LLM agents via turn-level reward design. *arXiv preprint arXiv:2505.11821*, 2025a.

Z. Wei, W. Yao, Y. Liu, W. Zhang, Q. Lu, L. Qiu, C. Yu, P. Xu, C. Zhang, B. Yin, H. Yun, and L. Li. WebAgent-R1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7920–7939, 2025b.

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

S. Wu, E. Choi, A. Khatua, Z. Wang, J. He-Yueya, T. C. Weerasooriya, W. Wei, D. Yang, J. Leskovec, and J. Zou. HumanLM: Simulating users with state alignment beats response imitation. *arXiv preprint arXiv:2603.03303*, 2026.

T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, et al. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024.

Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, W. Dai, T. Fan, G. Liu, L. Liu, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

Y. Yue, J. Broder, R. Kleinberg, and T. Joachims. The k-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5):1538–1556, 2012.

Z.ai. GLM-5.2: Built for long-horizon tasks. https://z.ai/blog/glm-5.2, 2026. Blog post, June 16, 2026.

Y. Zhang, C. Ye, S. Jin, C. Yu, W. Xiong, S. Sahu, and N. Jiang. Rethinking importance sampling in LLM policy optimization: A cumulative token perspective. *arXiv preprint arXiv:2605.07331*, 2026.

C. Zheng, S. Liu, M. Li, X.-H. Chen, B. Yu, C. Gao, K. Dang, Y. Liu, R. Men, A. Yang, J. Zhou, and J. Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

L. Zheng, W.-L. Chiang, Y. Sheng, S. Zhuang, Z. Wu, Y. Zhuang, Z. Lin, Z. Li, D. Li, E. P. Xing, et al. Judging LLM-as-a-judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623, 2023a.

R. Zheng, S. Dou, S. Gao, Y. Hua, W. Shen, B. Wang, Y. Liu, S. Jin, Q. Liu, Y. Zhou, et al. Secrets of RLHF in large language models Part I: PPO. *arXiv preprint arXiv:2307.04964*, 2023b.

S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig. WebArena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

Y. Zhou, A. Zanette, J. Pan, S. Levine, and A. Kumar. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62178–62209. PMLR, 2024.

S. Zhu, C. Yu, R. Yang, Z. Liu, J. Hu, Q. Chen, and Y. Zhang. GAGPO: Generalized advantage grouped policy optimization. *arXiv preprint arXiv:2605.13217*, 2026.

D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*, 2019.

Y. Zuo, K. Zhang, L. Sheng, S. Qu, G. Cui, X. Zhu, H. Li, Y. Zhang, X. Long, E. Hua, B. Qi, Y. Sun, Z. Ma, L. Yuan, N. Ding, and B. Zhou. TTRL: Test-time reinforcement learning. *arXiv preprint arXiv:2504.16084*, 2025.