RL without Tears: An Introduction in the Era of LLMs

Chenglong Wang WANGCHENGLONG@MAIL.NEU.EDU.CN
Northeastern University, China

Hang Zhou STCEUM@GMAIL.COM
Northeastern University, China

Tongran Liu LIUTR@PSYCH.AC.CN
Institute of Psychology, Chinese Academy of Sciences, China

Jingbo Zhu ZHUJINGBO@MAIL.NEU.EDU.CN
Northeastern University, China

Tong Xiao XIAOTONG@MAIL.NEU.EDU.CN
Northeastern University, China

GitHub Online Website

Abstract

Reinforcement learning (RL) has become a powerful and versatile paradigm in the era of large language models (LLMs). It is widely used to align models with human preferences and improve their reasoning capabilities. More recently, with the rapid rise of LLM-based agents, RL has become increasingly important for enabling models to learn from experience and build more capable and adaptive intelligent systems. Despite its growing importance, RL remains difficult for many LLM researchers to approach. Traditional RL literature often explains key concepts through robotics and control problems, making them less intuitive in the context of LLM training. To bridge this gap, this paper provides a comprehensive introduction to RL through LLM training examples. We first introduce the fundamental concepts of RL and explain key algorithms through a concrete example of training an LLM, including policy gradients, advantage estimation, importance sampling, and reward modeling. We then discuss recent advances in RL for LLMs, including improved reward construction, efficient training, and policy optimization. We further extend the discussion to reasoning models, LLM-based agents, and multimodal models. We show how RL can enhance reasoning, train agents through environment interaction, and optimize multimodal understanding and generation. Finally, we summarize promising future directions and commonly used systems and datasets. We hope this paper provides an accessible introduction to RL and helps LLM researchers better understand and apply RL techniques in modern foundation models.

Contents

1 Introduction

RL is an interdisciplinary field, and has its origins in both psychology and optimal control. Over the years, the concept has been further developed and formalized within the realms of artificial intelligence and machine learning. The basic idea is that an agent learns to make decisions by receiving feedback on its actions from the environment. This approach is very general and applies to a wide range of problems, from playing complex games like chess and Go to controlling autonomous vehicles.

A recent breakthrough is the large-scale application of RL in the field of natural language processing (NLP), in particular in the development of LLMs. For example, RL has been widely considered an effective way of aligning pre-trained LLMs with human preferences and values (Christiano et al., 2017). One such method, called reinforcement learning from human feedback (RLHF), forms the basis for the development of advanced LLMs like OpenAI's GPT-4. More recently, RL has shown great promise in enabling LLMs to perform complex reasoning (OpenAI, 2024; Deepseek, 2025; Team et al., 2025a). As a result, interest in RL for LLMs has exploded. Numerous conference papers now discuss RL in LLMs, with even more appearing in the past few months. The research community is highly enthusiastic, and many in the field are anticipating a new turning point where large-scale RL algorithms could further advance AI. This trend is further strengthened by the rise of LLM-based agents, where models must learn from interaction and experience to make effective decisions in dynamic environments. From this perspective, RL is becoming a key technique for building more capable and adaptive intelligent systems. This view closely echoes the "Reward Is Enough" hypothesis proposed by Silver et al. (2021):

> "Intelligence, and its associated abilities, can be understood as subserving the maximisation of reward by an agent acting in its environment."

This hypothesis highlights why RL is particularly relevant to the next stage of LLM development: instead of learning only from static supervision, models can acquire increasingly complex capabilities through large-scale experience and rewards.

Historically, RL has played a relatively limited role in mainstream NLP, where supervised learning has long been the dominant paradigm. Although applying RL to LLM training seems a natural choice for machine learning researchers, it introduces many new concepts to the NLP community, which has traditionally been dominated by supervised learning methodologies. As NLP researchers, when we attended presentations on LLMs, we often heard statements such as "use a value function to estimate the expected cumulative reward", "learn the policy with PPO", or simply "we need to train a reward model". These techniques were often presented as if they required little explanation. Yet, for researchers trained primarily in supervised learning, the mathematical formulations of RL, with their many expectations, value functions, and advantages, can be difficult to follow. This gap becomes increasingly important in LLM training, where understanding RL is no longer a specialized topic, but is becoming essential for following and developing modern AI systems.

It is natural to learn RL from standard references, which appears straightforward at first. However, common RL textbooks, such as Sutton & Barto (2018)'s book, are mostly based on classic robotics or control problems. This makes it difficult to align the concepts of RL with those of LLMs. On the other hand, most LLM literature lacks an in-depth discussion of RL techniques, often omitting related details. As a result, we find ourselves in an awkward middle ground between RL and LLMs, struggling to bridge the two fields.

The aim of this paper is to provide a comprehensive introduction to RL from the perspective of LLM training. We begin by introducing the basic concepts and algorithms of RL using terminology familiar to LLM researchers, and illustrate them through concrete LLM training examples. We then discuss recent advances in RL for LLMs, including improved reward modeling, advantage estimation, training efficiency, and policy optimization. Beyond standard LLM alignment, we further introduce how RL is applied to enhance reasoning capabilities, support long-horizon decision-making in LLM-based agents, and optimize multimodal understanding and generation models. Finally, we summarize promising future directions and provide an overview of commonly used systems and datasets.

2 Preliminary

2.1 Fundamentals of LLMs

LLMs have become the foundation of modern natural language processing. Their success largely follows a simple but powerful paradigm: first learn general language capabilities from large-scale text corpora, and then adapt these capabilities to specific tasks through prompting or supervised fine-tuning. At inference time, an LLM generates an output by predicting tokens sequentially according to the given input and previously generated tokens. This paradigm has substantially changed how NLP systems are developed, replacing many task-specific models with a unified foundation model that can support a wide range of language understanding and generation tasks.

Understanding these basic mechanisms is important before introducing RL for LLMs. Many RL concepts can be naturally connected to the standard generation process of an LLM. The input and previously generated tokens define the current context, the next-token distribution determines the model's possible actions, and the complete output forms a sequence of decisions that can be evaluated by a reward signal. In this subsection, we briefly review the fundamentals of LLMs and introduce the key concepts and notations used throughout this paper. We focus only on the concepts necessary for understanding the subsequent discussion of RL, rather than providing a comprehensive review of LLM techniques and their recent developments. Interested readers are referred to Xiao & Zhu (2025) for a more systematic introduction to LLMs.

In this paper, we mainly focus on generative LLMs based on decoder-only Transformers (Vaswani et al., 2017). Let $\mathcal{V}$ denote the vocabulary of tokens. A token is the basic unit processed by an LLM. It can be a word, a subword, a punctuation mark, or another text fragment produced by the tokenizer. Before a raw text is input into an LLM, it is first converted into a sequence of tokens from $\mathcal{V}$. Therefore, throughout this paper, all inputs and outputs of an LLM are represented as token sequences. Given a token sequence $\mathbf{z} = z_1 ... z_N$, a language model parameterized by θ estimates the probability of the sequence by factorizing it from left to right:

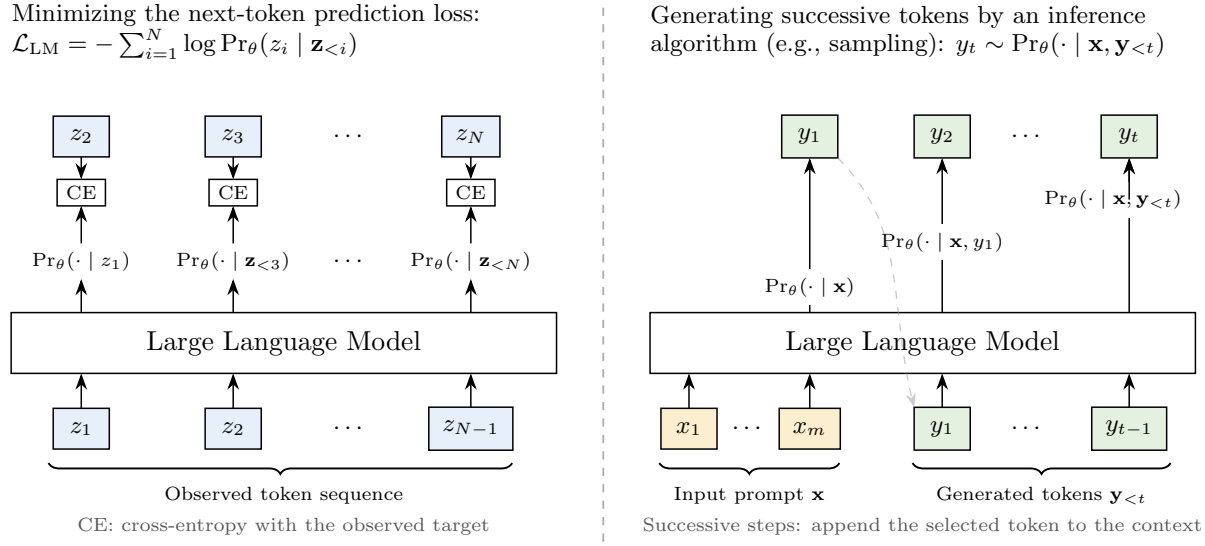
$$\Pr_\theta(\mathbf{z}) = \prod_{i=1}^{N} \Pr_\theta(z_i|\mathbf{z}_{<i}) \tag{1}$$

$$\log \Pr_\theta(\mathbf{z}) = \sum_{i=1}^{N} \log \Pr_\theta(z_i|\mathbf{z}_{<i}) \tag{2}$$

where $\mathbf{z}_{<i} = z_1 ... z_{i-1}$ denotes the tokens before z_i. A decoder-only Transformer implements this conditional distribution with causal self-attention, so that each position can only depend on previous tokens. The model then produces a probability distribution over $\mathcal{V}$, denoted by $\Pr_\theta(\cdot|\mathbf{z}_{<i})$. This next-token prediction view is the central interface through which LLMs are trained, prompted, fine-tuned, and later optimized by RL. Figure 1 illustrates this process together with prompt-based autoregressive generation. During training, the predicted distributions are matched to the observed target tokens. During inference, each selected token is appended to the context and used to condition the next prediction.

2.1.1 Pre-training

Pre-training is the first stage of building most LLMs. The goal is to train the model on large-scale unlabeled text so that it can acquire general knowledge about languages and the world. In generative LLMs, this is usually achieved through self-supervised next-token prediction: the model observes the preceding tokens and learns to predict the next token. Although no human-annotated labels are required, each token in the text can naturally serve as a supervision signal for predicting itself from its left context.



(a) Next-token prediction during training

(b) Autoregressive generation with prompting

Figure 1: Language model training and autoregressive generation. (a) In next-token prediction, an LLM predicts a distribution over the next token from the preceding context; the predicted distribution is compared with the observed target token using cross-entropy. (b) In autoregressive generation, an LLM selects the next token from the distribution conditioned on the prompt and previously generated tokens. The selected token is appended to the context and then conditions the next prediction step.

Let $\mathcal{S}_{\text{pre}}$ denote the pre-training corpus, where each sample is a token sequence $\mathbf{z} = z_1 ... z_N$. The pre-training objective is to maximize the log-likelihood of all tokens in the corpus:

$$\hat{\theta} \quad = \quad \arg\max_{\theta} \sum_{\mathbf{z} \in \mathcal{S}_{\text{pre}}} \sum_{i=1}^{N} \log \Pr_\theta(z_i | \mathbf{z}_{<i}) \tag{3}$$

where $\hat{\theta}$ denotes the optimized pre-trained parameters. Equivalently, this objective can be viewed as minimizing the cross-entropy loss between the observed next token and the model-predicted distribution.

After pre-training, the LLM learns to model natural language and acquires rich linguistic and factual knowledge from large-scale text. Given a preceding context, it can predict plausible continuations and generate fluent text. For example, given the prefix "The capital of France is", a pre-trained LLM may assign a high probability to the next token "Paris". Similarly, given a longer context such as "To solve this equation, we first move all terms to one side", the model can continue the text with a linguistically and semantically plausible sequence. These examples reflect the core capability learned during pre-training: predicting likely continuations from the preceding context.

However, pre-training primarily teaches the model to perform language modeling rather than to explicitly solve user-specified tasks. In particular, the model is not directly trained to interpret instructions or produce outputs in a desired format. As a result, simply providing a task description to a pre-trained LLM does not necessarily lead to the intended behavior. This limitation motivates us to use additional adaptation approaches to better equip pre-trained LLMs for downstream tasks.

2.1.2 Prompting

Prompting is a simple and lightweight way to adapt an LLM to different tasks without updating its parameters. A *prompt* is the input text used to guide the model toward a desired task or output. It may

contain an instruction, user-provided content, output requirements, or demonstrations. For example, if we want an LLM to act as a homework assistant and answer a student's question, we can provide the following prompt:

> You are a homework assistant helping students improve their problem-solving skills.
> Give me three tips to improve my accuracy in solving math problems.
> Please make the response clear, practical, and easy to follow.
>
> _____

In this example, the prompt specifies both the task and the desired response style. The LLM then generates the answer by continuing the input sequence. Prompts can also be constructed from templates. A *prompt template* contains placeholders that can be replaced with task-specific information. For example, we can use the following template for a homework assistant:

> You are a homework assistant helping students improve their problem-solving skills.
> Give me three tips to improve my accuracy in solving {*subject*} problems.
> Please make the response clear, practical, and easy to follow.
>
> _____

If we set {*subject*} to "math", the template becomes the prompt used above. In this way, users can construct different prompts by changing the placeholder while keeping the main task description unchanged.

Another important concept related to prompting is in-context learning. When prompting an LLM, we can add demonstrations to the context and let the model infer the desired input-output pattern from these examples. For instance, we can show the model how to answer similar student questions before asking it to respond to a new one:

> *Input:* Give me two tips to improve my English writing.
>
> *Output:* 1. Read well-written essays regularly.
>
> 2. Revise your sentences to make them clear.
>
> *Input:* Give me three tips to improve my accuracy in solving math problems.
>
> *Output:* _____

When a single demonstration is provided, this setting is commonly called *one-shot* prompting. When several demonstrations are included, it is referred to as *few-shot* prompting. In contrast, prompting without any demonstration is usually called *zero-shot* prompting.

In-context learning can also be combined with *chain-of-thought* (CoT) prompting (Wei et al., 2022b). The basic idea is to encourage the model to generate intermediate reasoning steps before producing the final answer. This can be achieved either by explicitly asking the model to reason step by step or by providing demonstrations that contain both reasoning traces and answers. For example:

Input: A student solves 8 math problems and gets 6 correct. What is the accuracy? Please reason step by step.

Output: The student gets 6 out of 8 problems correct. The accuracy is therefore $6/8 = 0.75$, or 75%.

Input: A student solves 20 math problems and gets 17 correct. What is the accuracy? Please reason step by step.

Output: _____

Prompting adapts an LLM by changing the context used for generation. Therefore, prompt quality can strongly affect model performance. Even for the same task, we can write the prompt in many different ways. For example, "Give me two tips to improve my English writing." can also be written as "What are two simple ways to improve my English writing skills?". The two prompts express nearly the same intent, but they may lead to noticeably different outputs. In practice, we often refine prompts through repeated trial and error for a given LLM. More advanced methods automate this process and search for better prompts using techniques such as RL (Deng et al., 2022) or evolutionary algorithms (Guo et al., 2024).

2.1.3 Supervised Fine-Tuning

Although pre-trained LLMs possess a vast amount of general knowledge, they are typically limited to tasks related to language modeling. Our ultimate goal is to enable them to perform various specific tasks, such as summarization, question answering, and machine translation. One straightforward method to achieve this goal is supervised fine-tuning (SFT), in which the pre-trained LLM is further trained on a dataset comprising task-specific inputs (i.e., instruction + user input)[1] paired with their expected outputs (Ouyang et al., 2022; Wei et al., 2022a). Table 1 gives several examples of SFT data for different tasks.

Specifically, let $\mathbf{x} = x_1 ... x_m$ be an input and $\mathbf{y} = y_1 ... y_T$ be the corresponding output. In SFT, we aim to maximize the probability of the output $\mathbf{y}$ given the input $\mathbf{x}$. Consider an LLM with pre-trained parameters $\hat{\theta}$. The fine-tuning objective can then be formulated as:

$$\tilde{\theta} \quad = \quad \arg\max_{\hat{\theta}^+} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{S}} \log \Pr_{\hat{\theta}^+}(\mathbf{y}|\mathbf{x}) \tag{4}$$

where $\Pr(\cdot)$ denotes the probability distribution, $\mathcal{S}$ denotes the labeled data, $\tilde{\theta}$ denotes the parameters optimized via SFT, and $\hat{\theta}^+$ represents an adjustment to $\hat{\theta}$. Here, we will omit the superscript + and use θ to represent $\hat{\theta}^+$ to keep the notation uncluttered. However, the reader should remember that the fine-tuning starts from the pre-trained parameters rather than randomly initialized ones.

The objective function $\log \Pr_\theta(\mathbf{y}|\mathbf{x})$ is computed by summing the log-probabilities of the tokens in $\mathbf{y}$, conditional on the input $\mathbf{x}$ and all the previous output tokens $\mathbf{y}_{<t}$:

$$\log \Pr_\theta(\mathbf{y}|\mathbf{x}) \quad = \quad \sum_{t=1}^{T} \log \Pr_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t}) \tag{5}$$

This formulation is equivalent to minimizing the cross-entropy loss over the output segment. In practice, the input tokens in $\mathbf{x}$ are used as the context, while the loss is usually computed only on the output tokens

[1] In this paper, the *instruction* represents the description of a specific task, e.g., "Summarize the following article"; the *user input* represents the extra content added to the instruction, e.g., "Article: In recent years, solar energy has seen a significant increase in the amount of energy available to the public. Solar energy has seen unprecedented growth, becoming the fastest-growing ..."

x (Instruction + User Input)	y (Output)
Summarization: Summarize the following article. Article: Solar energy has experienced rapid growth in recent years and has become one of the fastest-growing sources of renewable energy.	Solar energy has grown rapidly in recent years and has become a major renewable energy source.
Question Answering: Answer the following question. Question: What is the capital of France?	Paris.
Classification: Classify the following message as spam or not spam. Message: Congratulations! You have won a $500 gift card. Click here to claim it.	Spam.
Machine Translation: Translate the following sentence from English to Chinese. Sentence: RL is widely used to train large language models.	强化学习被广泛用于训练大语言模型。

Table 1: Examples of SFT data for different downstream tasks.

in $\mathbf{y}$. In this way, SFT teaches the model not only what knowledge to express, but also how to respond in a desired instruction-following format. In the following section, we refer to the LLM after SFT as the "SFT LLM" for short.

2.1.4 Inference

Inference is the process of applying a trained LLM to generate outputs for new inputs. Given an input $\mathbf{x}$, the model first predicts a next-token distribution $\Pr_\theta(\cdot|\mathbf{x})$ and selects a token y_1 according to a decoding strategy. The selected token is then appended to the context, after which the model predicts the next token according to $\Pr_\theta(\cdot|\mathbf{x}, y_1)$. This process is repeated until a stopping criterion is reached, such as generating an end-of-sequence token or reaching the maximum generation length. We refer to this sequential generation process as *autoregressive generation*.

Different decoding strategies determine how the next token is selected from the model-predicted distribution. Commonly used strategies include:

- **Greedy Decoding.** Greedy decoding is one of the simplest decoding strategies. At each generation step, it selects the token with the highest probability under the model. Consider the probability of the generated sequence up to timestep t:

$$\log \Pr_\theta(\mathbf{y}_{\le t}|\mathbf{x}) \quad = \quad \log \Pr_\theta(\mathbf{y}_{<t}|\mathbf{x}) + \log \Pr_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t}) \tag{6}$$

  where $\mathbf{y}_{<t} = y_1 ... y_{t-1}$ denotes the previously generated tokens. Since the first term is fixed once $\mathbf{y}_{<t}$ has been generated, greedy decoding selects

$$y_t^{\text{greedy}} \quad = \quad \arg\max_{w \in \mathcal{V}} \Pr_\theta(w|\mathbf{x}, \mathbf{y}_{<t}) \tag{7}$$

  The selected token is appended to the current prefix, and the same procedure is repeated at the next timestep. Greedy decoding is deterministic and computationally efficient, but locally selecting the most probable token does not necessarily produce the most probable complete sequence.

- **Beam Search.** Beam search extends greedy decoding by maintaining multiple candidate sequences during generation. Instead of keeping only the single best prefix at each timestep, it retains the top B candidates, where B is referred to as the *beam size*. Let $\mathcal{B}_{t-1}$ denote the set of candidate

prefixes retained at timestep $t-1$. Each prefix $\mathbf{y}_{<t}$ is extended with possible next tokens, and the resulting candidates are ranked according to their accumulated sequence scores:

$$\begin{aligned} s(\mathbf{y}_{\leq t}) &= \log \Pr_\theta(\mathbf{y}_{\leq t}|\mathbf{x}) \\ &= s(\mathbf{y}_{<t}) + \log \Pr_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t}) \end{aligned} \tag{8}$$

The beam for the next timestep is then constructed by retaining the B highest-scoring candidates:

$$\mathcal{B}_t = \text{TopB}\left\{(\mathbf{y}_{<t}, w) : \mathbf{y}_{<t} \in \mathcal{B}_{t-1}, w \in \mathcal{V}\right\} \tag{9}$$

By exploring multiple prefixes simultaneously, beam search can avoid some locally optimal decisions made by greedy decoding and often produces higher-probability sequences. However, it requires more computation and memory as the beam size increases.

- **Sampling.** Sampling is widely used in modern LLM inference, particularly when diverse outputs are desired. Instead of always selecting the most probable token or maintaining a fixed set of candidate sequences, the next token is sampled from the model-predicted distribution:

$$y_t \sim \Pr_\theta\left(\cdot|\mathbf{x}, \mathbf{y}_{<t}\right) \tag{10}$$

This makes generation stochastic, allowing the same input to produce different outputs. In practice, the sampling distribution is often adjusted using a temperature parameter T. Given the model logit l_w for token w, the temperature-adjusted probability is

$$\Pr_\theta^{(T)}(w|\mathbf{x}, \mathbf{y}_{<t}) = \frac{\exp(l_w/T)}{\sum_{v\in\mathcal{V}} \exp(l_v/T)} \tag{11}$$

A smaller temperature sharpens the distribution and favors high-probability tokens, while a larger temperature produces a flatter distribution and increases diversity. Sampling is also commonly combined with *top-k* or *top-p* filtering. Top-k sampling restricts the candidate set to the k tokens with the highest probabilities (Fan et al., 2018), whereas top-p sampling retains the smallest set of tokens whose cumulative probability reaches a threshold p (Holtzman et al., 2019). The next token is then sampled from the renormalized distribution over the retained candidates. These techniques help reduce the probability of selecting unlikely tokens while preserving diversity.

Regardless of the decoding strategy, LLM generation can be viewed as a sequence of token-level decisions. This view naturally connects LLMs with RL. At timestep t, the current context $(\mathbf{x}, \mathbf{y}_{<t})$ can be treated as the state, the next token y_t as the action, and the next-token distribution $\Pr_\theta(\cdot|\mathbf{x}, \mathbf{y}_{<t})$ as the policy. The complete output $\mathbf{y}$ then corresponds to a trajectory whose quality can be evaluated by a reward signal. In particular, sampling closely resembles the stochastic policy sampling process used in RL. Based on this connection, the following subsection reviews the fundamentals of RL, and Section 3 further shows how RL can be used to optimize an SFT LLM.

2.2 Fundamentals of RL

RL, supported by a well-established theoretical framework, has been widely applied across a broad range of domains. In particular, with the rapid advancement of LLMs, RL has become a standard approach in the post-training stage. Before using LLM training examples to explain RL algorithms, we first provide a brief overview of deep reinforcement learning in this subsection. We then introduce the general formulation of RL, along with key terminologies and notations that are essential for understanding RL.

2.2.1 Markov Decision Process

The RL formulation for LLMs is illustrated in Figure 2. An RL system primarily consists of two components: an agent and an environment. At each timestep t, the agent observes a state s_t from the environment and selects an action a_t according to a policy π, which is typically parameterized by a neural network. After

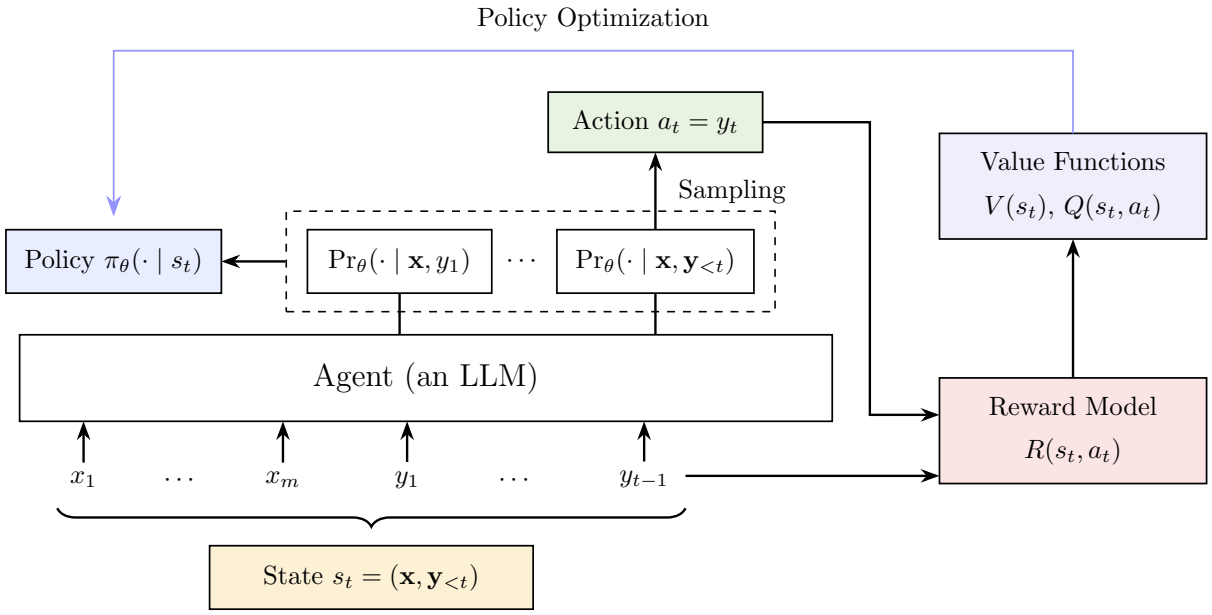


Figure 2: A schematic illustration of RL for LLMs.

executing the action, the environment transitions to a new state s_{t+1} and returns a reward r_t corresponding to the taken action.

This interaction process is commonly modeled as a Markov Decision Process (MDP), where the agent interacts with the environment over discrete timesteps (Sutton & Barto, 2018). A sequence of states and actions forms a trajectory, denoted as $\tau = (s_0, a_0, s_1, a_1, \cdots, s_{H-1}, a_{H-1})$, where H represents the trajectory length. Each trajectory accumulates rewards from the environment.

The objective of RL is to learn an optimal policy π^* that maximizes the expected cumulative reward over trajectories, which can be formulated as:

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t=0}^{H-1} r_t \mid \pi\right] \tag{12}$$

In practice, the agent improves its policy through repeated interactions with the environment by observing states and receiving feedback in the form of rewards. Each such interaction sequence constitutes a trajectory τ. The process of generating one or more trajectories under a given policy is commonly referred to as *sampling* in the RL literature.

2.2.2 Key Elements

The MDP formulation provides a compact way to describe RL, but its terminology can feel distant from the language modeling setting at first glance. To make the connection more explicit, Table 2 summarizes the core elements of RL and describes how each one can be interpreted in LLM training. This tabular view is intended to serve as a bridge between the standard RL notation introduced above and the token-level generation process discussed earlier.

With these correspondences in place, we can describe RL algorithms using familiar language-modeling objects such as contexts, tokens, policies, and reward signals. In contrast to conventional RL literature, which often presents algorithms through robotics or control examples, *we adopt an NLP perspective to introduce RL in the following section.*

Element	Interpretation
Agent	The learner or decision-maker in RL. In the context of LLMs, the agent corresponds to the language model itself, which generates tokens sequentially and updates its behavior based on feedback signals.
Environment	Everything external to the agent with which it interacts. Unlike traditional RL settings that involve physical or simulated environments, the environment in LLM-based RL is typically abstract, consisting of the training framework that provides feedback (e.g., reward models, human annotations, or evaluation metrics) for generated outputs.
State (s)	A state represents the current situation of the environment. For language modeling, the state at timestep t can be defined as the sequence of observed tokens up to that point, i.e., the context used to predict the next token. Formally, the state can be represented as $s_t = (\mathbf{x}, \mathbf{y}_{<t})$, where $\mathbf{x}$ denotes the input prompt and $\mathbf{y}_{<t}$ denotes the previously generated tokens.
Action (a)	An action corresponds to a decision made by the agent. In LLMs, actions are naturally defined as selecting the next token from the vocabulary, i.e., $a = y_t$.
Reward (r)	The reward provides feedback from the environment to evaluate the quality of an action. In general, the reward function can be defined as $r(s, a, s')$, representing the feedback received when the agent transitions from state s to s' by taking action a. At timestep t, this can be written as $r_t = r(s_t, a_t, s_{t+1})$. In deterministic settings, where the next state is uniquely determined by (s_t, a_t), the reward can be simplified as $r(s_t, a_t)$.
Policy (π)	The policy defines the agent's behavior, i.e., the probability of taking an action given a state. For LLMs, the policy corresponds to the conditional probability distribution over the next token given the context: $$\pi_\theta(a_t \mid s_t) = \Pr_\theta(y_t \mid \mathbf{x}, \mathbf{y}_{<t})$$ where $a_t = y_t$ and $s_t = (\mathbf{x}, \mathbf{y}_{<t})$. Under this formulation, an LLM can be naturally interpreted as a parameterized policy.
Value Function (V and Q)	The value function estimates the expected cumulative reward when following a policy. The state-value function $V(s)$ measures the expected discounted return starting from state s: $$V(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, \pi\right]$$ where $\gamma \in [0, 1]$ is the discount factor. The action-value function $Q(s, a)$ further conditions on the initial action: $$Q(s, a) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \middle| s_0 = s, a_0 = a, \pi\right]$$

Table 2: Key elements of RL and their interpretations

3 Understanding RL in LLM Training

We continue with a practical example of using an LLM as a homework assistant. In this context, we want to improve the capability of an SFT LLM to handle education-related inputs more effectively. Suppose we have a homework assistant powered by an SFT LLM, and a student types the input "Give me three tips to improve my accuracy in solving math problems." A SFT LLM typically generates a short output, such as

There are three tips for improving accuracy in solving math problems:

1. Practice regularly.
2. Understand the concepts.
3. Double-check your work.

Although this output adheres to the given input, it is very short and not sufficiently detailed or comprehensive for this scenario. In practice, the "short" feature in the generated output stems from the training approach of the SFT LLM. During SFT, the LLM learns from a large set of labeled samples that typically contain concise and factual answers to common inputs. These samples guide the LLM to prioritize brevity and clarity, so it often generates short outputs, like the one shown above. Of course, we could annotate enough additional data to fine-tune the LLM further and adjust it to generate the desired outputs. However, this approach is limited in its ability to scale. For example, in this scenario, the input involves a variety of potential answers, and the expectations of the student can be pretty diverse. Therefore, describing the "ideal" output would require immense annotation effort. Consequently, collecting or annotating fine-tuning data is not as straightforward as it is with SFT, particularly when attempting to cover the breadth of possible student inputs and outputs. Instead, we can use RL to enable the model to discern outputs that better align with human preferences, such as generating more detailed and comprehensive content that not only adheres to the given input but also meets the expectations of the student in this scenario.

In the following sections, we use a specific LLM training example to explain how RL works, where the LLM is trained to generate a longer output in the context of a homework assistant. Throughout this example, we introduce key RL algorithms and their improvements.

3.1 Policy Gradient

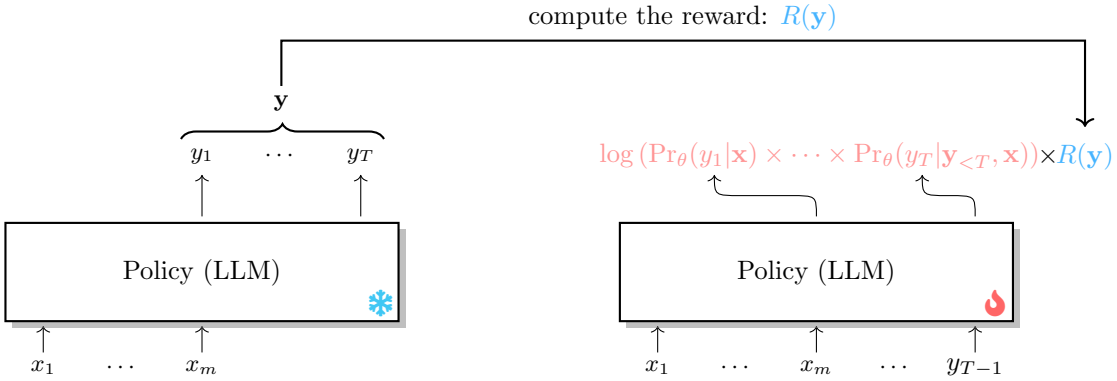
In this section, we aim to lengthen the LLM output for the "give me three tips" input using a classical RL algorithm called **policy gradient**. To define our optimization objective, let $R(\cdot)$ be the reward function that describes the goal the algorithm aims to optimize. In this scenario, the reward is designed to align the output of the LLM with a specific behavior, that is, generating longer outputs. Therefore, we define the reward function based on a length-related metric. More specifically, the reward can be proportional to the length of the output, that is, longer outputs can receive higher rewards:

$$R(\mathbf{y}) = \frac{\text{Length}(\mathbf{y})}{10} \tag{13}$$

where $\text{Length}(\cdot)$ indicates the length of the given output. We can use the number of tokens in the output as the length for simplicity. This allows us to assign an immediate reward r_t for the t-th token generated by the LLM, which is $\frac{1}{10}$.

Next, we apply RL to encourage the LLM to generate longer outputs based on the reward function. First, let us review the optimization objective of RL: learning a policy that maximizes the agent's cumulative reward (or return) over the long term. In this case, the agent can be defined as the LLM itself, and the policy can be defined as the probability distribution over the tokens the LLM predicts, given the preceding context tokens, i.e., $\Pr_\theta(\cdot|\mathbf{x})$[2].

[2]To maintain consistency with notations in RL, this policy is also often represented as $\pi_\theta(\cdot|\mathbf{x})$ in much of the related literature. Here, from the perspective of LLM research, we opt to use the probability notation $\Pr_\theta(\cdot|\mathbf{x})$, which is more familiar to researchers in this field.



(a) sample an output $\mathbf{y}$ (b) train the LLM via policy gradient

Figure 3: Illustration of training an LLM using policy gradient. The central component is an LLM. Given an input $\mathbf{x} = x_1...x_m$, we sample an output $\mathbf{y}$ from the LLM. Note that gradients are not computed during the sampling to prevent extensive memory consumption because the sampling is performed via an autoregressive mode. Instead, by leveraging the parallel computation capability of the LLM, similar to SFT, the sampled output $\mathbf{y}$ is used for a subsequent forward pass to compute the gradients produced during the generation of each token.

The optimization objective can be given by

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\Omega} \Pr_\theta(\mathbf{y}|\mathbf{x}) R(\mathbf{y})\right] \\ &= \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\Omega} \Pr_\theta(\mathbf{y}|\mathbf{x}) \sum_{t=1}^{T} r_t\right] \end{aligned} \tag{14}$$

where $\mathcal{S}_x$ indicates the input-only dataset, $\mathbf{y} \in \Omega$ indicates that output $\mathbf{y}$ is drawn from the hypothesis space Ω, and T indicates the length of $\mathbf{y}$. $J(\theta)$ is also called the performance function. Then the training objective is to maximize $J(\theta)$:

$$\hat{\theta} = \arg\max_\theta J(\theta) \tag{15}$$

Note that the hypothesis space Ω is typically very large, as LLMs often have large vocabularies[3]. Therefore, it is not feasible to enumerate all possible outputs directly. Instead, we typically use a Monte Carlo method to estimate this expectation. This approach involves sampling outputs $\{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_d\}$ from the LLM and using those samples to approximate the hypothesis space, denoted by $\mathcal{D}$:

$$J(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\mathcal{D}} \Pr_\theta(\mathbf{y}|\mathbf{x}) \sum_{t=1}^{T} r_t\right] \tag{16}$$

[3]The hypothesis space is large because LLMs typically have vast vocabularies and complex tokenization schemes. The number of possible combinations of tokens, even for a modest length output, grows exponentially, leading to a very large output space. This makes exhaustive enumeration of all possible outputs computationally infeasible.

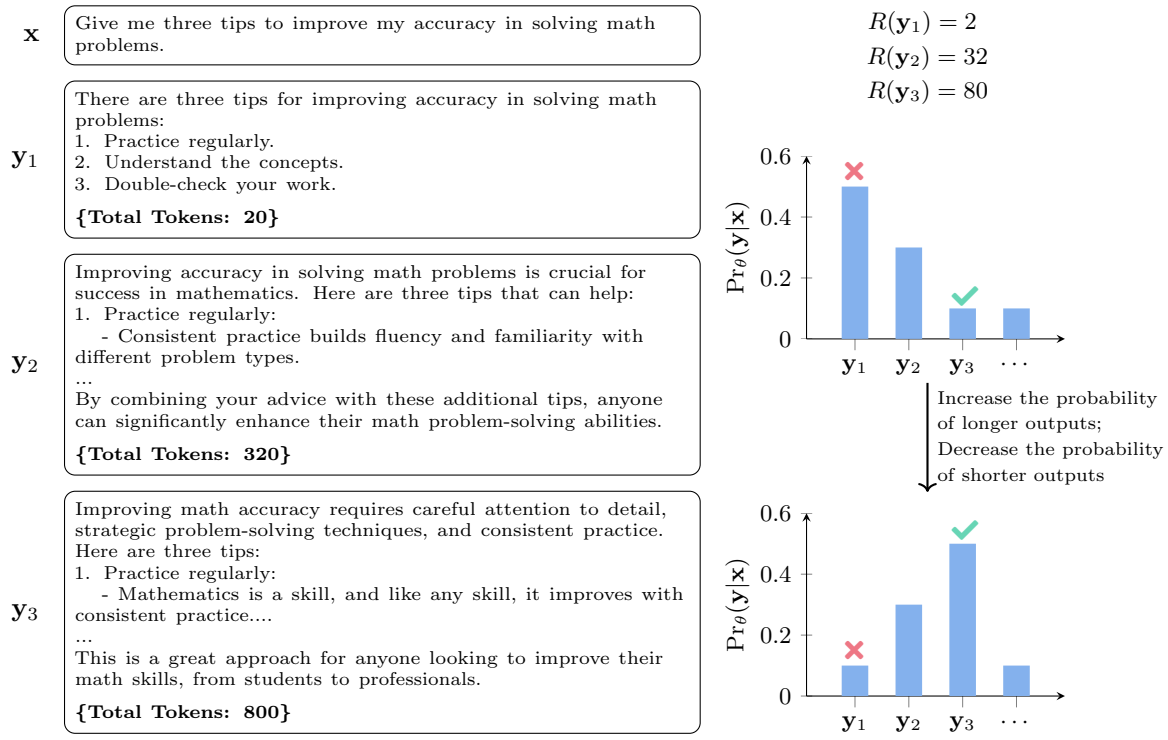


Figure 4: The mechanism of RL in training LLMs from the perspective of hypothesis space reranking: given an input $\mathbf{x}$, the model increases the probability of longer outputs, such as $\mathbf{y}_3$, within the hypothesis space by assigning higher rewards to them; conversely, shorter outputs like $\mathbf{y}_1$ receive lower rewards, which reduces their probabilities.

We can further refine the process when optimizing the objective using gradient descent. First, we compute the gradient of $J(\theta)$ with respect to θ:

$$\begin{aligned}
\frac{\partial J(\theta)}{\partial \theta} &= \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\mathcal{D}} \frac{\partial \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x}) R(\mathbf{y})}{\partial\theta}\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\mathcal{D}} \frac{\partial \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta} R(\mathbf{y})\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\mathcal{D}} \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x}) \frac{\partial \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x})/\partial\theta}{\mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x})} R(\mathbf{y})\right] \\
&= \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\sum_{\mathbf{y}\in\mathcal{D}} \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x}) \frac{\partial \log \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta} R(\mathbf{y})\right]
\end{aligned} \tag{17}$$

Using a Monte Carlo sample average over d outputs in $\mathcal{D}$, we can simplify Eq. (17) and need only consider the terms $\frac{\partial \log \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta}$ and $R(\mathbf{y})$:

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\frac{1}{d}\sum_{\mathbf{y}\in\mathcal{D}} \frac{\partial \log \mathrm{Pr}_\theta(\mathbf{y}|\mathbf{x})}{\partial\theta} R(\mathbf{y})\right] \tag{18}$$

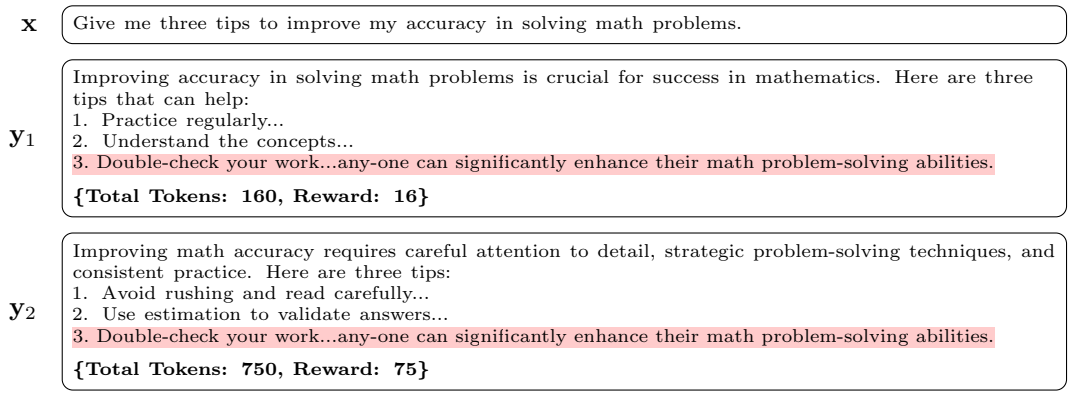


Figure 5: Example of sampled outputs demonstrating that the same tokens will receive significantly different rewards (i.e., 16 vs. 75). This introduces high gradient variance in the optimization process of the policy gradient.

Now, as illustrated in Figure 3, we have an RL approach to optimize the output of the LLM: 1) we sample inputs from the input-only dataset and sample outputs from the LLM; then, 2) we evaluate each sampled output using a reward function that we define; then, 3) we update the LLM using gradient descent, following Eq. (18), to maximize the performance function.

We can understand this optimization process from the perspective of hypothesis space reranking, as illustrated in Figure 4. The objective is to increase the probability of longer outputs in the hypothesis space by assigning a high reward to those outputs, while simultaneously decreasing the probability of shorter outputs by assigning them a lower reward. In other words, the policy gradient approach encourages the model to generate longer outputs by reinforcing those outputs that meet the desired length criteria, and it penalizes shorter outputs that do not meet the objective.

3.2 Temporal Decomposition

While optimizing using Eq. (18) is intuitive, a potential issue arises: in some cases, the sampled outputs may significantly overlap, yet the rewards for the overlapping parts differ. For example, as illustrated in Figure 5, if outputs $\mathbf{y}_1$ and $\mathbf{y}_2$ both include the same tokens in the third tip "3. Double-check your work...anyone can significantly enhance their math problem-solving abilities.", but due to variations in the entire outputs, they receive markedly different rewards (e.g., $R(\mathbf{y}_1) = 50$ vs. $R(\mathbf{y}_2) = 10$). Ideally, we would want similar generation behaviors to be rewarded consistently, without significant variation, as their contributions are equivalent to the sum of rewards.

Before discussing an approach to solve this issue, we first perform temporal decomposition for the term $\frac{\partial \log \Pr_\theta(\mathbf{y}|\mathbf{x})}{\partial \theta} R(\mathbf{y})$ in Eq. (18), and obtain

$$\begin{aligned} \frac{\partial \log \Pr_\theta(\mathbf{y}|\mathbf{x})}{\partial \theta} R(\mathbf{y}) &= \Big(\sum_{t=1}^{T} \frac{\partial \log \Pr_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t})}{\partial \theta}\Big)\Big(\sum_{k=1}^{T} r_k\Big) \\ &= \sum_{t=1}^{T} \frac{\partial \log \Pr_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t})}{\partial \theta}\Big(\sum_{k=1}^{t-1} r_k + \sum_{k=t}^{T} r_k\Big) \end{aligned} \tag{19}$$

In this form, we observe that, when generating the token at t-th time step, it does not affect the term $\sum_{k=1}^{t-1} r_k$ and only affects the term $\sum_{k=t}^{T} r_k$. Therefore, we can safely remove the former part, $\sum_{k=1}^{t-1} r_k$, to prevent it from affecting the optimization of the current step. By doing so, we can achieve a more stable

gradient computation for Eq. (18):

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\frac{1}{d}\sum_{\mathbf{y}\in\mathcal{D}}\sum_{t=1}^{T}\frac{\partial \log \Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\partial\theta}\sum_{k=t}^{T} r_k\right] \tag{20}$$

In fact, this simplification follows the Markov process, asserting that the future is independent of the past, given the present. As a result, we focus on the reward terms from the t-th time step onward, which are directly influenced by the token generation at time step t.

3.3 Reducing Gradient Variance

Returning to the case discussed in Section 3.2, while the strategy of excluding rewards for past tokens reduces gradient variance, substantial gradient variance still occurs in practice. First, if overlapping portions of the output appear early, the modified Eq. (21) may still experience similar problems, i.e., the same tokens receive vastly different rewards. Furthermore, the reward distribution can vary significantly between different time steps within a single output. For example, consider an output sequence of 500 tokens. According to Eq. (21), the reward at the 10th step might be significantly higher than at the 450th step, i.e., 49 vs. 5. Such varying rewards for good and poor tokens can result in a very low total reward for the entire output, even if it includes good tokens.

One simple method for further reducing the variance of the gradient is to set a baseline b and subtract it from $\sum_{k=t}^{T} r_k$, resulting in $\sum_{k=t}^{T} r_k - b$.[4] Here, the baseline can be interpreted as a reference point. By centering the rewards around this baseline, we remove systematic biases in the reward.

This policy gradient model with a baseline can be given by

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\frac{1}{d}\sum_{\mathbf{y}\in\mathcal{D}}\sum_{t=1}^{T}\frac{\partial \log \Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\partial\theta}\big(\sum_{k=t}^{T} r_k - b\big)\right] \tag{21}$$

There are many ways to define the baseline b. For example, we can set the baseline b to the average length of the sampled outputs across $\mathcal{D}$, i.e., $b = (\sum_{\mathbf{y}\in\mathcal{D}} \text{Length}(\mathbf{y}))/d$. While this approach can reduce the relative size of the gradient variance, it does not address the issue of varying gradient variance across different time steps within a single output. To tackle this challenge, we would want a different baseline for each time step that can specifically be adjusted to reduce variance at that step. To achieve this ideal baseline, we can introduce a value function $V(\cdot)$. In the training of the LLM, this value function calculates the expected value of the sum of future rewards (or return for short) when generating y_t, given the input $\mathbf{x}$ and the tokens previously generated $\mathbf{y}_{<t}$, i.e., $V(\mathbf{x},\mathbf{y}_{<t},y_t) = \mathbb{E}[\sum_{k=t}^{T} r_k]$. Hence we have

$$\begin{aligned} A(\mathbf{x},\mathbf{y}_{<t},y_t) &= \sum_{k=t}^{T} r_k - b \\ &= \sum_{k=t}^{T} r_k - V(\mathbf{x},\mathbf{y}_{<t},y_t) \end{aligned} \tag{22}$$

where $\sum_{k=t}^{T} r_k$ represents the actual return received, and $V(\mathbf{x},\mathbf{y}_{<t},y_t)$ (or V_t for short) represents the expected return at time step t. $A(\mathbf{x},\mathbf{y}_{<t},y_t)$ (or A_t for short) is called the advantage at time step t, which

[4] In fact, the use of a baseline b does not change the variance of the total rewards $\sum_{t=1}^{T} r_t$. However, it is important to note that while introducing a baseline does not alter the overall variance of the rewards, it helps reduce the variance of the gradient estimates. This is because subtracting the baseline from the total rewards effectively reduces fluctuations around their mean, which makes the gradient estimates more stable. In general, the operation $\sum_{k=t}^{T} r_k - b$ centers the rewards around zero (e.g., b is defined as the expected value of $\sum_{k=t}^{T} r_k$), which can lead to reduced variance in the product $\sum_{k=t}^{T} \log \Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})(\sum_{k=t}^{T} r_k - b)$.

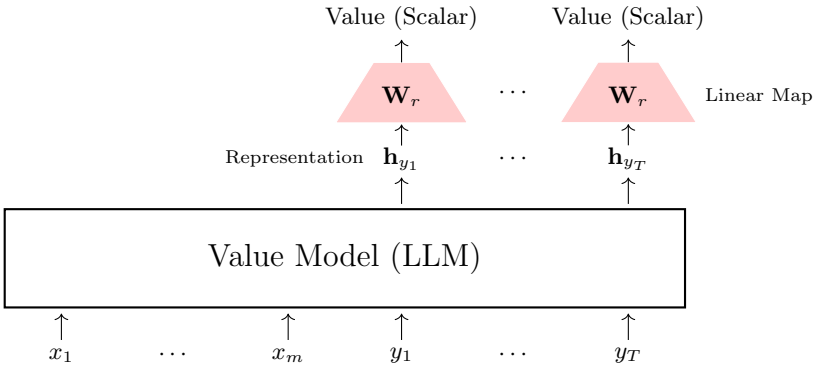


Figure 6: Architecture of the value model based on a pre-trained LLM. The main component of this model is still an LLM. For a given sequence $\text{seq}_{\mathbf{x},\mathbf{y}} = [\mathbf{x}, \mathbf{y}]$, the model extracts representations corresponding to the positions of the tokens in $\mathbf{y}$. These representations are then individually mapped to scalars through a linear transformation. Each scalar represents the value associated with each token in $\mathbf{y}$. Note that the linear mapping matrix used for the transformations is typically shared across all tokens.

quantifies the relative benefit of the current generated y_t compared to the expected return. Computing the advantage using Eq. (22) is a method known as Monte Carlo-based advantage estimation. By using the advantage function A_t, the gradient of $J(\theta)$ can be written in the form

$$\frac{\partial J(\theta)}{\partial \theta} = \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\left[\frac{1}{d}\sum_{\mathbf{y}\in\mathcal{D}}\sum_{t=1}^{T}\frac{\partial(\log \Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})A_t)}{\partial\theta}\right] \tag{23}$$

Based on this training objective, the loss function of training the LLM can be written in the form

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\mathbb{E}_{\mathbf{y}\sim\mathcal{D}}\left[\sum_{t=1}^{T}\log \Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})A_t\right] \tag{24}$$

In practice, there are many ways to implement the value function. One simple approach is to build it based on a pre-trained LLM (e.g., an SFT LLM). In this way, the value function is also called the value model (or critic model). More specifically, we can concatenate $\mathbf{x}$ and $\mathbf{y}$ to form a single token sequence $\text{seq}_{\mathbf{x},\mathbf{y}} = [\mathbf{x}, \mathbf{y}]$. We run an SFT LLM on this sequence, as usual, and at each position, we obtain a representation from the top-most Transformer layer. Then, we take the representations at the output (denoted by $\{\mathbf{h}_{y_1}, \mathbf{h}_{y_2}, \cdots, \mathbf{h}_{y_T}\}$) and map them to scalars via linear transformation, respectively. Figure 6 illustrates this architecture of the value model. Hence, at t-th time step, we can compute the value V_t

$$V_t = \mathbf{h}_{y_t}\mathbf{W}_v \tag{25}$$

where $\mathbf{h}_{y_t}$ is a d-dimensional vector, and $\mathbf{W}_v$ is a $d \times 1$ linear mapping matrix. This value model is typically trained using the Temporal Difference (TD) error. This training method relies on the principle that the difference between the values at adjacent time steps should correspond to the reward received at the former time step, i.e., $V_t - V_{t+1} = r_t$. Suppose the value model is parameterized by ω. Given a sequence $\text{seq}_{\mathbf{x},\mathbf{y}}$, the loss function is given by

$$\mathcal{L}_v(\omega) = \frac{1}{T}\sum_{t=1}^{T}\left(r_t + V_{\omega,t+1} - V_{\omega,t}\right)^2 \tag{26}$$

or alternatively, introduce the discount factor γ to obtain a more general form

$$\mathcal{L}_v(\omega) = \frac{1}{T}\sum_{t=1}^{T}\left(r_t + \gamma V_{\omega,t+1} - V_{\omega,t}\right)^2 \tag{27}$$

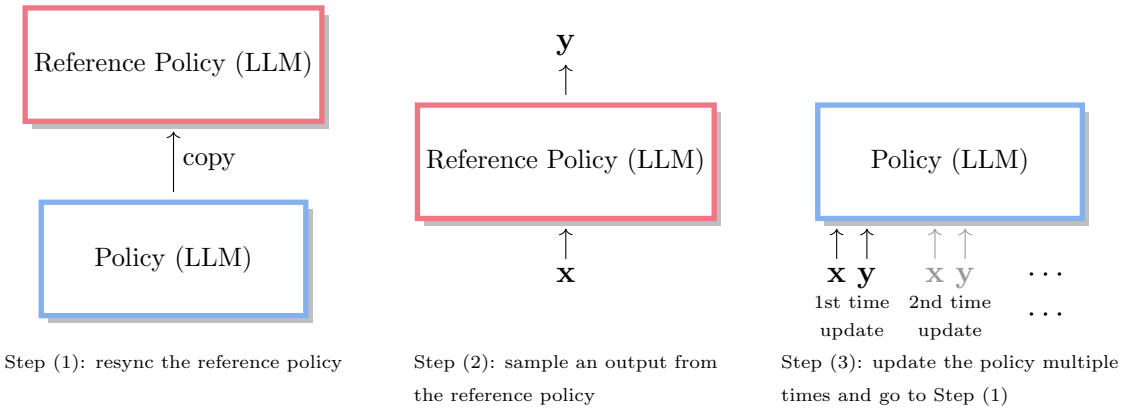


Figure 7: Workflow of integrating importance sampling in the training of LLMs with policy gradient. In Step 1, the current policy is synchronized to establish the reference policy. In Step 2, we sample an output $\mathbf{y}$ from this reference policy for a given input $\mathbf{x}$. Finally, in Step 3, the sequence $[\mathbf{x}, \mathbf{y}]$ is used to optimize the current policy multiple times. After optimization, the updated policy is synchronized to become the reference policy.

where $\gamma \in [0, 1]$ is the discount factor that adjusts the importance of future rewards. When γ is set to less than 1, it signifies that early rewards are considered more important than future rewards. This basic idea is also applied in other fields. For example, in LLMs, it has been demonstrated that early token generation plays a crucial role, as it can influence the style and accuracy of the entire output (Wang & Zhou, 2024).

In this subsection, we have detailed the integration of a value model in training LLMs. In practice, this training approach, represented by Eq. (23), is known as the Advantage Actor-Critic (A2C) method (Mnih et al., 2016). The A2C method facilitates an interaction where a policy model (the actor) and a value model (the critic) learn in parallel and undergo synchronous updates. However, RL is a vast field, and many technical details cannot be covered here. The interested reader can refer to RL books for more details (Szepesvári, 2010; Sutton & Barto, 2018).

3.4 Importance Sampling

In this subsection, we discuss methods to improve the efficiency of the policy gradient, focusing on the sampling process. Although we discussed in Section 3.1 that Monte Carlo-based sampling can estimate the hypothesis space, it is awfully inefficient for one single update in scenarios like ours where the sampled output may consist of hundreds or thousands of tokens. A natural idea from this point is to consider whether it is possible to reuse these sampled outputs to update the LLMs multiple times. With importance sampling, this is feasible, and we can rewrite the loss function of training the LLM as

$$\mathcal{L}(\theta) \quad = \quad -\mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\mathbb{E}_{\mathbf{y}\sim\Pr_{\theta_{\text{ref}}}(\cdot|\mathbf{x})}\left[\sum_{t=1}^{T}\frac{\Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\Pr_{\theta_{\text{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}A_t\right] \tag{28}$$

where θ_{ref} denotes the parameters of the previously used LLM (also called reference policy or old policy). Here, we use $\Pr_\theta(\cdot|\mathbf{x})$ or $\Pr_{\theta_{\text{ref}}}(\cdot|\mathbf{x})$ to denote $\mathcal{D}$, distinguishing from which model the output is sampled. The ratio $\frac{\Pr_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\Pr_{\theta_{\text{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}$, also called the ratio function, compares the probability of token y_t under the current and reference policies. This ratio function is used to reweight the observed rewards, reflecting how much more or less likely a token is under the current policy compared to the reference policy. When this ratio is greater than 1, it indicates that the token y_t is more favored by the current policy than the reference policy. Conversely, a ratio less than 1 indicates that y_t is less favored by the current policy. However, when the current θ_{ref} diverges significantly from θ, the accuracy of the hypothesis space estimation decreases.

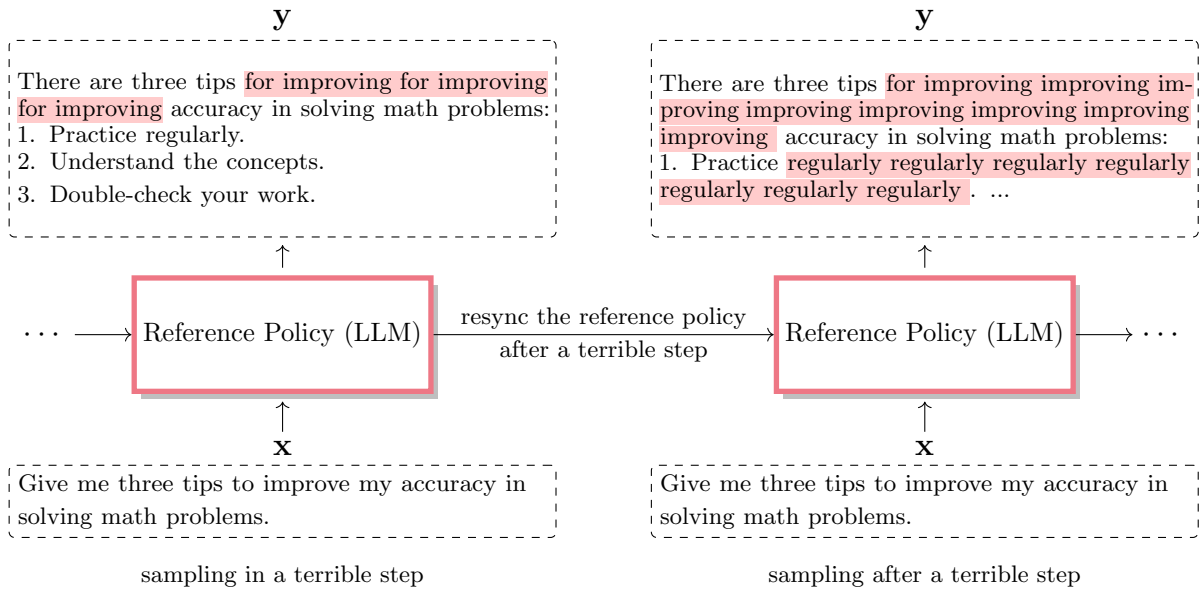


Figure 8: Illustration of the impact of a terrible step on sampling within an LLM training cycle using importance sampling. The left shows poor output sampled with slight repetition resulting in a terrible step. The right shows subsequent output sampled with more severe repetition after a terrible step.

Therefore, we do need to resync it regularly. As illustrated in Figure 7, one simple way is to designate the reference policy as the LLM from which we initiate updates during a training step. Note that we can conserve computational time and memory by eliminating the need for model copying in Step 1. Specifically, rather than maintaining a separate copy of the model, we directly sample from the current policy, retain the output probabilities $\{\Pr_\theta(y_1|\mathbf{x}), \cdots, \Pr_\theta(y_T|\mathbf{x}, \mathbf{y}_{<T})\}$, and later use these stored probabilities to act as $\{\Pr_{\theta_{\text{ref}}}(y_1|\mathbf{x}), \cdots, \Pr_{\theta_{\text{ref}}}(y_T|\mathbf{x}, \mathbf{y}_{<T})\}$ when updating the policy with Eq. (28).

However, an inherent issue arises when using importance sampling to update LLMs. As shown in Figure 8, when a particular optimization step results in poor updates (or a terrible step for short), the reference policy utilized in subsequent Step 1 of the next iteration inherits these poor characteristics. Consequently, the samples drawn in Step 2 are likely to be adversely affected, leading to a more terrible step. Unlike SFT[5], this cycle does not correct the initial terrible step but potentially exacerbates it, creating a downward spiral in policy performance.

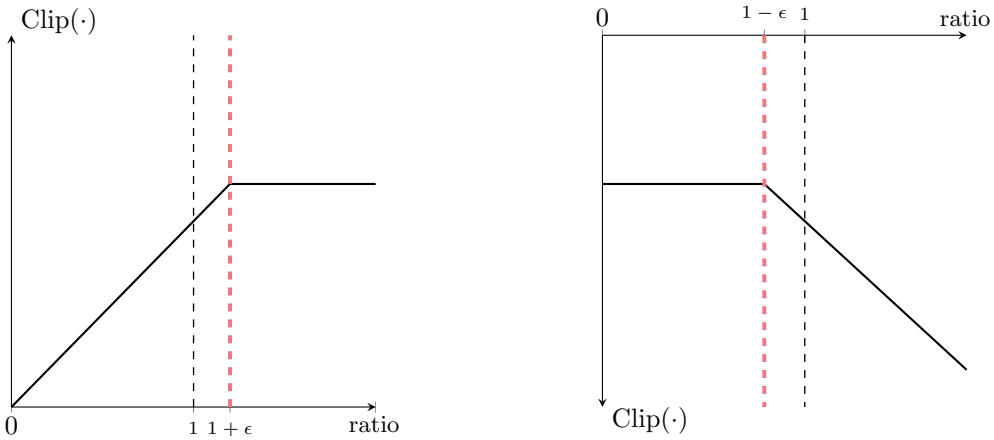
Addressing this issue involves considering trust regions in optimization (Schulman et al., 2015), which refers to a region around the current parameter estimate where the model is well-behaved. One approach to incorporating trust regions is to impose a constraint on the size of the policy update. This can be effectively managed by imposing a constraint that prevents significant deviations from the policy before optimization. In training LLMs, this constraint is typically achieved by adding a penalty to the objective function based on a divergence measure between current and reference policies. A simple form of such a penalty is given by the difference in the log-probability of the sampled $\mathbf{y}$ under the current policy versus the reference policy:

$$\text{Penalty} = \log \Pr_\theta(\mathbf{y}|\mathbf{x}) - \log \Pr_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x}) \tag{29}$$

At the time step t, we can obtain the penalty as

$$\text{Penalty}_t = \log \Pr_\theta(y_t|\mathbf{x}, \mathbf{y}_{<t}) - \log \Pr_{\theta_{\text{ref}}}(y_t|\mathbf{x}, \mathbf{y}_{<t}) \tag{30}$$

[5]In supervised learning, a terrible step during a training step caused by bad samples can often be corrected in subsequent steps by good samples.



(a) $A_t > 0$ (b) $A_t < 0$

Figure 9: PPO clipping at time step t. The basic idea can be represented with a hypothesis space as described in Figure 4. The sub-figure (a) shows the case where the advantage A_t is positive, suggesting that the output is preferred. Here, the probability of the sampled output is increased within a defined constraint (up to $1+\epsilon$) in the hypothesis space. The sub-figure (b) shows the case where A_t is negative, indicating a dispreferred output. In this case, the probability of the sample output is reduced to a minimum (down to $1-\epsilon$) in the hypothesis space.

By including this penalty in the optimization objective, we encourage the current policy to remain close to the reference policy, limiting very large updates in a terrible step.

We can incorporate this penalty into the Eq. (28), and obtain

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\mathbb{E}_{\mathbf{y}\sim\mathrm{Pr}_{\theta_{\mathrm{ref}}}(\cdot|\mathbf{x})}\left[\sum_{t=1}^{T}\left(\frac{\mathrm{Pr}_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\mathrm{Pr}_{\theta_{\mathrm{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}A_t - \beta\mathrm{Penalty}_t\right)\right] \tag{31}$$

where β is the weight of the penalty. This training method is also called trust region policy optimization (TRPO) (Schulman et al., 2015).

3.5 Proximal Policy Optimization

A further improvement to TRPO involves addressing the gradient variance problem outlined in Section 3.3. In Eq. (31), the range of the ratio function is from $[0, +\infty)$, which could lead to high gradient variance in the learning process. To mitigate this problem, clipping is commonly employed to limit the magnitude of importance weights, thereby preventing excessively large updates and promoting stability in the learning process. A clipped version can be given by

$$\mathcal{L}(\theta) = -\mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x}\mathbb{E}_{\mathbf{y}\sim\mathrm{Pr}_{\theta_{\mathrm{ref}}}(\cdot|\mathbf{x})}\left[\sum_{t=1}^{T}\left(\mathrm{Clip}\big(\frac{\mathrm{Pr}_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\mathrm{Pr}_{\theta_{\mathrm{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}A_t\big) - \beta\mathrm{Penalty}_t\right)\right] \tag{32}$$

where the clipping function Clip(·) can be defined by

$$\mathrm{Clip}\big(\frac{\mathrm{Pr}_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\mathrm{Pr}_{\theta_{\mathrm{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}A_t\big) = \min\left(\frac{\mathrm{Pr}_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\mathrm{Pr}_{\theta_{\mathrm{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}A_t, \mathrm{bound}\big(\frac{\mathrm{Pr}_\theta(y_t|\mathbf{x},\mathbf{y}_{<t})}{\mathrm{Pr}_{\theta_{\mathrm{ref}}}(y_t|\mathbf{x},\mathbf{y}_{<t})}, 1-\epsilon, 1+\epsilon\big)A_t\right) \tag{33}$$

where the function bound(·) constrains the ratio function to within the range $[1-\epsilon, 1+\epsilon]$. The clipping imposed by Eq. (33) is illustrated in Figure 9. This training method is also known as proximal policy

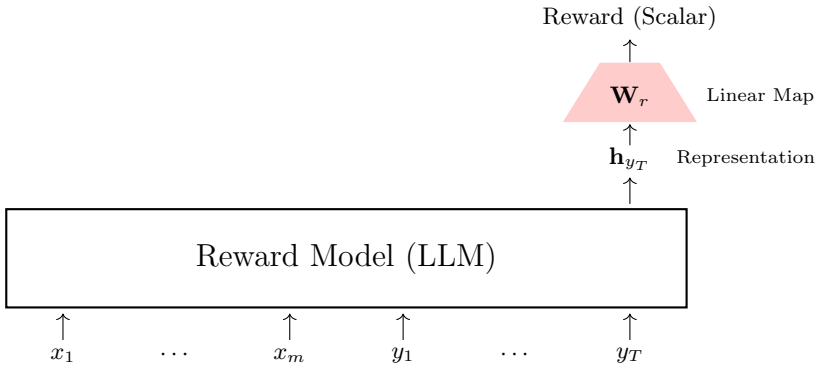


Figure 10: Architecture of the reward model based on an LLM. Unlike the value model depicted in Figure 6, this model extracts the representation from the last position of the output $\mathbf{y}$ to represent the entire sequence $[\mathbf{x}, \mathbf{y}]$. This representation is then mapped to a scalar through a linear transformation, which serves as the reward for the output.

optimization (PPO) (Schulman et al., 2017), currently the most widely used method for training LLMs with RL. Originally, PPO also introduced an adaptive β to dynamically control this penalty. However, this adaptive approach has not been widely applied in training LLMs. This is because this process depends on the expectation of the penalty (i.e., $\mathbb{E}(\text{Penalty})$), which would introduce additional computational overhead. Interested readers can refer to the original literature for more details on this adaptive approach.

3.6 Training Reward Models

While our initial reward function based on output length, as described in Section 3.1, provides effective signals in the learning process, human preferences are considerably more complex and extend beyond merely preferring longer outputs. For example, in scenarios like a homework assistant, it is essential not only to generate longer outputs but also to avoid redundancy, ensure accuracy, and maintain fluency. This complexity underscores the need for a sophisticated reward modeling technique, moving beyond simple function-based methods to more effectively capture human preferences.

Given these complexities, we typically train a reward model, a neural network that maps a pair of input and output token sequences to a scalar value, to capture human preferences. Given an input $\mathbf{x}$ and an output $\mathbf{y}$, the reward is expressed as $\text{Reward}(\mathbf{x}, \mathbf{y})$, where $\text{Reward}(\cdot)$ denotes the reward model. There are many ways to implement the reward model. One simple approach is to build the reward model based on a pre-trained LLM, similar to the value model as presented in Section 3.3. More specifically, we employ the sequence $\text{seq}_{\mathbf{x},\mathbf{y}} = [\mathbf{x}, \mathbf{y}]$ to serve as the input. We run the LLM on this sequence and obtain a representation from the top-most Transformer layer. Then, we take the representation at the last position of the output and map it to a scalar via linear transformation[6]:

$$R_\phi(\mathbf{x}, \mathbf{y}) \quad = \quad \mathbf{h}_{y_T} \mathbf{W}_r \tag{34}$$

where $\mathbf{W}_r$ is a $d \times 1$ linear mapping matrix, and ϕ represents the parameters of the reward model, which includes both the parameters of the LLM and the $\mathbf{W}_r$. This architecture of the reward model is illustrated in Figure 10.

[6]In practice, when employing an LLM for training a reward model, we utilize it primarily as an encoder to encode the sequence $\text{seq}_{\mathbf{x},\mathbf{y}}$ into a representation. Here, the representation from the last position is selected as it encapsulates the semantic information of the entire sequence.

To train the reward model, the first step is to collect human feedback on a set of generated outputs. Given an input $\mathbf{x}$, we use the LLM to produce multiple candidate outputs $\{\mathbf{y}_1, ..., \mathbf{y}_d\}$. Then, we can obtain human feedback in the following ways:

- **Rating**. Human experts provide a score or rating for each output. This score is often a continuous or discrete numerical value, such as a score on a scale (e.g., 1-5 stars or 1-10 points). In some cases, the rating might be binary, indicating a "yes/no" or "positive/negative" preference.
- **Listwise Ranking**. Human experts are asked to rank or order the given set of possible outputs.
- **Pairwise Comparison** (**Pairwise Ranking**). Given two different outputs, human experts select which one is better. This annotation approach is particularly popular because it is easier to annotate than the other two methods.

Here we consider pairwise comparison feedback to train a reward model. In this setting, each time, two outputs $(\mathbf{y}_a, \mathbf{y}_b)$ are randomly drawn from the candidate output pool $\{\mathbf{y}_1, ..., \mathbf{y}_d\}$. Human experts are then presented with these pairs and asked to decide which output they prefer based on specific criteria, such as information, accuracy, and fluency. The human feedback can be encoded as a binary label, $\mathbf{y}_a \succ \mathbf{y}_b$ for a preference for $\mathbf{y}_a$, and $\mathbf{y}_b \succ \mathbf{y}_a$ for a preference for $\mathbf{y}_b$.

One simple and widely used model for describing such pairwise comparisons is the Bradley-Terry model (Bradley & Terry, 1952). It is a probabilistic model that estimates the probability that one item is preferred over another. Adapting this model to the notation used here, we can write the probability that $\mathbf{y}_a$ is preferred over $\mathbf{y}_b$ in the form

$$\begin{aligned} \Pr_\phi(\mathbf{y}_a \succ \mathbf{y}_b|\mathbf{x}) &= \frac{e^{R_\phi(\mathbf{x},\mathbf{y}_a)}}{e^{R_\phi(\mathbf{x},\mathbf{y}_a)} + e^{R_\phi(\mathbf{x},\mathbf{y}_b)}} \\ &= \frac{e^{R_\phi(\mathbf{x},\mathbf{y}_a) - R_\phi(\mathbf{x},\mathbf{y}_b)}}{e^{R_\phi(\mathbf{x},\mathbf{y}_a) - R_\phi(\mathbf{x},\mathbf{y}_b)} + 1} \\ &= \text{Sigmoid}(R_\phi(\mathbf{x},\mathbf{y}_a) - R_\phi(\mathbf{x},\mathbf{y}_b)) \end{aligned} \tag{35}$$

When training the reward model, we want to maximize this preference probability. A loss function based on the Bradley-Terry model is given by

$$\begin{aligned} \mathcal{L}_r(\phi) &= -\mathbb{E}_{(\mathbf{x},\mathbf{y}_a,\mathbf{y}_b)\sim\mathcal{D}_r}\left[\log \Pr_\phi(\mathbf{y}_a \succ \mathbf{y}_b|\mathbf{x})\right] \\ &= -\mathbb{E}_{(\mathbf{x},\mathbf{y}_a,\mathbf{y}_b)\sim\mathcal{D}_r}\left[\log \text{Sigmoid}(R_\phi(\mathbf{x},\mathbf{y}_a) - R_\phi(\mathbf{x},\mathbf{y}_b))\right] \end{aligned} \tag{36}$$

where $(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b)$ is drawn from a human-annotated dataset $\mathcal{D}_r$ consisting of preference pairs of outputs and their corresponding inputs. The goal of training the reward model is to find the optimal parameters $\hat{\phi}$ that minimize this loss function, given by

$$\hat{\phi} = \arg\min_\phi \mathcal{L}_r(\phi) \tag{37}$$

Since the reward model itself is also an LLM, we can directly reuse the Transformer training procedure to optimize the reward model. The difference from training a standard LLM is that we only need to replace the cross-entropy loss with the pairwise comparison loss as illustrated in Figure 11. After the training of the reward model, we can apply the trained reward model $R_{\hat{\phi}}(\cdot)$ to supervise the target LLM for alignment.

It is worth noting that although we train the reward model to perform pairwise ranking, we apply it to score each input-output pair independently during the alignment process. The pairwise ranking objective ensures that the reward model is sensitive to subtle differences between outputs, but we rely on the continuous scores produced by the reward model to guide the optimization of the LLM. An advantage of this approach is that we can choose from or combine various ranking loss functions and still apply the resulting reward models in the same way as we have done in Section 4.1.3. However, a challenge arises with this method: the

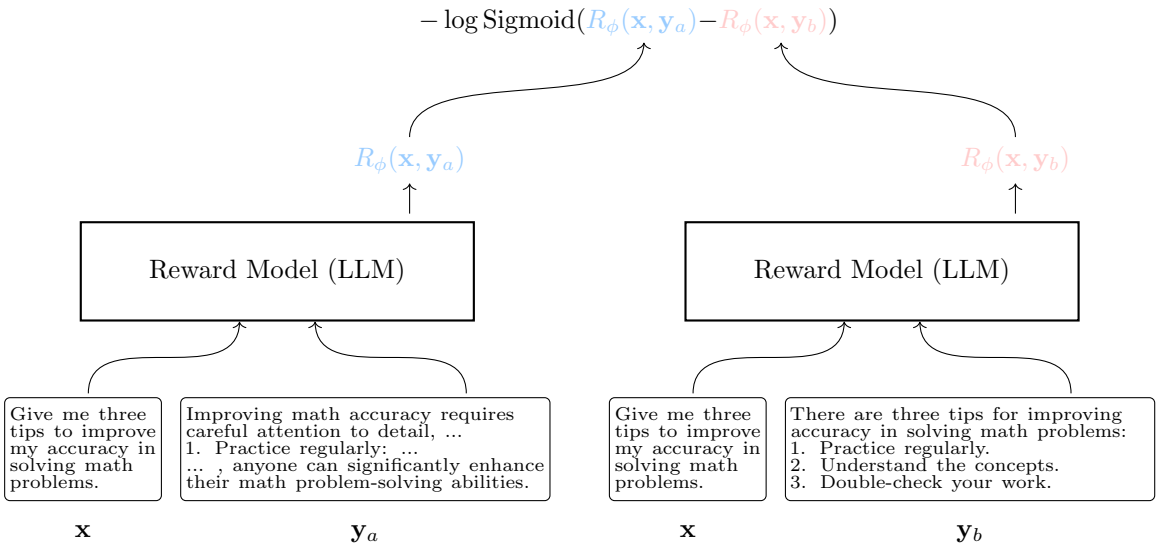


Figure 11: An example of loss computation using the Bradley-Terry model. Given an input $\mathbf{x}$ and two outputs $(\mathbf{y}_a, \mathbf{y}_b)$, where $\mathbf{y}_a \succ \mathbf{y}_b$, we initially obtain rewards $R(\mathbf{x}, \mathbf{y}_a)$ and $R(\mathbf{x}, \mathbf{y}_b)$ for two sequences $\text{seq}_{\mathbf{x},\mathbf{y}_a}$ and $\text{seq}_{\mathbf{x},\mathbf{y}_b}$. These rewards are subsequently used to compute the loss as described in Eq. (36). Note that, for efficiency, both sequences are typically processed in a single batch during one forward pass to simultaneously obtain $R(\mathbf{x}, \mathbf{y}_a)$ and $R(\mathbf{x}, \mathbf{y}_b)$. In fact, this training approach is also common in Siamese networks (Bromley et al., 1993).

reward model can provide only sparse rewards; that is, it offers a delayed reward rather than an intermediate one. Hence, in this case, we can obtain

$$r_t = \begin{cases} 0 & \text{if } t < T \\ R_\phi(\mathbf{x}, \mathbf{y}) & \text{if } t = T \end{cases} \tag{38}$$

To address this challenge, we can typically incorporate shaping rewards into the learning process (Wu et al., 2023) or train a process reward model by annotating process preference data (Lightman et al., 2024a). Please refer to Section 5.1.2 for more details.

3.7 A Complete Workflow

Up to this point, we have used numerous examples to discuss how to use RL to train LLMs to better align with human preferences. We have also shown how different RL algorithms are designed to address specific challenges in our scenarios. In this subsection, we will outline a complete workflow for training LLMs using RL, specifically employing the PPO. To implement PPO, we first build four models, all based on an LLM.

- **Reward Model** (denoted by $R_\phi(\cdot)$ where ϕ denotes the parameters). The reward model learns from human preference data to predict the reward for each pair of input and output token sequences. It is typically initialized with an SFT LLM or a pre-trained LLM.
- **Value Model** or **Value Function** (denoted by $V_\omega(\cdot)$ where ω denotes the parameters). The value model receives rewards from the reward model and is trained to predict the expected sum of rewards. It is generally initialized with a reward model or an SFT LLM.
- **Reference Model** or **Old Policy Model** (denoted by $\Pr_{\theta_{\text{ref}}}(\cdot)$ where θ_{ref} denotes the parameters). The reference model is the baseline LLM that serves as a starting point for policy training. Unlike

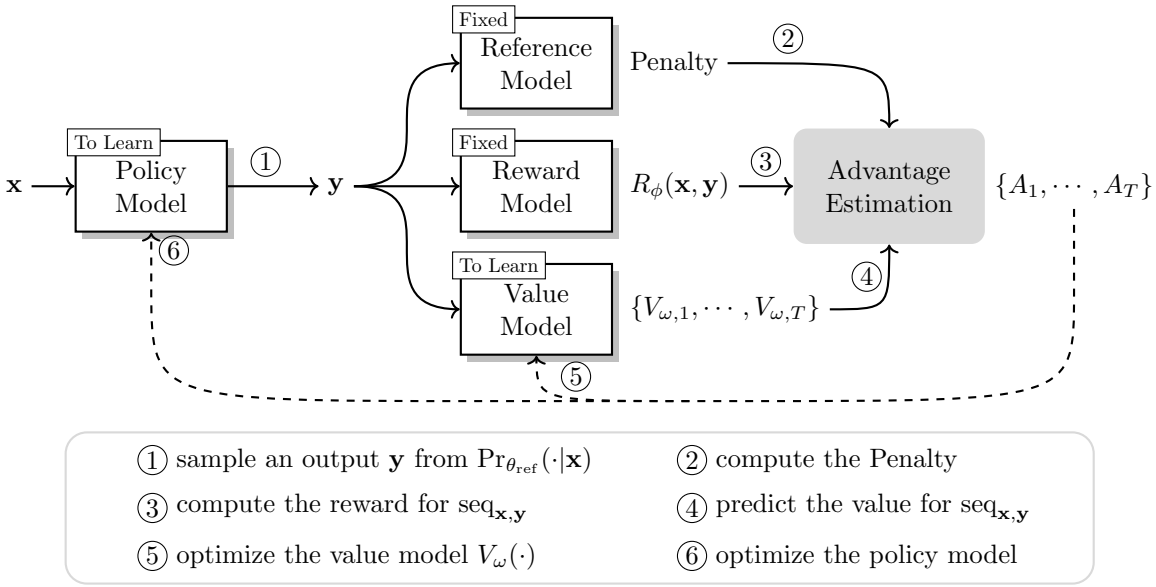


Figure 12: Illustration of PPO-based optimization workflow. Given an optimized reward model and a reference model, we proceed to train both the policy and the value model. At each prediction step, we compute the sum of the PPO-based loss and update the policy parameters. This requires access to the reward model, the reference model, and the value model at hand. At the same time, we update the parameters of the value model. It is worth noting that steps 2, 3, and 4 do not follow a fixed order, and they can be executed simultaneously or in any order.

the discussion in Section 3.4, when computing the penalty in RLHF, we typically use the previous version of the model or a model trained without human feedback to serve as the reference policy model, making a more stable learning process.

- **Policy Model** (denoted by $\Pr_\theta(\cdot)$ where θ denotes the parameters). Given its context, this policy governs how the LLM decides the most appropriate next token. It is trained under the supervision of both the reward model and the value model.

In practice, these models need to be trained in a certain order. First, we need to initialize them using some other models. For example, the reward model and the value model can be initialized with a pre-trained LLM or an SFT LLM, while the reference model and the policy model can be initialized with an SFT LLM. Note that, at this point, the reference model is ready for use and will not be further updated. Second, we need to collect human preference data and train the reward model on this data. Third, both the value model and the policy are trained simultaneously using the reward model[7]. At each position in an output sequence, we update the value model by minimizing the MSE error of value prediction, and the policy is updated by minimizing the PPO loss.

Although the RL process introduced above seems highly promising, as it can effectively align the SFT LLM with human preferences, there are significant challenges in implementing this process. For example,

- Training a reward model requires labeled preference data. Consequently, it is essential to generate multiple outputs for a given input and label them according to human preferences. For example, in our scenario, we need to annotate extensive data to accurately reflect student preferences, thereby

[7]During PPO-based optimization, a cold-start approach can be used where only the value model is updated in the initial phases, avoiding adjustments to the policy model based on potential inaccuracies in early value predictions (Wang et al., 2024a).

enabling the homework assistant to produce the content students expect. Furthermore, in this process, it is crucial to ensure that the data is of high quality and accurately mirrors human preferences. This is because inaccuracies in data can lead to misguided learning, where the model might adopt undesirable biases or fail to meet user expectations, also called overoptimization or reward hacking (Gao et al., 2023). Therefore, such preference data must be meticulously collected and sufficiently extensive to enable the reward model to accurately capture human preferences.

- RL is computationally expensive for training LLMs. This arises from two primary factors. One is that during the RL process, we need to perform sampling in an autoregressive mode (Xiao & Zhu, 2023). This is highly computationally intensive because the policy model usually has a large number of parameters. Furthermore, during the PPO-based optimization, we need to load four LLMs simultaneously, which requires significantly more GPU memory compared to the SFT. To address these challenges, there has been significant interest in developing efficient RL strategies, such as dynamic sampling (Wang et al., 2024b) and rule-based rewards (Shao et al., 2024), which can maintain state-of-the-art performance without requiring high computational and time costs.

- RL is often an unstable process. While RL provides a strong theoretical foundation for the design of each component, it is also highly sensitive to changes in any part of the system. Even small adjustments to the reward model, policy model, or hyperparameters can lead to significant fluctuations in performance, making it difficult to ensure consistent and stable results. This fragility highlights the importance of carefully tuning and stabilizing the various elements involved in RL, especially when applied to complex systems like LLMs. Techniques such as reward shaping, adaptive learning rate, and more sophisticated optimization strategies are often required to mitigate instability and improve the robustness of the learning process.

4 Improved RL for LLMs

In the previous section, we introduced some improvements to RL, such as importance sampling and reward baseline techniques. However, directly applying them to train LLMs still presents numerous challenges. In this section, we will delve deeper into the improvements for using RL to train LLMs.

4.1 Advanced Reward Models

Reward models are a fundamental component in RL, as they define what a policy model optimizes for. Therefore, the quality of reward model training directly influences the effectiveness of LLM optimization during RLHF. A poorly trained reward model can lead to a suboptimal LLM, while a well-optimized reward model ensures that the LLM is effectively aligned with the desired behaviors and objectives. Here we will introduce several methods for obtaining advanced reward models. Our discussion will be relatively general, since reward models are widely used in many RL problems, such as robot planning and control. This broad applicability makes it straightforward to adapt the methods discussed here to other related applications.

4.1.1 Automatic Preference Data Generation

Although learning from human preferences is an effective and popular method for aligning LLMs, annotating preference data is costly. Relying on human feedback not only faces scalability limitations but may also introduce bias, as human feedback is inherently subjective. Consequently, AI-based feedback methods offer a promising solution to address these scalability and consistency issues, avoiding the limitations associated with human annotators.

One simple method is to generate preference data using an LLM. Given a set of inputs, we first use an LLM to generate pairs of outputs. Then, we prompt the LLM to label the preference between each pair of

outputs, along with its corresponding input. Below is an example of prompting the LLM to generate a preference label for a pair of outputs.

> Consider a homework assistant scenario in which a student types an input. You will review two outputs to this input. Please indicate which output you prefer. A good output should be informative, accurate, and well-articulated. It should directly address the student's question, provide thorough explanations or solutions, and maintain an encouraging tone.
>
> Input:
>
> *Give me three tips to improve my accuracy in solving math problems.*
>
> Output A:
>
> *Improving math accuracy requires careful attention to detail, strategic problem-solving techniques, and consistent practice. Here are three tips:*
> *1. Practice regularly:*
> *- Consistent practice builds fluency and familiarity with different problem types...*
> *...*
> *By combining your advice with these additional tips, anyone can significantly enhance their math problem-solving abilities.*
>
> Output B:
>
> *There are three tips for improving accuracy in solving math problems:*
> *1. Practice regularly.*
> *2. Understand the concepts.*
> *3. Double-check your work.*
>
> Output A is preferred.

Once we collect such preference labels, we can use them, along with the output pair and input, to train the reward model. Of course, these labels are not entirely accurate either. Recent studies have shown that AI-based feedback often exhibits a position bias problem, making it more likely to prefer the output shown first (Zheng et al., 2023). We can consider demonstrating a few examples or using advanced prompting techniques, such as Chain-of-Thought (CoT), to improve the labeling performance (Liu et al., 2023).

For data generation, although it is easy to scale up, it is often necessary to ensure the data is accurate and diverse. Here, the data quality and diversity issues involve not only the labeling of preferences but also the inputs and outputs of the model. Therefore, we often need to use a variety of techniques to obtain large-scale, high-quality data. For example, one can generate diverse model outputs and annotations by using different LLMs, prompts, in-context demonstrations, and so on (Cui et al., 2024b). Furthermore, Dubois et al. (2023) report that the variability in pairwise preference data is important for training LLMs from either human or AI feedback.

While learning from AI feedback is highly scalable and generally objective, this method is more suited to well-defined tasks where objective performance metrics are available. By contrast, learning from human feedback is more advantageous when aligning AI systems with human values, preferences, and complex real-world tasks that require an understanding of subtle or subjective context. These methods can be combined to train LLMs that benefit from both human insights and the scalability of AI feedback.

4.1.2 Reward Shaping

As discussed in Section 3.6, while the reward model is effective in capturing human preferences, it provides sparse rewards. These rewards, known as delayed rewards, are received only at the end of the output

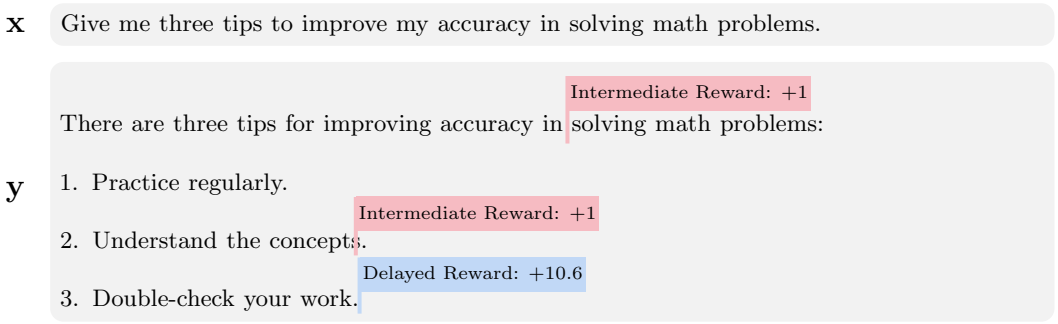


Figure 13: An example of reward shaping. Here, length-based shaping rewards are implemented, e.g., awarding a +1 intermediate reward for every 10 tokens generated. Additionally, a delayed reward from the reward model evaluates the overall quality of the output at the end.

generation process, as opposed to intermediate rewards that would be distributed continuously throughout it.

In fact, dealing with sparse rewards has long been a concern in RL, and has been one of the challenges in many practical applications. For example, robotics often needs to shape the reward function to ease optimization rather than relying solely on end-of-sequence rewards. Various methods have been developed to address this issue. One common approach is reward shaping, where the original function is modified to include intermediate rewards, thereby providing more immediate feedback. Here, the intermediate reward is often an indirect signal for improving the delayed reward. For example, as shown in Figure 13, setting a length-based reward as an intermediate reward encourages the generation of more content, potentially increasing the overall quality of the final output as assessed by the delayed reward from the reward model. Additional examples of reward shaping in training LLMs can be found in Kumar et al. (2025). In this way, we can obtain

$$r'_t = r_t + f(\mathbf{x}, \mathbf{y}_{<t}, y_t) \tag{39}$$

where $r'(\cdot)$ is the transformed reward, $r(\cdot)$ is the original delayed reward function, and $f(\cdot)$ is the shaping reward function that provides an intermediate reward. To ensure the optimality of the policy under the transformed reward function, the shaping reward function can be given in the form

$$f(\mathbf{x}, \mathbf{y}_{<t}, y_t) = \gamma\Phi(\mathbf{x}, \mathbf{y}_{<t+1}, y_{t+1}) - \Phi(\mathbf{x}, \mathbf{y}_{<t}, y_t) \tag{40}$$

where $\Phi(\cdot)$ is called the potential value function, and γ is the discount factor that controls how much the next-step potential contributes to the current shaping reward. If we define $\Phi(\cdot)$ as the common value function and substitute Eq. (40) into Eq. (39), we obtain

$$r'_t = r_t + \gamma V_{t+1} - V_t \tag{41}$$

It is interesting to see that this function is exactly the same as the advantage function used in PPO. This relates advantage-based methods to reward shaping: the advantage is essentially a shaped reward.

In practice, an alternative method for implementing reward shaping in training LLMs involves utilizing the reward model to provide intermediate rewards based on the differences between output rewards (Bahdanau et al., 2017; Wu et al., 2023). For example, at time step t, the intermediate reward r_t can be obtained by

$$r_t = R(\mathbf{x}, \mathbf{y}_{<t+1}) - R(\mathbf{x}, \mathbf{y}_{<t}) \tag{42}$$

However, in this case, it is crucial that the reward model is capable of accurately assigning rewards for partial outputs, i.e., $\{\mathbf{y}_{<2}, \mathbf{y}_{<3}, \cdots, \mathbf{y}_{<T}\}$.

In addition to reward shaping, another method to address the sparse reward issue is adopting curriculum learning, where tasks are structured sequentially with gradually increasing complexity. This can help models master simpler tasks first, which prepares them for more complex challenges as their skills develop. There are many methods that can mitigate the impact of sparse rewards, such as Monte Carlo methods and intrinsic motivation. Most of these methods are general, and the discussion of them can be found in the broader literature on RL, such as Sutton & Barto (2018)'s book.

4.1.3 Improved Reward Generalization

As discussed in Section 3.6, reward models are trained using preference data and subsequently used to optimize LLMs. However, a problem arises: the data distribution during LLM optimization may differ from the distribution of the preference data, making it challenging for the reward model to generalize to unseen input-output pairs.

A well-known failure mode associated with this problem is commonly referred to as *overoptimization* or *reward hacking*, where the optimization stage improves the reward model score but deteriorates the alignment with true rewards (Gao et al., 2023; Eisenstein et al., 2023). This mode occurs because the reward model may incorrectly assign high rewards to unseen input-output pairs, leading the LLM to learn and optimize for behaviors that do not truly align with the desired behaviors and objectives. For example, consider a scenario from Section 3.1 where the reward model is designed to favor informative and accurate outputs for a homework assistant. However, if the reward model fails to generalize to an overly verbose output and assigns a high reward to it (perhaps due to its length or certain keywords), the LLM may learn to prioritize generating a long output, which is not actually more informative but receives a higher reward according to the reward model. Consequently, the LLM becomes misaligned with its true objectives, delivering concise and relevant information, because it optimizes for the incorrect reward signal from the reward model with weak generalization.

Addressing this generalization problem is challenging, and no mature solution exists yet. The ideal approach would be to develop an oracle reward model that perfectly captures the true objectives of the task and generalizes across all input-output pairs during LLM optimization. However, creating such a model is extremely difficult due to the complexity of the real-world environment, as well as the challenge of collecting sufficient preference data. Instead, a more practical approach is to combine multiple reward models, improving generalization and providing more accurate rewards (Coste et al., 2024).

Given a set of reward models, combining them is straightforward, and in some cases, we can simply treat this problem as an ensemble learning problem. A simple yet common approach is to average the outputs of these models to obtain a more precise reward estimation:

$$R_{\text{combine}}(\mathbf{x}, \mathbf{y}) \quad = \quad \frac{1}{K} \sum_{k=1}^{K} w_k \cdot R_k(\mathbf{x}, \mathbf{y}) \tag{43}$$

where $R_k(\cdot)$ is the k-th reward model in the ensemble, w_k is the weight of $R_k(\cdot)$, and K is the number of reward models. This combined reward can then be used to supervise the training of a policy. In fact, there are many ways to combine different models; for example, one can make predictions using Bayesian model averaging or develop a fusion network to learn to combine the predictions from different models. Alternatively, one can frame this task as a multi-objective optimization problem and use multiple reward models to train the policy simultaneously. These methods have been intensively discussed in the literature on optimization and machine learning (Miettinen, 1999; Bishop, 2006).

On the other hand, to improve the generalization of reward models, it is important to prevent overfitting to preference data. Reward models are typically trained on an SFT LLM, which has a strong generalization capability. However, training the LLM with a large amount of preference data could lead to overfitting, which may ultimately reduce generalization performance. A common strategy to mitigate this issue is to employ a parameter freezing technique, which helps preserve the original features of the LLM while learning

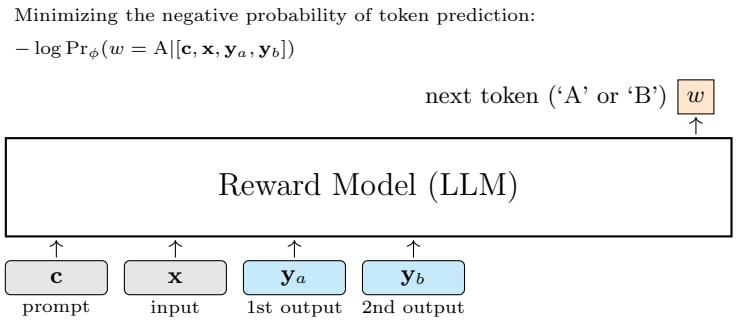
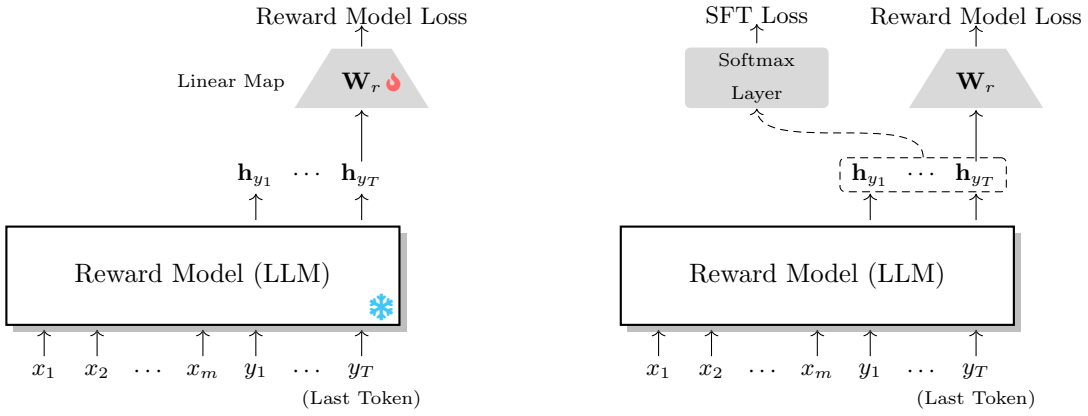


Figure 15: In generative reward models, we use an LLM to predict the label token given a prompt, an input, and a pair of outputs. This model can be trained in the same way as standard LLMs.

the reward model. Another practical approach is to add a regularization term to the preference learning loss function, which helps regulate the features of the LLM (Yang et al., 2024). For example, we can use a simple SFT loss as regularization by adding a term that maximizes the probability of the preferred output $\mathbf{y}_a$ to Eq. (36):

$$\mathcal{L}_{\text{reg}}(\phi) = -\mathbb{E}_{(\mathbf{x},\mathbf{y}_a,\mathbf{y}_b)\sim\mathcal{D}_r}\left[\log \Pr_\phi(\mathbf{y}_a \succ \mathbf{y}_b|\mathbf{x}) + \alpha \log(\Pr_\theta(\mathbf{y}_a|\mathbf{x}))\right] \tag{44}$$

where α is a balancing factor. We further illustrate the parameter freezing and regularization methods in Figure 14. Note that the regularization term applies only to the LLM. That is, when optimizing this term, only the parameters of the LLM are updated, while the parameters of the reward linear map remain fixed. Therefore, in this equation, the parameter associated with the regularization term is θ, which specifically refers to the parameters of the LLM.



(a) Parameter Freezing (b) Regularization

Figure 14: When training reward models, we can use parameter freezing and regularization methods to preserve the features of the LLM and thereby improve reward generalization.

4.1.4 Generative Reward Models

Reward models are typically trained as discriminative models to assign numerical rewards to outputs and classify them as preferred or dispreferred. However, this method does not leverage the text-generation capabilities for which LLMs are fundamentally designed (Zhang et al., 2025b; Wang et al., 2025a). For example, the discriminative reward model cannot perform CoT reasoning. To address this, an LLM can alternatively be employed as a reward model, thus endowing it with the ability to engage in text generation

and reasoning, as depicted in Figure 15. This model works as follows. First, we input a prompt $\mathbf{c}$, along with the tuple $(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b)$, to the LLM. The prompt is a description of the task, as demonstrated in the example below.

> You are given two outputs to an input. Evaluate which output is better based on quality, relevance, and clarity. If the first output is better, return 'A'. If the second output is better, return 'B'.

Then, the LLM predicts subsequent tokens based on this input sequence. Let w be the label token predicted by the LLM and w^* be the annotated preference label. If $w^* = \text{A}$, it indicates a preference for $\mathbf{y}_a$ over $\mathbf{y}_b$; if $w^* = \text{B}$, then $\mathbf{y}_b$ is preferred. Note that here $\mathbf{y}_a$ and $\mathbf{y}_b$ do not have a pre-defined preference relationship as described in Section 3.6. Their relationship is instead represented by the label token.

The loss function can be defined as the log-probability of predicting the annotated preference label:

$$\mathcal{L}_{\text{gen}}(\theta) \quad = \quad -\mathbb{E}_{(\mathbf{c},\mathbf{x},\mathbf{y}_a,\mathbf{y}_b,w^*)\sim\mathcal{D}_r}\big[\log \Pr_\theta(w = w^*|\mathbf{s})\big] \tag{45}$$

where $\mathbf{s}$ denotes the string $[\mathbf{c}, \mathbf{x}, \mathbf{y}_a, \mathbf{y}_b]$[8], and $\Pr_\theta(\cdot)$ denotes the probability of token prediction by the LLM.

Although we discuss methods for training a generative reward model here, an interesting question arises: how can we use the generative reward model in RLHF? We provide some guidance as follows. Specifically, when applying this generative reward model to provide a reward for an input-output pair $(\mathbf{x}', \mathbf{y}')$, we can generate a reference output $\mathbf{y}_{\text{ref}}$ by using the LLM, for example, through greedy search, and concatenate $\mathbf{x}'$, $\mathbf{y}'$ and $\mathbf{y}_{\text{ref}}$ into $\mathbf{s}' = [\mathbf{c}, \mathbf{x}', \mathbf{y}', \mathbf{y}_{\text{ref}}]$. Additionally, to mitigate the positional bias problem (Wang et al., 2024c), we can introduce an alternative input order by transposing the positions of output, i.e., presenting $\mathbf{y}_{\text{ref}}$ before $\mathbf{y}'$, to construct a secondary input string $\mathbf{s}'_T = [\mathbf{c}, \mathbf{x}', \mathbf{y}_{\text{ref}}, \mathbf{y}']$. The reward for $(\mathbf{x}', \mathbf{y}')$ is thus defined as the probability that $\mathbf{y}'$ is preferred over $\mathbf{y}_{\text{ref}}$:

$$R_\phi(\mathbf{x}', \mathbf{y}') \quad = \quad \frac{\Pr_\theta(w = \text{A}|\mathbf{s}') + \Pr_\theta(w = \text{B}|\mathbf{s}'_T)}{2} \tag{46}$$

where the reward ranges from 0 to 1.

To further improve the generative reward model, we can label the relevant explanation to enable the generative reward model to produce a CoT rationale (Zhang et al., 2025a; Wang et al., 2026c). In this case, we use a prompt $\mathbf{c}$ with a CoT rationale generation instruction, as shown below.

> You are given two outputs to a user input. Evaluate which output is better based on quality, relevance, and clarity. If the first output is better, return 'A'. If the second output is better, return 'B'. Note that, before presenting the evaluation results, you should provide a detailed analytical process and reasoning steps.

We can then define a new loss function for training the generative reward model as follows:

$$\mathcal{L}_{\text{gen}}(\theta) \quad = \quad -\mathbb{E}_{(\mathbf{c},\mathbf{x},\mathbf{y}_a,\mathbf{y}_b,\mathbf{rat},w^*)\sim\mathcal{D}_r}\big[\log \Pr_\theta(\mathbf{rat}|\mathbf{s}) + \log \Pr_\theta(w = w^*|[\mathbf{s}, \mathbf{rat}])\big] \tag{47}$$

where $\mathbf{rat}$ is the labeled CoT rationale for generating a preference between $\mathbf{y}_a$ and $\mathbf{y}_b$. In practice, we can obtain the evaluation results by prompting a standard LLM (also known as LLM-as-a-judge), as demonstrated in Section 4.1.1. However, this approach typically underperforms compared to LLMs trained with preference data (Zhang et al., 2025a; Mahan et al., 2024). One reason is that standard LLMs are not fine-tuned specifically for the reward modeling task, and as a result, they may not fully capture the nuanced

[8] In this work, we will use $\mathbf{s}$ interchangeably to refer to either the tuple of a training sample or a string representing that tuple.

decision-making process that aligns better with human preferences. Also, LLMs trained with preference data learn to prioritize outputs based on feedback, enhancing their capability to evaluate outputs according to the desired behaviors and objectives.

Although incorporating CoT rationales into generative reward models improves preference learning, it also introduces a probability degeneration issue in Eq. (46). Specifically, preference token probabilities often become saturated near 0 or 1, resulting in limited reward variance. This issue is particularly challenging for RL algorithms such as GRPO, where relative reward differences are essential for policy optimization. To address this limitation, researchers investigate ranking-based reward construction approaches. The basic idea is to use only the ranking capability of generative reward models to derive rewards for RL (Wang et al., 2026d). Interested readers can refer to this work for further details.

4.1.5 Rubric-based Reward Models

While generative reward models have shown strong performance in preference prediction, they typically produce an overall judgment for each response or response pair. For complex open-ended tasks, such holistic evaluation may be insufficient because response quality often depends on multiple instruction-specific aspects. To address this issue, we can employ rubric-based reward modeling. The basic idea is to introduce a set of task-specific criteria that explicitly guide the reward model on *what aspects to evaluate* when predicting preferences. In practice, these criteria can be organized into different rubric formats, as illustrated below, where the underlined text in each example represents the rubric.

- **Holistic Rubric.** A holistic rubric provides multiple evaluation criteria but asks the reward model to produce a single overall judgment. For example,

> **Reward Model Input:**
> *Input:* Give me three tips to improve my accuracy in solving math problems.
> *Output A:* Improving math accuracy requires careful attention to detail and consistent practice. Here are three tips: practice regularly, understand the concepts, and ...
> *Output B:* Practice more, read carefully, and check your answers.
> Please compare the two outputs based on accuracy, informativeness, clarity, and helpfulness, and give an overall preference.
>
> **Reward Model Output:**
> Output A is better overall because it provides more informative and helpful suggestions. ... Therefore, the reward model assigns a higher reward to Output A and selects Output A as the preferred response.

- **Criterion-wise Scoring Rubric.** A criterion-wise scoring rubric asks the reward model to assign a separate score to each evaluation criterion. These criterion-level scores are then averaged or weighted to obtain the final reward.

Reward Model Input:
Input: Give me three tips to improve my accuracy in solving math problems.
Output A: Improving math accuracy requires careful attention to detail and consistent practice. Here are three tips: practice regularly, understand the concepts, and ...
Output B: Practice more, read carefully, and check your answers.
Please score each output from 1 to 5 on the following criteria: accuracy, informativeness, clarity, and helpfulness.

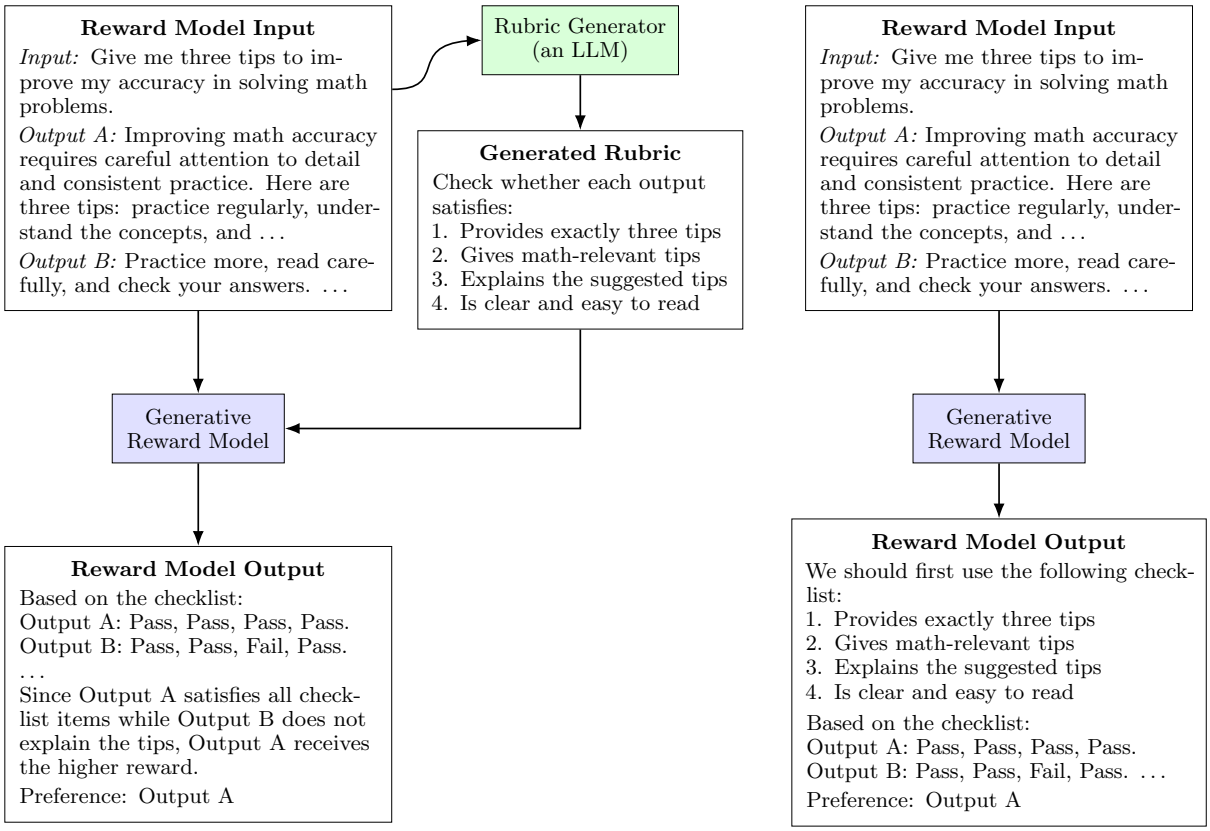
Reward Model Output:
Output A: Accuracy 5/5; Informativeness 5/5; Clarity 4/5; Helpfulness 5/5.
Output B: Accuracy 4/5; Informativeness 2/5; Clarity 4/5; Helpfulness 3/5. ... After aggregating the criterion-level scores, Output A receives the higher overall score and is preferred over Output B.

- **Checklist Rubric.** A checklist rubric decomposes evaluation into atomic and verifiable conditions. Each condition is evaluated independently, typically using binary labels such as *Pass* or *Fail*.

Reward Model Input:
Input: Give me three tips to improve my accuracy in solving math problems.
Output A: Improving math accuracy requires careful attention to detail and consistent practice. Here are three tips: practice regularly, understand the concepts, and ...
Output B: Practice more, read carefully, and check your answers.
Please check whether each output satisfies the following requirements:
1. Does the response provide exactly three tips?
2. Are the tips relevant to improving math accuracy?
3. Does the response explain the suggested tips?
4. Is the response clear and easy to understand?

Reward Model Output:
Output A: Pass, Pass, Pass, Pass.
Output B: Pass, Pass, Fail, Pass. ... Since Output A satisfies all checklist items while Output B does not explain the tips, Output A is assigned the higher reward.

- **Hierarchical Rubric.** A hierarchical rubric organizes evaluation criteria into multiple levels. The reward model first evaluates candidate outputs using lower-level criteria and then aggregates these results into higher-level dimensions to obtain the final reward.



(a) Using an external LLM to generate the rubric **(b) GRM self-generates the rubric**

Figure 16: Two commonly used approaches for automatic rubric generation. We use checklist rubrics as an illustrative example.

Reward Model Input:
Input: Give me three tips to improve my accuracy in solving math problems.
Output A: Improving math accuracy requires careful attention to detail and consistent practice. Here are three tips: practice regularly, understand the concepts, and ...
Output B: Practice more, read carefully, and check your answers.
Please evaluate the outputs using the following hierarchical criteria:
Content Quality: Accuracy, Informativeness
Presentation Quality: Clarity, Helpfulness

Reward Model Output:
Output A: Content Quality 5/5; Presentation Quality 4.5/5.
Output B: Content Quality 3/5; Presentation Quality 3.5/5. ... By aggregating the lower-level evaluations into the two higher-level dimensions, Output A obtains the stronger final reward and is selected as the better response.

It is worth noting that reasoning can also be incorporated into rubric-based evaluation. For example, in a checklist-style rubric, the reward model can be asked to reason about each criterion before producing the corresponding Pass/Fail judgment. Additionally, although we use pairwise evaluation, i.e., taking $(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b)$

as reward model input, as the running example, rubric-based reward modeling is equally applicable to pointwise evaluation, i.e., taking $(\mathbf{x}, \mathbf{y})$ as reward model input.

The quality of the rubric is crucial to the performance of rubric-based reward models. As a result, recent work has devoted increasing attention to constructing high-quality rubrics for reward modeling. Similar to many other components in machine learning, rubric acquisition generally follows two approaches: manual design and automatic generation.

For manual design, one straightforward approach is to recruit human experts to write rubrics based on their domain knowledge and task requirements. However, it is often difficult to design a rubric that comprehensively covers the diverse cases that may arise across different inputs and tasks. This is because evaluation criteria that are appropriate for one case may be insufficient for another. Additionally, manually designed rubrics inherit the prompt sensitivity of LLM-based evaluation: even when the underlying evaluation dimensions remain the same, small differences in wording can lead to noticeably different results.

One promising strategy is to generate rubrics dynamically based on the input. As shown in Figure 16, existing approaches generally fall into two categories. The first uses an external LLM to generate task-specific rubrics from the input and evaluation requirements, which are then provided to the reward model for preference prediction. The second allows the generative reward model to generate its own rubrics for the current input and then evaluate candidate responses according to these self-generated criteria.

Both approaches formulate rubric generation as an explicit task. This naturally leads to another idea: can we optimize the rubric generation process itself to produce better rubrics? The answer is yes. Rubrics generated directly by LLMs may suffer from limited coverage, redundant criteria, or preference misalignment (Liu et al., 2026a; Shen et al., 2026). To improve rubric quality, we can explicitly refine the generated criteria. For example, OpenRubrics generates rubrics by contrasting preferred and rejected responses to identify more discriminative rules and principles (Liu et al., 2026a). We can further learn the rubric generation process itself through RL training (Xu et al., 2026). More recent studies go one step further by allowing rubrics to evolve with the policy, so that the evaluation criteria continue to capture new weaknesses as the model improves (Rezaei et al., 2025; Ding et al., 2026; Yu et al., 2026a). Rubric optimization is becoming an active research direction, and interested readers can refer to the aforementioned works for further details.

4.1.6 Reward Model Evaluation

After training a reward model, an important question is *how to evaluate whether the learned reward function can accurately capture human preferences*. Unlike conventional supervised models that directly predict explicit labels, reward models learn to assign scores that reflect the relative quality of different outputs. As a result, existing evaluation approaches mainly focus on measuring the preference modeling ability of reward models or their effectiveness in downstream RL optimization. Figure 17 compares three mainstream evaluation paradigms, including RL-based evaluation, pairwise ranking evaluation, and listwise ranking evaluation. We summarize these commonly used evaluation approaches as follows.

- **RL-based Evaluation.** A straightforward approach to evaluate a reward model is to measure its effectiveness in downstream RL training. Specifically, given multiple reward models $\{\mathcal{M}_r^1, \cdots, \mathcal{M}_r^k\}$, we use each reward model to provide reward signals for training a policy model while keeping the training data, RL algorithm, and hyperparameters identical (Wang et al., 2026b; Frick et al., 2025). After RL training, we obtain a set of optimized policies $\{\mathcal{P}_1, \cdots, \mathcal{P}_k\}$ and evaluate them on human preference benchmarks or downstream tasks. The performance of each policy then serves as an indirect measure of the quality of its corresponding reward model. In this way, we consider that a high-quality reward model should provide more reliable reward signals, enabling the policy to achieve better final performance.
- **Pairwise Ranking Evaluation.** Although RL-based evaluation directly measures whether a reward model can improve downstream policy optimization, it suffers from two limitations. First, it introduces significant evaluation costs. Since RL training requires substantial computational

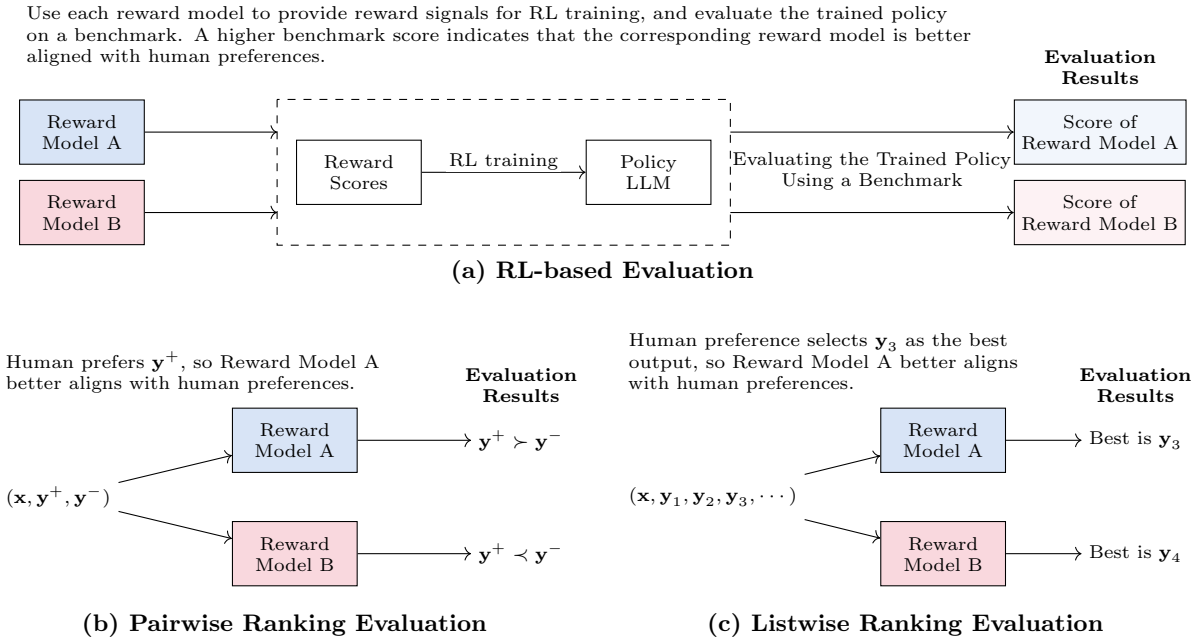


Figure 17: Commonly used reward model evaluation approaches. For the RL-based evaluation approach, the reward model is used to provide reward signals for policy optimization, and its quality is measured by the downstream performance of the optimized policy. For the pairwise evaluation approach, the reward model compares a preferred output with a dispreferred output, and its prediction is checked against human preference annotations. For the listwise evaluation approach, the reward model selects the best output from multiple candidates, and the selected output is compared with the human-preferred one.

resources and time, it is difficult to efficiently compare different reward models. Second, the evaluation results can be sensitive to other factors in the RL pipeline, such as the choice of RL algorithms, which may introduce additional variations beyond the quality of the reward model itself. To address these limitations, a more efficient approach is to directly evaluate the preference ranking ability of reward models. Given a prompt $\mathbf{x}$ and two candidate outputs $\mathbf{y}^+$ and $\mathbf{y}^-$, where $\mathbf{y}^+$ is preferred over $\mathbf{y}^-$ according to human preference annotations, the reward model predicts their relative preference. Specifically, for discriminative reward models, we compare the predicted scores of the two outputs. If the reward model assigns a higher score to $\mathbf{y}^+$, its prediction is consistent with human preference. For generative reward models, the model directly selects $\mathbf{y}^+$ as the preferred output based on its generated preference. Otherwise, the prediction is considered incorrect. By constructing a large number of pairwise ranking samples, we can evaluate the reward model using ranking accuracy:

$$\text{Acc} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\left[R_\theta(\mathbf{x}_i, \mathbf{y}_i^+) > R_\theta(\mathbf{x}_i, \mathbf{y}_i^-)\right] \tag{48}$$

where N denotes the number of evaluation samples. Examples of this evaluation approach include RewardBench (Lambert et al., 2025; Malik et al., 2026), RM-Bench (Liu et al., 2025b), RMB (Zhou et al., 2025), and JudgeBench (Tan et al., 2025).

- **Listwise Ranking Evaluation.** In practical alignment scenarios, the reward model often needs to select the best output from multiple candidates, such as in best-of-n sampling or reranking. Therefore, listwise ranking evaluation measures whether a reward model can produce a consistent ranking over a set of candidate outputs. Specifically, given a prompt $\mathbf{x}$ and a candidate output set: $\mathcal{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_n\}$, the reward model assigns scores to all candidates. Based on the computed

rewards, the reward model selects the highest-scoring output:

$$\mathbf{y}_{\text{best}} = \arg\max_{\mathbf{y}_i \in \mathcal{Y}} R_\theta(\mathbf{x}, \mathbf{y}_i) \tag{49}$$

The selected output is then compared with the human-preferred output to evaluate how well the reward model captures human preferences. We take PPE (Frick et al., 2025) as an example to illustrate this evaluation approach. Specifically, we can construct each evaluation sample with multiple candidate outputs, where the quality of each output can be automatically verified. For example, in mathematical reasoning tasks, we can use an answer verifier to determine whether each generated solution is correct. Given a set of candidates $\mathcal{Y}$, the verification result of each output can be represented as:

$$v_i = \text{Verify}(\mathbf{x}, \mathbf{y}_i), \quad v_i \in \{0, 1\} \tag{50}$$

where $v_i = 1$ indicates that the output is correct. If the output selected by the reward model is verified as correct: $\text{Verify}(\mathbf{x}, \mathbf{y}_{\text{best}}) = 1$, the reward model is considered successful on this instance. The final performance is measured by the selection accuracy over all evaluation instances:

$$\text{Acc} = \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}(\text{Verify}(\mathbf{x}_j, \mathbf{y}^j_{\text{best}}) = 1) \tag{51}$$

where N denotes the number of evaluation instances.

4.2 Better Advantage Estimation

Accurate advantage estimation is critical in RL, particularly when optimizing the policy model to truly reflect the potential benefits of different actions (or tokens in training LLMs). In this subsection, we will delve into methods for improving advantage estimation, providing more accurate advantages in the process of optimizing the policy model.

4.2.1 Temporal Difference-based Advantage Estimation

Let us first review the Monte Carlo-based advantage estimation. As discussed in Section 3.3, outputs are sampled and their values computed. The advantage can be obtained by comparing the actual received rewards with the predicted values at time step t:

$$A_t = \sum_{k=t}^{T} r_k - V_t \tag{52}$$

While the Monte Carlo-based estimation of advantage provides a relatively stable training process, there are two main challenges associated with it:

- Sampling an entire output for each input is necessary. In some cases where immediate rewards are available, it could be more efficient to update the policy model with partial output. Unfortunately, when using Eq. (52) for advantage estimation, this becomes unfeasible. This is because we must compute the term $\sum_{k=t}^{T} r_k$, which requires the rewards of the entire output.
- This method still results in high variance. Since a single sample operation might be influenced by random factors, the estimated results may not be stable, as illustrated in Figure 18.

To address these challenges, an alternative approach is the temporal difference-based advantage estimation (Sutton, 1988), which allows for estimating the advantage using incomplete outputs. More specifically, this method computes the difference between the predicted values at consecutive time steps (i.e., current and next steps), adjusting the advantage estimate based on new information as it becomes available, thus

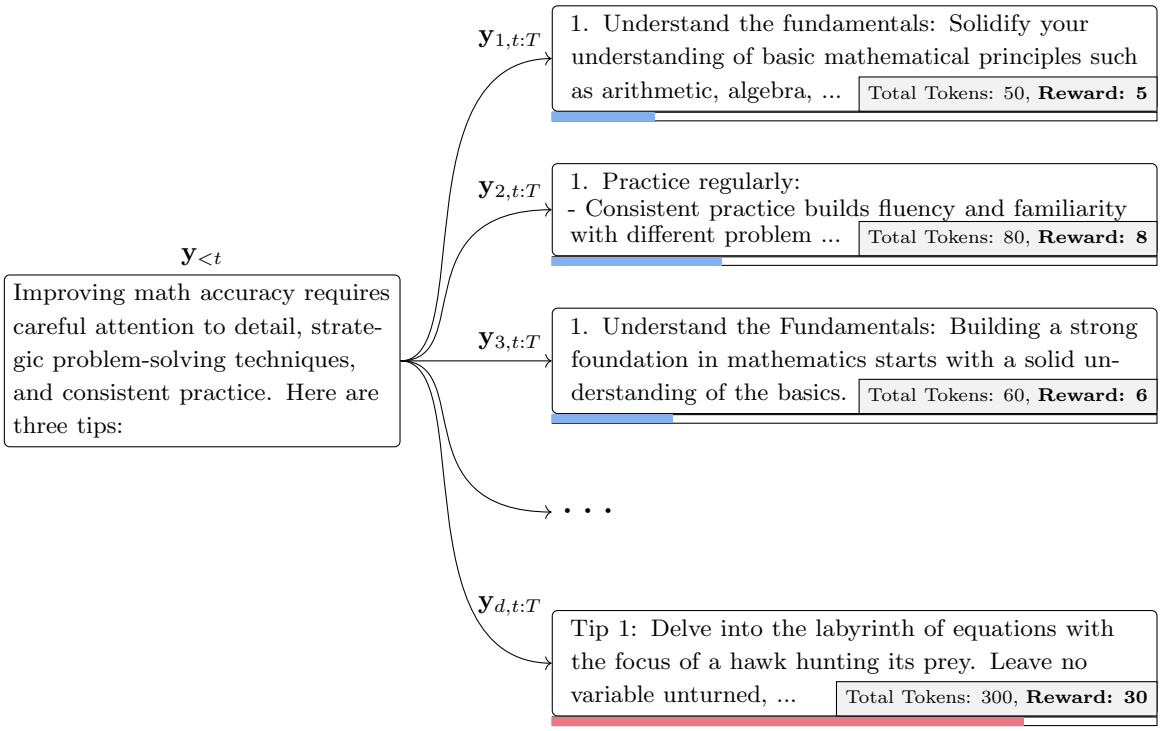


Figure 18: Illustration of the issue of high variance in Monte Carlo-based advantage estimation, using a length-based reward function as described in Eq. (13). Typically, if the value model is well-trained, it would predict an average V_t of 6 at time step t, based on most outputs having around 60 tokens in length. However, for outlier outputs like $\mathbf{y}_{d,t:T}$, which might extend up to 300 tokens, the advantage estimation can exhibit high variance. For instance, the advantage for such an outlier could be computed as $A_t(\mathbf{x}, \mathbf{y}_{d,<t}, y_{d,t}) = \sum_{k=t}^{T} r_k - V_t = 30 - 6 = 24$, a stark contrast to another output, say $\mathbf{y}_{1,t:T}$, which is closer to the average length, where the advantage might be $A_t(\mathbf{x}, \mathbf{y}_{1,<t}, y_{1,t}) = -1$. This discrepancy leads to high gradient variance, potentially destabilizing the learning process.

reducing the dependency on the entire output. In a practical implementation, the temporal difference-based advantage estimation can be given by

$$A_t^{\text{TD}} \quad = \quad r_t + \gamma V_{t+1} - V_t \tag{53}$$

By focusing on the current and next steps, temporal difference-based advantage estimation minimizes the influence of distant future events. This leads to lower variance in optimizing the policy model and improves the stability of policy training.

4.2.2 Generalized Advantage Estimation

While temporal difference-based estimation effectively reduces variance, it can also increase estimation bias due to its heavy reliance on the predictions of the value model. By contrast, one effective method is generalized advantage estimation (GAE), which is one of the most popular advantage estimation methods used in LLMs and other fields (Schulman et al., 2016). This method mainly refines the advantage estimation by using exponentially weighted averages of temporal-difference residuals across multiple future steps. More specifically, consider the temporal-difference residual δ_t at time step t, defined as

$$\delta_t \quad = \quad r_t + \gamma V_{t+1} - V_t \tag{54}$$

GAE introduces parameters that dynamically balance the trade-off between bias and variance. This basic idea is to use the weighted sum of multi-step temporal-difference residuals as an estimation of the advantage,

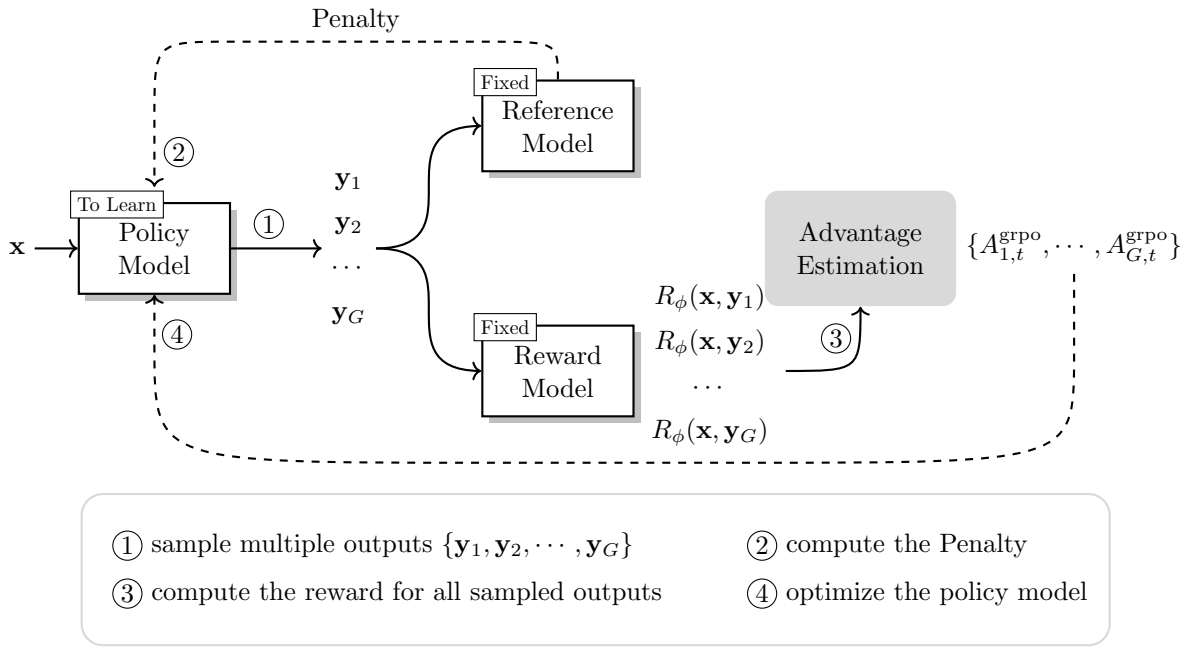


Figure 19: Group relative policy optimization. Compared to PPO, it foregoes the value model by estimating advantages directly from the rewards of a group of outputs. This method can simplify the training process and reduce the computational resources.

combining the advantages of Monte Carlo (high variance and low bias) and temporal difference methods (low variance and high bias). This combination can be expressed as

$$A_t^{\text{gae}} = \sum_{n=0}^{T-t} (\gamma\lambda)^n \delta_{t+n} \tag{55}$$

We can recursively develop the above model:

$$A_t^{\text{gae}} = \delta_t + \gamma\lambda A_{t+1}^{\text{gae}} \tag{56}$$

This recursive formulation shows that the current advantage estimate incorporates not only the immediate temporal difference error at time t but also the estimated advantages of future time steps, weighted and adjusted by the parameters γ and λ. This method effectively combines the strengths of both Monte Carlo and temporal difference methods, leading to a more stable and efficient training process.

4.2.3 Group Relative Policy Optimization

Although the methods discussed in the previous subsections effectively estimate advantage during the learning process, they all rely on a value model that is typically trained concurrently with the policy model. This dependence not only complicates the training process but also requires significant computational resources when applied to training LLMs. Given this, a natural idea arises: could we use more lightweight methods to compute the advantage without relying on the value model? By doing so, we could forego the value model component, thereby improving efficiency.

In fact, this is entirely feasible, and one example of such an approach is Shao et al. (2024)'s approach, called group relative policy optimization or GRPO for short. In the GRPO approach, the advantage is computed based on the relative rewards of the outputs within each group. More specifically, given an input $\mathbf{x}$, GRPO samples a group of outputs $\mathbf{Y} = \{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_G\}$. Then, the rewards for each output are computed using a

reward model or a reward function, denoted as $\mathbf{R} = \{R(\mathbf{x}, \mathbf{y}_1), R(\mathbf{x}, \mathbf{y}_2), \ldots, R(\mathbf{x}, \mathbf{y}_G)\}$. The advantage of the i-th output can be computed through a relative reward in comparison to the other outputs:

$$A_{i,t}^{\text{grpo}} = \frac{R(\mathbf{x}, \mathbf{y}_i) - \text{Mean}(\mathbf{R})}{\text{Std}(\mathbf{R})} \tag{57}$$

where $\text{Mean}(\cdot)$ and $\text{Std}(\cdot)$ represent the mean and standard deviation functions, respectively. Note that the advantage computation used here differs slightly from that in PPO. In this case, this computed advantage is applied to each time step, whereas in the traditional PPO, we separately compute an advantage for each time step. At this point, we can also use the process reward model to supplement the advantage computation, allowing us to compute an advantage specific to each time step. A simple way to achieve this is by implementing Eq. (57) at each time step, where the rewards are computed using the process reward model. As a result, the objective for GRPO can be defined according to Eq. (32):

$$\mathcal{L}_g(\theta) \quad = \quad -\mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x} \frac{1}{G} \sum_{i=1}^{G} \left[\sum_{t=1}^{T} \text{Clip}\Big(\frac{\text{Pr}_\theta(y_{i,t}|\mathbf{x}, \mathbf{y}_{i,<t})}{\text{Pr}_{\theta_{\text{ref}}}(y_{i,t}|\mathbf{x}, \mathbf{y}_{i,<t})} A_{i,t}^{\text{grpo}}\Big) - \beta\text{Penalty} \right] \tag{58}$$

In addition to optimizing the advantage computation, GRPO modifies the penalty term to enhance the optimization objective further. However, since our focus here is on advantage estimation, we will not delve into its details. Interested readers can refer to the GRPO paper for further details. The workflow of GRPO is illustrated in Figure 19, showcasing how these modifications integrate into the training process.

Moreover, there are other methods like GRPO that design advantage estimation without relying on a value model for training LLMs, such as those discussed in Li et al. (2024) and Hu (2025).

4.3 Efficient RL Methods

Efficiency is a critical consideration for many practical applications of RL, and it becomes even more significant in LLM training due to their vast number of parameters. For example, we might wish to train LLMs using RL, given memory and time constraints. In practice, the efficiency of RL is not a single issue but encompasses a wide range of challenges. While these challenges can be categorized in various ways in the existing literature, we focus on two fundamental aspects: *time* and *space* efficiency, which are commonly considered in efficiency-related problems. For these two efficiency problems, we aim to achieve the desired RL performance with the least amount of time and memory required.

In this section, we will not discuss all the issues related to the efficiency of RL, which is an extensive topic. Instead, we will focus on the commonly used efficient methods for training LLMs with RL. Some of these methods refine the sampling process, while others aim to eliminate certain components, such as the reward model, and utilize alternative lightweight methods in their place. Nonetheless, although these efficient methods are suggested for training LLMs, they are rather general and can be utilized in other RL scenarios.

4.3.1 Dynamic Sampling

As mentioned in Section 3.1, training LLMs with RL typically requires sampling for each sample $\mathbf{x}$ in $\mathcal{S}_x$, which introduces significant computational time overhead. This becomes particularly challenging in large-scale RL scenarios, where thousands of training steps need to be conducted, such as in DeepSeek-R1 (Guo et al., 2025).

From the perspective of LLM inference, there are many methods to reduce the time overhead of sampling: since the sampling process is implemented by LLM inference, we have reason to believe that any method that accelerates inference can also be applied to reduce the time required for sampling. Examples include KV-Cache (Pope et al., 2023), quantization (Zhao et al., 2024), and speculative decoding (Chen et al., 2023b). However, in this section, we will not discuss these methods in detail, as there are numerous approaches, each addressing different aspects of LLM inference. We refer the interested readers to these

papers for more details. Instead, we will focus on one particular method that aims to reduce sampling overhead from the perspective of optimizing RL for training LLMs.

Before discussing specific methods, let us first review the purpose of sampling. Given a sample $\mathbf{x}$, the goal of sampling is to explore a better output $\mathbf{y}$ from the policy model. This process allows us to evaluate different possible outputs in order to find those that lead to improved outcomes, enabling the model to make more informed decisions and optimize its performance on the state. Unlike typical RL scenarios, where policy models may start with limited information or random initialization, the policy model in LLM training is usually quite advanced. These LLMs often begin as pre-trained models equipped with substantial knowledge acquired through pre-training and SFT. As a result, for certain samples, we can consider that the policy model may already perform well without the need for further exploration and optimization. This can be viewed through the lens of the classic RL dilemma of exploration versus exploitation (Sutton & Barto, 2018). In such cases, exploration may be less necessary, and exploiting the knowledge already embedded in the pre-trained model could be more effective. Thus, a balance must be struck between exploring new possibilities and leveraging the pre-existing knowledge of the policy model to maximize efficiency.

To achieve this balance between exploration and exploitation, one simple and straightforward approach to improving the efficiency of sampling is to identify the samples that are required to explore and perform sampling only for those samples. Given an input-only dataset $\mathcal{S}_x = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$, where N denotes the size of the dataset, we can use a reward model to evaluate the need for exploration by the policy model. First, we generate $\hat{\mathbf{y}}$ for each $\mathbf{x}$ using greedy search. Then, by taking $\mathbf{x}$ and $\hat{\mathbf{y}}$ as input to the reward model, the reward for the k-th sample $\mathbf{x}_k$ is computed as $R_\phi(\mathbf{x}_k, \hat{\mathbf{y}}_k)$.

At this point, we can directly use the reward value to determine whether the policy model requires further exploration and optimization for the sample. Specifically, we set a threshold r_{upper}. If the reward for a sample exceeds r_{upper}, we can consider that no further sampling and optimization is required. Otherwise, exploration and optimization are required. This approach enables selective sampling and optimization, improving the efficiency of the overall process by focusing resources on the most promising samples that are likely to benefit from further exploration (Wang et al., 2024b). A challenge in implementation is determining the value of r_{upper}. Typically, we set it as the average reward:

$$r_{\text{upper}} = \frac{1}{N} \sum_{i=1}^{N} R_\phi(\mathbf{x}_i, \hat{\mathbf{y}}_i) \tag{59}$$

A similar dynamic sampling strategy, known as prioritized sampling, has proven effective in large-scale RL scenarios (Team et al., 2025b). However, we must consider another important factor in dynamic sampling. For samples with very low rewards, the policy model may lack the capacity to learn and optimize effectively. Excessive exploration of such samples can be inefficient. Consequently, we can introduce an additional threshold r_{lower} to eliminate unnecessary exploration of these low-reward samples, thereby enhancing efficiency. In this case, the policy model can concentrate its exploration and optimization efforts on a select group of samples, avoiding wasted sampling time on those that are unlikely to explore a more optimal output (Li et al., 2025a).

A concept closely related to the discussion here is sample efficiency. This topic has been widely discussed in recent literature. For example, some research on knowledge distillation seeks to select a smaller subset of samples to achieve more optimal performance (Wang et al., 2021). More recently, much work on instruction sample selection in SFT aims to improve SFT efficiency by learning from a small number of instruction samples (Chen et al., 2024).

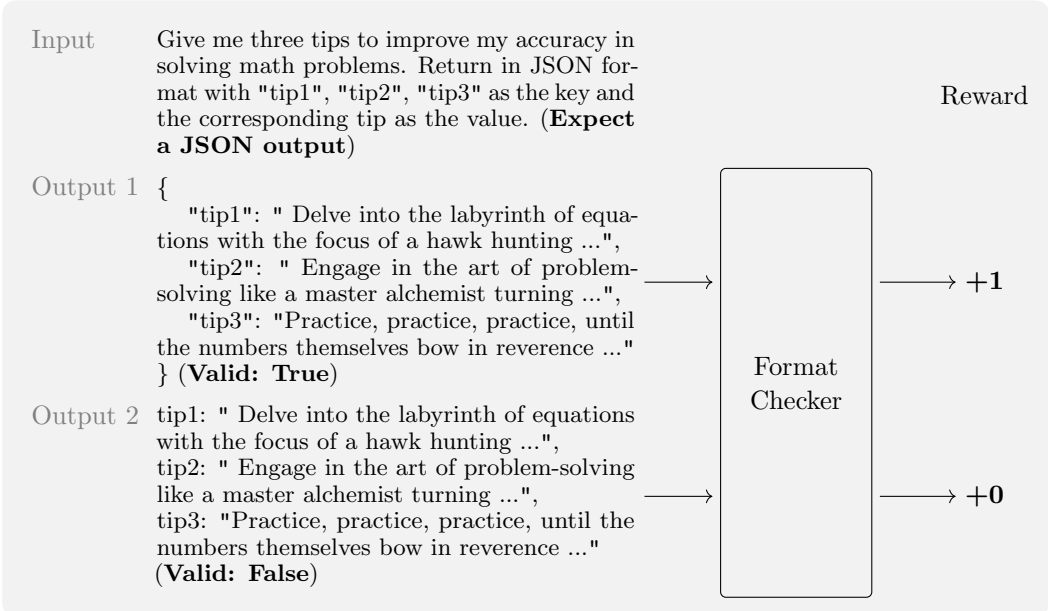
4.3.2 Lightweight Reward Methods

An additional computational overhead in training LLMs with RL is the frequent need to call the reward model to compute rewards. Furthermore, to ensure the reliability and generalizability of the reward model,

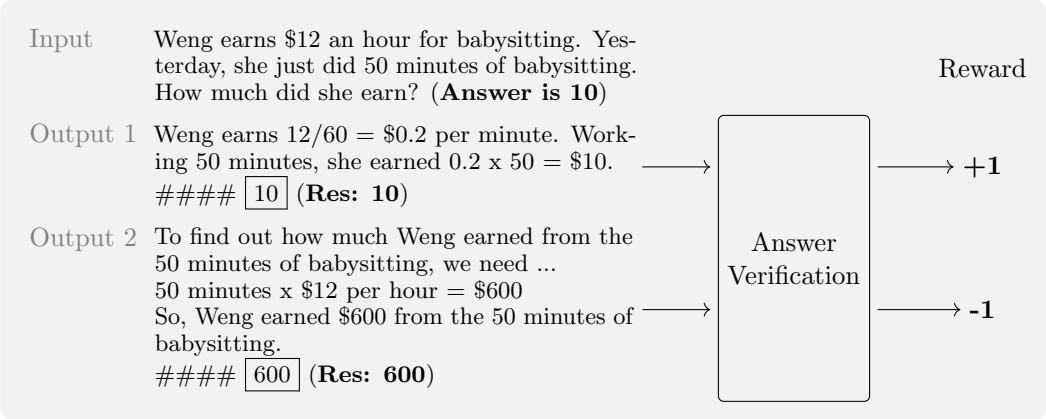
we often scale it to a large size (Gao et al., 2023), which increases the computational time for reward computations.

One approach for mitigating this issue is to explore rule-based rewards, for example, as discussed in Section 3.1, the length of the outputs could be utilized as a reward. Instead of relying on large-scale, resource-intensive models, rule-based rewards assign rewards using predefined and task-specific rules. These rules, often derived from expert knowledge or task-specific requirements, are computationally inexpensive and provide fast, effective feedback to the policy model. By replacing or complementing traditional reward models with rule-based approaches, we can reduce both the time and computational cost associated with reward computations while still offering meaningful guidance for the learning process. This approach is also based on the understanding that, in some tasks, human preferences are simple to describe and do not require the complexity of training a reward model. Instead, we can use rules to express these preferences efficiently.

Here, in addition to the previously mentioned length-based rewards, we further demonstrate two examples of rule-based rewards. First, considering the output format, we can design rewards that encourage the policy model to generate outputs adhering to specific formats. For example, in response to the input "Give me three tips to improve my accuracy in solving math problems," we can assign rewards based on whether the output can be parsed as JSON correctly and include "tip1", "tip2", and "tip3" as the keys.



Similarly, in mathematical reasoning tasks, where the desired outcome is a correct final answer, we can design a rule to verify the correctness of the answer and provide rewards based on whether the result is correct (Havrilla et al., 2024; Shao et al., 2024).



In addition to improving computational efficiency, another benefit of using rule-based rewards is their stability. Compared to reward models, rule-based rewards are less prone to issues such as reward hacking. As the rules are predefined and not subject to the same complex training process as traditional models, the risk of misalignment between the optimization objectives of the policy model and the oracle rewards is reduced. This stability makes rule-based rewards a more predictable and reliable approach, ensuring that the learning process of the policy model is less influenced by prediction errors or biases that may arise from the reward model. Furthermore, recent research has shown that employing rule-based rewards can significantly enhance the reasoning capabilities of LLMs through RL, underscoring its practical effectiveness (Guo et al., 2025; Xie et al., 2025).

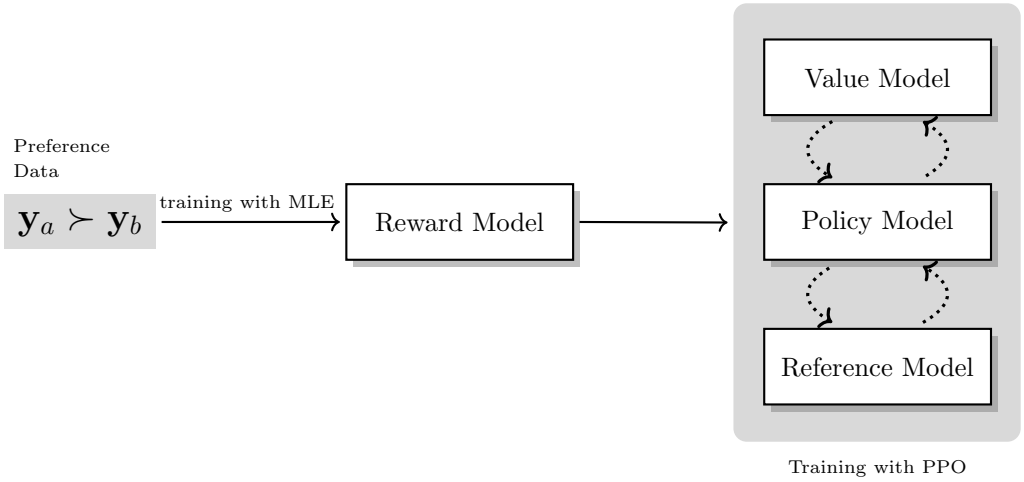
Another lightweight reward method is to use a reward model with fewer parameters, which would reduce both computational time and resource usage. There are several methods to achieve this. For example, given that reward models fundamentally utilize an LLM, many efficient methods, such as Mixture of Experts (MoE) (Masoudnia & Ebrahimpour, 2014) and model pruning (Sun et al., 2024a), originally developed for LLMs can be adapted to the reward model to improve its efficiency. Furthermore, knowledge distillation techniques offer a pathway to construct a smaller reward model that learns to emulate the behavior of a larger one (Wang et al., 2023a). These approaches not only improve the efficiency of the reward model but also maintain its ability to deliver accurate rewards.

4.4 Direct Preference Optimization

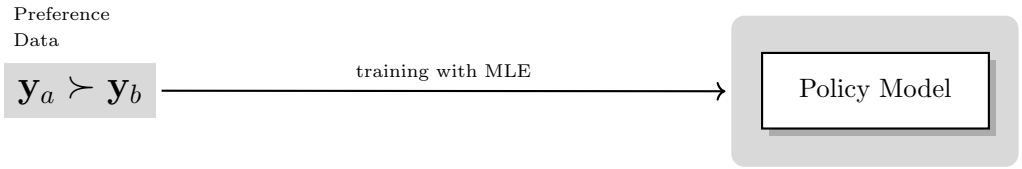
Although learning reward models is a standard step in RL, it makes the entire training process much more complex than supervised training. Training a reliable reward model is itself not an easy task and a poorly trained reward model can greatly affect the outcome of policy learning. We now consider an alternative method, called direct preference optimization (DPO), which simplifies the training framework by eliminating the need to explicitly model rewards (Rafailov et al., 2023). This method directly optimizes the policy model based on user preferences rather than developing a separate reward model. As a result, we can achieve human preference alignment in a supervised learning-like fashion. Figure 20 shows a comparison of the standard PPO method and the DPO method.

DPO simplifies the RL process for LLMs by utilizing a straightforward cross-entropy loss, which streamlines the learning process and enhances its stability. Rather than delving into the detailed derivation of the DPO loss function, we will discuss its optimization objective from the perspective of optimizing an LLM. The loss function derived from a Bradley-Terry reward model can be given by

$$\mathcal{L}_{\text{dpo}}(\theta) \quad = \quad -\mathbb{E}_{(\mathbf{x},\mathbf{y}_a,\mathbf{y}_b)\sim\mathcal{D}_r}\big[\log \text{Sigmoid}\big(\beta \log \frac{\Pr_\theta(\mathbf{y}_a|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a|\mathbf{x})} - \beta \log \frac{\Pr_\theta(\mathbf{y}_b|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b|\mathbf{x})}\big)\big] \tag{60}$$



(a) Training LLMs with PPO



(b) Training LLMs with DPO

Figure 20: Standard PPO vs. DPO in training LLMs. In PPO, the human preference data is used to train a reward model, which is then employed to train the policy and the value function. In DPO, the use of human preference data is more direct, and the policy is trained on this data without the need for reward model training.

This loss function utilizes the implicit reward for DPO training, which replaces the need for an external reward model. The implicit reward is computed as the log-ratio of probabilities:

$$R(\mathbf{x}, \mathbf{y}) = \beta \log \frac{\Pr_\theta(\mathbf{y}|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}|\mathbf{x})} \tag{61}$$

Here, we can consider that DPO directly models human feedback through the relative likelihood of outputs under the policy and reference models. This method provides a more straightforward approach than traditional methods that use a separate reward model and then apply RL algorithms like PPO to adjust the probabilities of sampled outputs. In this way, we can further understand the optimization objective of DPO by leveraging the hypothesis space as mentioned in Section 3.1. In practice, as illustrated in Figure 21, we can see DPO as reranking in the hypothesis space through the Bradley-Terry model approach. Specifically, this method increases the probabilities of preferred outputs in preference data while decreasing those of dispreferred ones. Similar to TRPO, DPO also incorporates a penalty derived from the reference model, which ensures that updates remain within a trusted behavior space for the policy model.

However, there are two sides to every coin. While DPO simplifies the RL process, it also introduces limitations. Notably, since DPO eliminates the exploration phase during training, its potential peak performance is generally considered lower compared to RL-based methods like PPO, which incorporate exploration to discover more effective policies. In other words, the performance of DPO is inherently bounded by the labeled preferred outputs in the preference data. To address this limitation, current approaches focus on ensuring high-quality preferred outputs, either by employing advanced LLMs like GPT-4 or through human labeling (Cui et al., 2024a; Morimura et al., 2024).

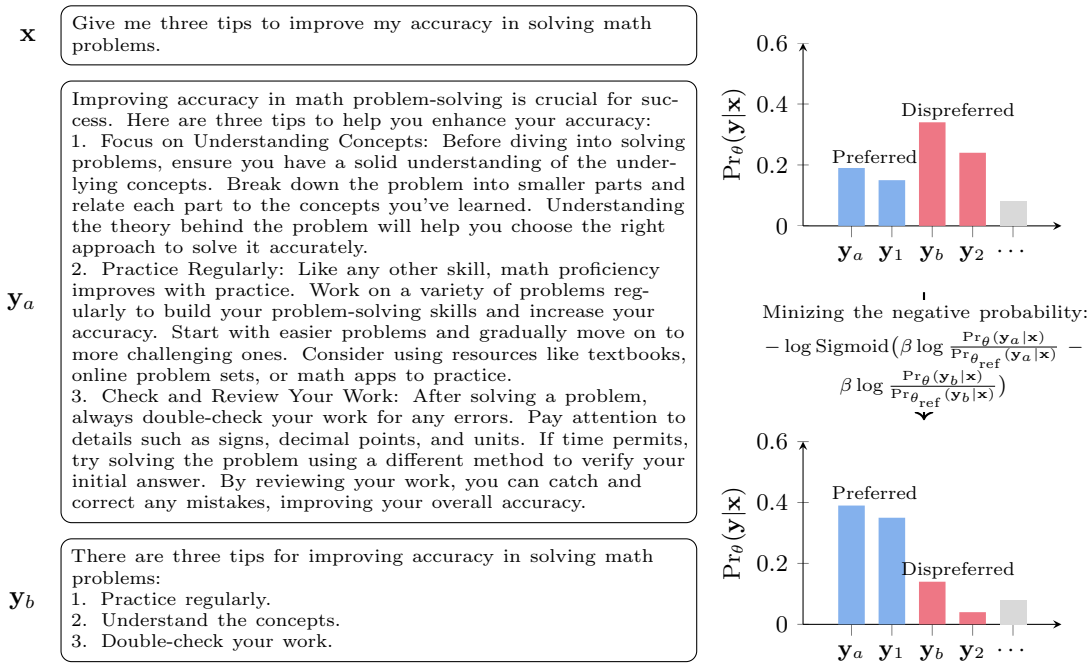


Figure 21: Illustration of the optimization objective of DPO from a hypothesis space perspective. The right-top chart shows the probability distributions over different hypotheses before optimization, where $\mathbf{y}_a$ is the preferred output and $\mathbf{y}_b$ is the dispreferred output. The right-bottom chart shows the probabilities after applying DPO, where the likelihood of the preferred output increases and that of dispreferred outputs decreases. Note that during this optimization process, outputs similar to the preferred output $\mathbf{y}_a$ (such as $\mathbf{y}_1$) will obtain an increased probability, while those similar to the less preferred output $\mathbf{y}_b$ (such as $\mathbf{y}_2$) will experience a decreased probability, due to the generalization of the model.

Another limitation associated with DPO is over-optimization. Since DPO employs the Bradley-Terry model for modeling preferences, it can suffer from over-optimization, such as length exploitation, where a longer output might be mistakenly deemed as more aligned with human preferences (Singhal et al., 2023; Wang et al., 2023b). Many efforts have been made to address this issue and propose different variants of DPO, as detailed in Table 3. Note that here we only provide an introduction to their optimization objectives. For more discussions on these variants, interested readers can refer to the related papers.

5 RL for LLM Reasoning

So far, our discussion has mainly focused on various aspects of using and improving RL for training LLMs. The methods mentioned can be easily adapted to a wide range of scenarios where the correctness of an output can be examined by checking whether the desired result is included. For example, in the task of calculating a mathematical expression, a reward model can provide positive feedback if the answer is correct and negative feedback if the answer is wrong. However, in many problems that require complex reasoning, simply examining the correctness of the final answer is insufficient for learning. Imagine a student who is only given the final answer to a challenging math problem. Knowing whether the final answer is right or wrong does not help the student figure out where they went wrong and how to calculate the correct answer. A better approach would be to guide the student with a step-by-step breakdown of the problem-solving process and encourage understanding of the underlying concepts and logic behind these steps. To address this, researchers also explore the potential of RL to teach LLMs not just to generate correct answers but to develop and present coherent, step-by-step reasoning that aligns with human cognitive processes. In this

Method	Objective
IPO (Azar et al., 2024)	$\left(\log \frac{\Pr_\theta(\mathbf{y}_a\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a\|\mathbf{x})} - \log \frac{\Pr_\theta(\mathbf{y}_b\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b\|\mathbf{x})} - \frac{1}{2\tau}\right)^2$
CPO (Xu et al., 2024)	$-\log \text{Sigmoid}\left(\beta \log \Pr_\theta(\mathbf{y}_a\|\mathbf{x}) - \beta \log \Pr_\theta(\mathbf{y}_b\|\mathbf{x})\right) - \lambda \log \Pr_\theta(\mathbf{y}_a\|\mathbf{x})$
KTO (Ethayarajh et al., 2024)	$-\lambda_a \text{Sigmoid}\left(\beta \log \frac{\Pr_\theta(\mathbf{y}_a\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a\|\mathbf{x})} - z_{\theta_{\text{ref}}}\right) + \lambda_b \text{Sigmoid}\left(z_{\theta_{\text{ref}}} - \beta \log \frac{\Pr_\theta(\mathbf{y}_b\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b\|\mathbf{x})}\right)$ where $z_{\theta_{\text{ref}}} = \mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{S}}\left[\beta \text{KL}\left(\Pr_\theta(\mathbf{y}\|\mathbf{x})\|\Pr_{\theta_{\text{ref}}}(\mathbf{y}\|\mathbf{x})\right)\right]$
ORPO (Hong et al., 2024)	$-\log p_\theta(\mathbf{y}_a\|\mathbf{x}) - \lambda \log \text{Sigmoid}\left(\log \frac{p_\theta(\mathbf{y}_a\|\mathbf{x})}{1-p_\theta(\mathbf{y}_a\|\mathbf{x})} - \log \frac{p_\theta(\mathbf{y}_b\|\mathbf{x})}{1-p_\theta(\mathbf{y}_b\|\mathbf{x})}\right)$ where $p_\theta(\mathbf{y}\|\mathbf{x}) = e^{\frac{1}{\|\mathbf{y}\|}\log \Pr_\theta(\mathbf{y}\|\mathbf{x})}$
R-DPO (Gallego, 2024)	$-\log \text{Sigmoid}\left(\beta \log \frac{\Pr_\theta(\mathbf{y}_a\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a\|\mathbf{x})} - \beta \log \frac{\Pr_\theta(\mathbf{y}_b\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b\|\mathbf{x})} + (\alpha\|\mathbf{y}_a\| - \alpha\|\mathbf{y}_b\|)\right)$
PCDPO (Zhou et al., 2024)	$-\log \text{Sigmoid}\left(\Delta^* - \Delta_{\Pr_\theta}\right) - \log \text{Sigmoid}\Delta_{\Pr_\theta}$, where $\Delta_{\Pr_\theta} = \beta \log \frac{\Pr_\theta(\mathbf{y}_a\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a\|\mathbf{x})} - \beta \log \frac{\Pr_\theta(\mathbf{y}_b\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b\|\mathbf{x})}$, $\Delta^* = \frac{\beta_1}{\text{Sim}(\mathbf{y}_a,\mathbf{y}_b\|\mathbf{x})+\beta_2} + \beta_3$
SimPO (Meng et al., 2024)	$-\log \text{Sigmoid}\left(\frac{\beta}{\|\mathbf{y}_a\|} \log \Pr_\theta(\mathbf{y}_a\|\mathbf{x}) - \frac{\beta}{\|\mathbf{y}_b\|} \log \Pr_\theta(\mathbf{y}_b\|\mathbf{x}) - \gamma\right)$
ODPO (Amini et al., 2024)	$-\log \text{Sigmoid}\left(\beta \log \frac{\Pr_\theta(\mathbf{y}_a\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a\|\mathbf{x})} - \beta \log \frac{\Pr_\theta(\mathbf{y}_b\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b\|\mathbf{x})} - \delta(\mathbf{x}, \mathbf{y}_a, \mathbf{y}_b)\right)$
Cal-DPO (Xiao et al., 2024)	$-\log \text{Sigmoid}\left(\Delta_{\Pr_\theta}\right) + \lambda \mathcal{L}_{\text{cal}}$ where $\Delta_{\Pr_\theta} = \beta \log \frac{\Pr_\theta(\mathbf{y}_a\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_a\|\mathbf{x})} - \beta \log \frac{\Pr_\theta(\mathbf{y}_b\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_b\|\mathbf{x})}$, $\mathcal{L}_{\text{cal}}$ calibrates implicit rewards.
TDPO (Zeng et al., 2024b)	$-\log \text{Sigmoid}\left(\sum_{t=1}^{T_a} \beta \log \frac{\Pr_\theta(\mathbf{y}_{a,t}\|\mathbf{x},\mathbf{y}_{a,<t})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_{a,t}\|\mathbf{x},\mathbf{y}_{a,<t})} - \sum_{t=1}^{T_b} \beta \log \frac{\Pr_\theta(\mathbf{y}_{b,t}\|\mathbf{x},\mathbf{y}_{b,<t})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_{b,t}\|\mathbf{x},\mathbf{y}_{b,<t})}\right) + \lambda \sum_t \text{KL}_t$ where KL_t denotes the token-level KL regularization term.
AOT (Melnyk et al., 2024)	$\mathcal{L}_{\text{OT}}\left(\{r_\theta(\mathbf{x}, \mathbf{y}_a)\}, \{r_\theta(\mathbf{x}, \mathbf{y}_b)\}\right)$ where $r_\theta(\mathbf{x}, \mathbf{y}) = \beta \log \frac{\Pr_\theta(\mathbf{y}\|\mathbf{x})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}\|\mathbf{x})}$, and $\mathcal{L}_{\text{OT}}$ aligns preference distributions via optimal transport.
D^2PO (Shao et al., 2025)	$-\log \text{Sigmoid}\left(\sum_{t=0}^{T} \gamma^t \beta \log \frac{\Pr_\theta(\mathbf{y}_{a,t}\|\mathbf{x},\mathbf{y}_{a,<t})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_{a,t}\|\mathbf{x},\mathbf{y}_{a,<t})} - \sum_{t=0}^{T} \gamma^t \beta \log \frac{\Pr_\theta(\mathbf{y}_{b,t}\|\mathbf{x},\mathbf{y}_{b,<t})}{\Pr_{\theta_{\text{ref}}}(\mathbf{y}_{b,t}\|\mathbf{x},\mathbf{y}_{b,<t})}\right)$

Table 3: DPO variants and their optimization objectives.

regard, RL has previously demonstrated its effectiveness in training neural networks for complex planning and reasoning within game environments, as evidenced by notable successes such as AlphaGo (Silver et al., 2016) and AlphaStar (Vinyals et al., 2019). Given these advancements and the inherently interactive nature of problem-solving, it is also natural to consider the application of RL to LLM reasoning.

In this section, we delve deeper into the application of RL to enhance the reasoning capabilities of the LLM, a topic that has recently garnered significant attention. We begin by giving a general introduction to test-time scaling that fully unleashes the reasoning potential of LLMs, including best-of-N sampling, step-by-step verification, and Monte Carlo Tree Search. We then discuss two critical issues: how to scale RL effectively, and how to iterate the RL process to enhance the reasoning capabilities of LLMs. Note that the following methods are primarily illustrated through mathematical reasoning problems, but they are applicable to a broad range of decision-making problems.

5.1 Test-time Scaling

Initially, we review the fundamental objective of RL: to maximize the rewards obtained during output generation. This goal has been extensively pursued through various training-time optimization techniques, such as policy gradient or PPO algorithms. Beyond training, recent research has illuminated the benefits of scaling up test-time computation, a process also known as test-time scaling, which can significantly enhance reward maximization, especially in reasoning tasks. In a practical implementation, one simple approach to achieve test-time scaling is the use of prompting techniques. For example, appending the phrase "Let's think step-by-step." to the input can effectively stimulate the LLM to engage in a more detailed reasoning process during generation. While this approach can improve reasoning accuracy, it often leads to suboptimal performance because the model is not inherently trained to understand the most beneficial reasoning processes. To address this issue, we can perform test-time scaling with guidance from a reward model, as discussed in the following subsections.

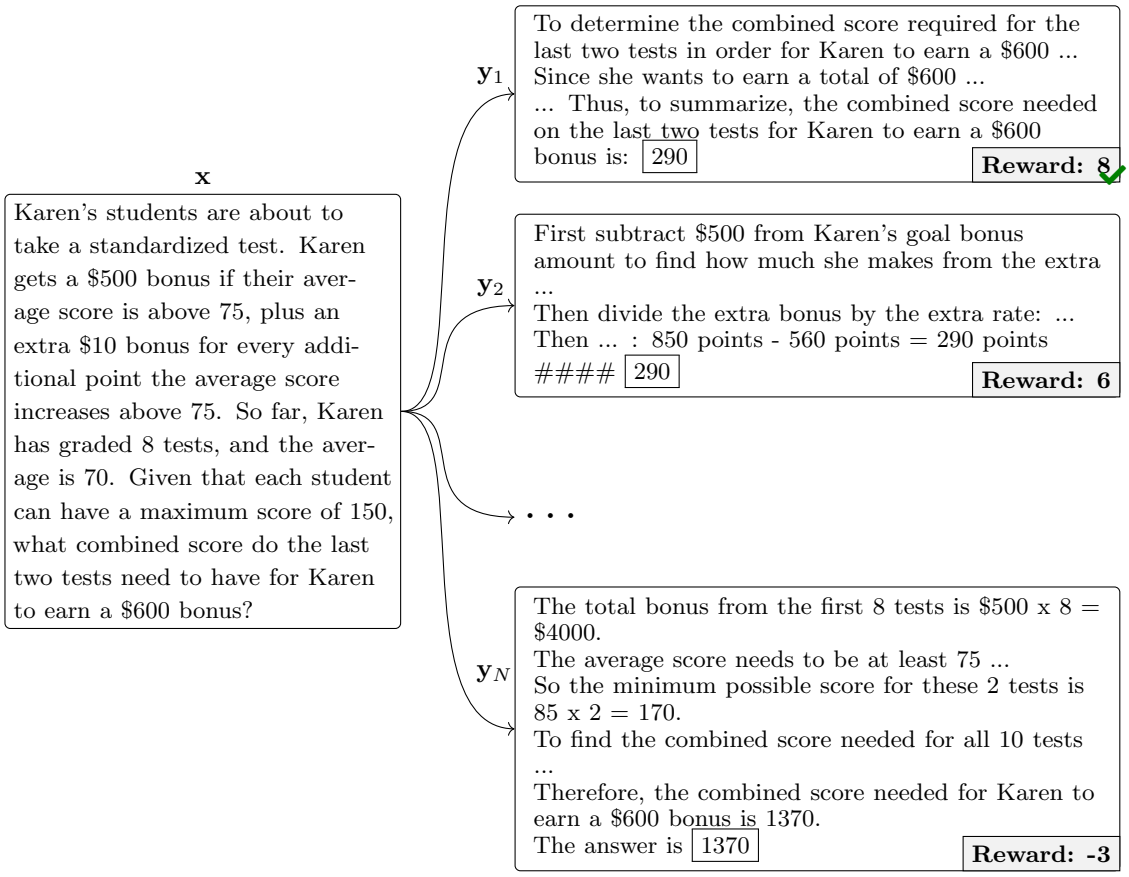


Figure 22: Best-of-n sampling. Given an input, multiple candidate outputs are sampled, and a reward model is used to select the best output. A majority vote can also determine the final output in scenarios without a reward model. For example, in this figure, most of the outputs suggest an answer of 290. Therefore, we can consider any one of the outputs resulting in 290 to be the final output.

5.1.1 Best-of-N Sampling

One approach to test-time scaling using a reward model involves sampling multiple reasoning paths given input and then using the reward model to select the best one from N alternative outputs generated by the LLM, called best-of-N sampling (BoN sampling). We can consider BoN sampling a reranking technique. In fact, reranking methods have been a prevalent technique in NLP, particularly in machine translation, where they have been employed for a long time to enhance output quality by selecting the most appropriate translation from a set of candidates. Additionally, this method often functions as a simple model ensemble approach. In such cases, different outputs generated by various models can be reranked according to a specified metric.

As illustrated in Figure 22, in the BoN sampling, we first sample N different outputs $\{\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N\}$ for the input $\mathbf{x}$:

$$\mathbf{y}_i \quad \sim \quad \Pr_\theta(\cdot|\mathbf{x}), \quad i = 1, \ldots, N \tag{62}$$

where the outputs can be sampled in various ways, depending on the decoding algorithm used by the model (e.g., nucleus sampling or beam search). Once the N output candidates are sampled, the reward model is used to evaluate and select the best one:

$$\mathbf{y}_{\text{best}} \quad = \quad \arg \max_{\mathbf{y}_i \in \{\mathbf{y}_1, \cdots, \mathbf{y}_N\}} R_\phi(\mathbf{x}, \mathbf{y}_i) \tag{63}$$

This process not only identifies the output with the highest reward but also makes it a direct evaluation of the effectiveness of the reward model. For example, the agreement between the reward model and human judgments can be assessed by judging whether $\mathbf{y}_{\text{best}}$ matches the best output selected by humans. Given its speed and cost-efficiency compared to more complex evaluation methods such as those used in PPO, this approach is widely used for evaluating the performance of reward models (Rafailov et al., 2023; Gao et al., 2023). Nevertheless, in scenarios without a reward model, we can also employ the majority vote approach to determine the final output of the reasoning task. The basic steps involve first identifying the answer that most reasoning paths converge upon and then considering any of the outputs leading to this answer as the final output.

It is worth noting that the result of BoN sampling is also influenced by the diversity of the N-best list. This is a common issue with most reranking methods. Typically, we wish the N-best output candidates to be relatively high quality but sufficiently different from each other. In many text generation systems, the N-best outputs are very similar, often differing by just one or two words. The diversity issue is even more challenging in LLMs, as the N-best outputs sampled by an LLM can differ in their wordings. Yet, their semantic meanings are often quite similar. In practice, one can adjust the model hyperparameters and/or adopt different LLMs to generate more diverse output candidates for reranking. Nevertheless, as with many practical systems, we need to make a trade-off between selecting high-quality candidates and ensuring sufficient variation in the sampled outputs.

BoN sampling can also be used to train LLMs. A closely related method is rejection sampling. In this method, we first select the "best" outputs from the N-best lists via the reward model and then take these selected outputs to fine-tune the LLM. In this way, we can introduce human preferences into the training of LLMs via a much simpler approach than PPO. Many LLMs have adopted rejection sampling for fine-tuning (Nakano et al., 2021; Touvron et al., 2023).

5.1.2 Step-by-step Verification

While BoN sampling is effective for scaling test-time computation, it is inefficient because the model must generate the entire reasoning path before evaluating its quality. Addressing this inefficiency, one approach is step-by-step verification, which evaluates the quality of each step while generating a reasoning path[9]. This allows us to identify the potential of a path at intermediate steps early and further reduce computational resources by abandoning unpromising paths. Note that in this approach, traditional reward models, which are designed to evaluate the entire output, are inadequate. Instead, we need to develop a process reward model to evaluate each step in the reasoning path.

We can collect or generate reasoning paths corresponding to problems from existing datasets to train a process reward model. Human experts then annotate each step in these paths for correctness. These annotations can be used to train LLMs or as rewards in reward modeling directly. However, in practice, richer annotations are often introduced (Lightman et al., 2024b). In addition to the *correct* and *incorrect* labels, a step can also be labeled as *neutral* to indicate that while the step may be technically correct, it might still be problematic within the overall reasoning process. Additionally, an automatic process annotation framework can be utilized, which relies on generating multiple entire reasoning paths based on the current step and using the accuracy of these paths to serve as the quality annotation of the step (Wang et al., 2024d).

Given a set of step-level annotated reasoning paths and corresponding inputs, we can train a reward model to provide a reward for each step in the reasoning process. The reward model can be treated as a classification model. Thus, its architecture can be an LLM with a Softmax layer stacked on top, akin to the architecture depicted in Figure 10. Here, consider a reasoning path including n_s steps, represented as

[9]We typically evaluate each reasoning step from multiple dimensions. A primary consideration is the presence of errors, assessing whether the intermediate step contains any inaccuracies. Furthermore, we evaluate the advantage conferred by the step, which examines the potential of the reasoning step to facilitate superior outcomes in subsequent steps (Setlur et al., 2025).

$\mathbf{y} = \{\bar{\mathbf{y}}_1, \cdots, \bar{\mathbf{y}}_{n_s}\}$. At each step k, the process reward model takes both the problem description, denoted by $\mathbf{x}$, and the reasoning steps generated so far, denoted by $\bar{\mathbf{y}}$, as inputs. It then outputs a probability distribution over the set of labels *correct*, *incorrect*, or *correct*, *incorrect*, *neutral*, to evaluate the reasoning at that point. This model can be trained with a standard classification loss, e.g., cross-entropy. Once trained, the reward model can be used to evaluate reasoning paths by assessing the correctness of each step. A simple method is to use classification log-probabilities to define the reward of each reasoning step. For example, the reward of the k-th reasoning step can be given by

$$R_\phi(\mathbf{x}, \bar{\mathbf{y}}_{\leq k}) \quad = \quad \log \Pr_\phi(\textit{correct}|\mathbf{x}, \bar{\mathbf{y}}_{\leq k}) \tag{64}$$

where $\Pr_\phi(\textit{correct}|\mathbf{x}, \bar{\mathbf{y}}_{\leq k})$ denotes the probability of the *correct* label generated by the reward model. The reward score $R_\phi(\mathbf{x}, \mathbf{y})$ can then be used to select the best step while generating a reasoning path. Additionally, as discussed in Section 4.1.4, there is the option to train a generative process reward model, also called a generative verifier in the literature, to further improve reasoning-step evaluation (Zhang et al., 2025a). Note that in practice, the process reward model serves not only to provide rewards for test-time scaling but also to train the model using RL, e.g., the rewards from this model can be employed as shaping rewards.

As illustrated in Figure 23, step-by-step verification can be conducted using a simple greedy search method. Specifically, multiple candidate reasoning steps are sampled at each step, and the process reward model is employed to select the best one. This selected step then serves as the foundation for generating the subsequent step, continuing this process until the entire reasoning path is obtained. In this way, any search method can be applied to improve test-time scaling. For example, we can expand the search space with beam search, retaining multiple promising reasoning steps at each step. Furthermore, the reasoning steps can be refined through the self-refinement technique at each step using verification feedback, such as rewards, to enhance accuracy (Yao et al., 2023a).

5.1.3 Monte Carlo Tree Search

In this subsection, we discuss the use of Monte Carlo Tree Search (MCTS), a popular search method in step-by-step verification. In practice, MCTS is not a new method but is well-established across various domains. Notably, it typically serves as an effective alternative to traditional RL methods, particularly in environments characterized by large or intricate state spaces, where conventional RL algorithms may falter. For example, within an RL framework, MCTS can aid in planning by simulating diverse actions to maximize rewards, as demonstrated by its success in complex game environments (Silver et al., 2016). In LLM reasoning, MCTS leverages randomness and structured tree search to probe potential reasoning paths, thereby expanding the search space in an efficient way. More specifically, as illustrated in Figure 24, MCTS repeatedly cycles through the following four stages to explore potential reasoning paths:

- **Selection**. This process begins at the root node (i.e., an input), where the algorithm selects promising child nodes (i.e., reasoning steps) based on specific selection strategies, such as the Upper Confidence Bound (namely UCT)[10]. More specifically, at each step k, among the sampled candidate reasoning steps, we aim to select the one that maximizes the UCT objective:

$$\bar{\mathbf{y}}_k^* \quad = \quad \arg\max_{\bar{\mathbf{y}}_k} \left(\bar{R}(\bar{\mathbf{y}}_k) + c\sqrt{\frac{\ln N_p}{N_{\bar{\mathbf{y}}_k}}} \right) \tag{65}$$

  where N_p denotes the total number of times the parent node has been visited, $N_{\bar{\mathbf{y}}_k}$ is the number of visits to the child node $\bar{\mathbf{y}}_k$, and c is a constant that balances the trade-off between exploitation of known good paths and exploration of new paths. $\bar{R}(\cdot)$ denotes the reward of a reasoning step. This reward can be static, set by a process reward model, or dynamic, continuously updated based on the delayed rewards obtained from simulations.

[10]When MCTS uses the upper confidence bound strategy for node selection, the resulting algorithm is commonly referred to as *Upper Confidence bounds applied to Trees* (UCT).

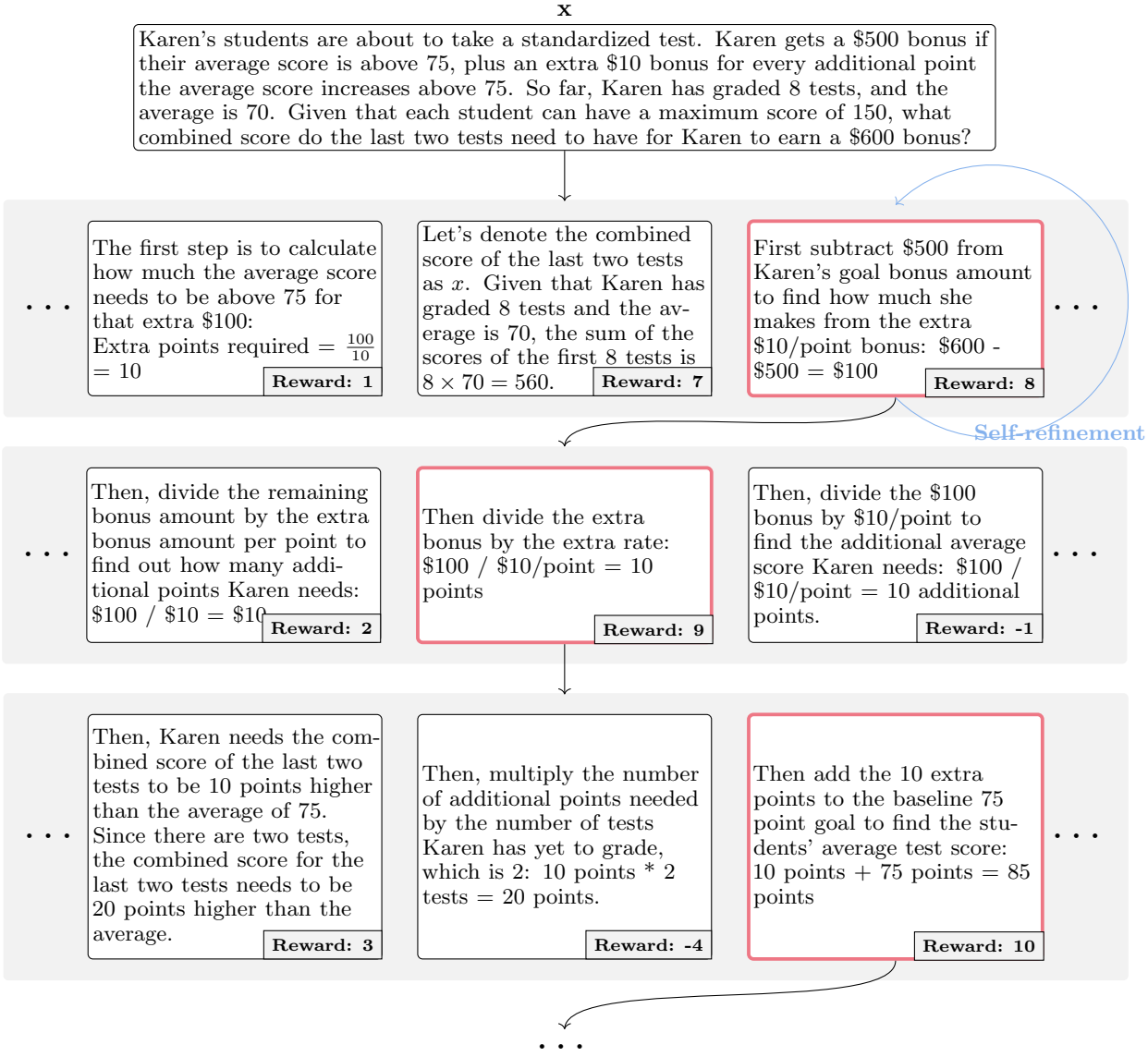


Figure 23: Step-by-step verification with greedy search. This process can be readily extended to beam search by retaining the top steps at each step from all candidate paths for further scaling up test-time computation. As the blue line indicates, the reasoning steps can be refined using verification feedback, such as rewards, to enhance accuracy.

- **Expansion**. Once a leaf node (i.e., a newly generated reasoning step based on previously selected steps) is encountered and it does not represent a terminal state of the reasoning process, the algorithm expands this node by adding it as a new child node.
- **Simulation**. The process performs simulations (or rollouts) for each new node added during the expansion stage. These simulated paths help evaluate the effectiveness of the reasoning steps initiated from the node. For example, the final answer obtained from the simulation can be compared with the correct answer to derive a delayed reward.
- **Backpropagation**. The process updates the UCT values of prior nodes using the results of these simulations. Key updates include the visitation counts, N_p and $N_{\bar{\mathbf{y}}_k}$ in Eq. (65), reflecting how

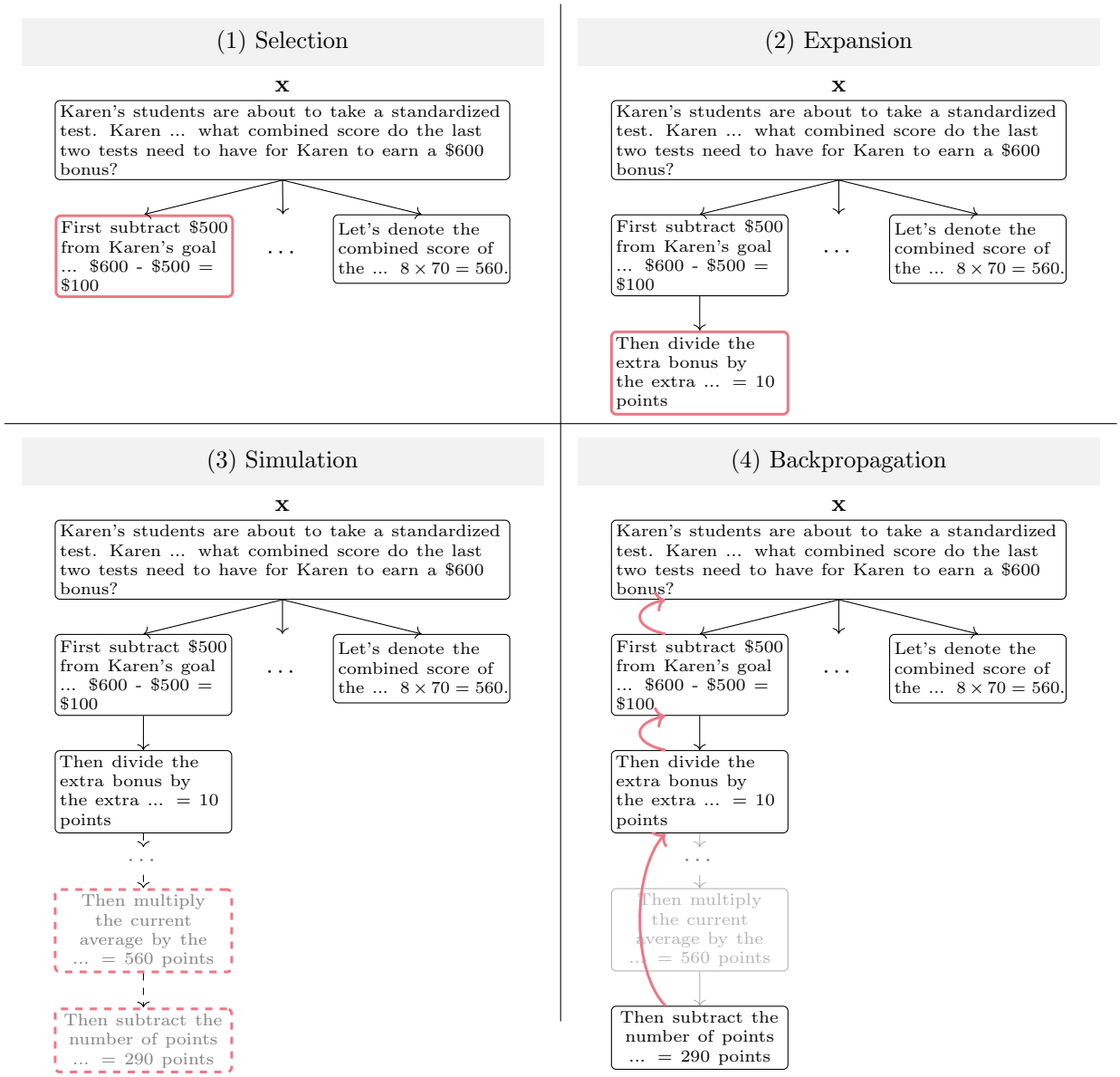


Figure 24: Illustration of step-by-step verification with MCTS. The process consists of four phases: (1) Selection, where the algorithm navigates the decision tree to select the most promising node based on strategies like UCT; (2) Expansion, where new potential nodes are added to the tree for further exploration; (3) Simulation, where each new node is evaluated by simulating possible outcomes (i.e., reasoning paths); and (4) Backpropagation, where the outcomes from the simulations are used to refine node values, such as UCT values, thus enhancing the decision-making for subsequent explorations.

frequently each node has been explored. Additionally, this stage allows for the incorporation of delayed rewards to adjust the $\bar{R}(\bar{\mathbf{y}}_k)$ values.

5.2 Iterative RL

Training LLMs with RL usually follows a two-phase approach: training a pre-trained LLM with SFT and further training with an RL algorithm applied to the SFT LLM. However, using large-scale RL in such a

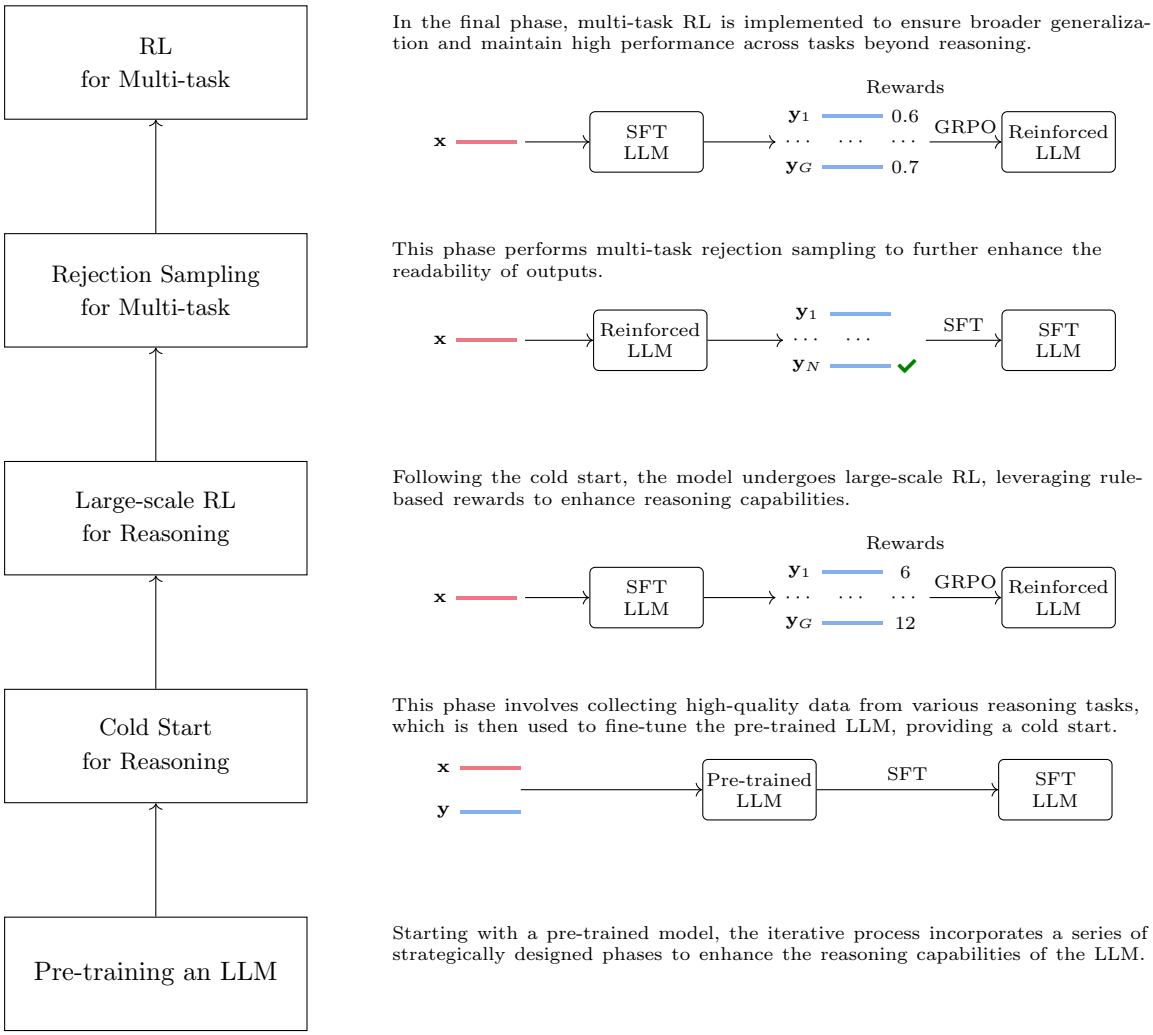


Figure 25: Illustration of iterative RL in DeepSeek-R1 (Guo et al., 2025). This method maintains multi-phase RL to develop a robust reasoning LLM with a GRPO algorithm. Initially, a pre-trained LLM is fine-tuned using a small set of high-quality labeled reasoning data with SFT as a cold start. Then, large-scale RL is applied specifically to enhance reasoning capabilities. After that, the model undergoes rejection sampling across multiple tasks to refine output quality. In the final phase, the model is further trained using RL to enhance generalization across various tasks.

two-phase approach to train LLMs in reasoning may lead to significant knowledge forgetting as the model continuously adjusts to fit the reward model. For example, as mentioned in DeepSeek-R1 (Guo et al., 2025), directly applying large-scale RL can achieve the desired reasoning outcomes, but often at the cost of reduced readability in the reasoning process. To address these challenges, we can utilize an iterative RL approach to continuously enhance the various capabilities of the LLM. Here we consider DeepSeek-R1 as an example to illustrate how to perform an iterative RL. The idea is to split the RL process into multiple phases, each designed to enhance different capabilities using varied rewards, to develop a robust reasoning model that is able to generate clearer and more comprehensible reasoning paths. Figure 25 provides a schematic illustration of the iterative RL process. Here we give a brief outline of each phase involved.

- The iterative RL aims to train an LLM that generates a human-like reasoning process. Initially, it involves collecting high-quality data from various reasoning tasks, such as mathematical problem-

solving and code generation. This data is then used to fine-tune a pre-trained LLM as a cold start. This phase is designed to equip the LLM with basic reasoning capabilities, preventing linguistically confused or nonsensical reasoning paths from being sampled during the following RL phase.

- After the cold start phase, large-scale RL is employed on the reasoning task. During this phase, rule-based rewards (i.e., format checking and answer verification) are utilized to optimize the reasoning process continuously. By doing so, the model learns longer and more reasonable reasoning paths, progressively enhancing its capacity to tackle complex reasoning tasks.
- While the model has learned how to generate reasoning paths through large-scale RL, it often produces outputs with poor readability. This can mainly be attributed to the large-scale RL phase, which does not focus on optimizing the readability of the reasoning paths but instead solely on format correctness and answer accuracy. To address this issue, the third phase involves using rejection sampling to further enhance the readability of outputs. More specifically, multiple outputs are sampled from the model trained in the previous phase across various tasks such as mathematical reasoning, code generation, and question answering. A reward model then selects the best outputs, as discussed in Section 5.1.1. These selected outputs are used to train the model using SFT, which aims to refine its performance by aligning it more closely with human-like reasoning.
- In the final phase, multi-task RL is implemented to ensure broader generalization and maintain high performance across tasks beyond reasoning. The process involves training the model using RL on various tasks against a general reward model.

An interesting issue arises with this design of iterative RL: why is RL aimed at enhancing reasoning capabilities placed before the later rejection-sampling and multi-task RL phases rather than postponed to the end? The key reason is that the later phases depend on a model that can already explore meaningful reasoning trajectories. If the model has not first acquired this ability, rejection sampling is likely to draw from low-quality reasoning paths, and multi-task RL has to optimize general alignment and task performance while also trying to repair basic reasoning behavior. Placing reasoning-oriented RL early therefore expands the pool of useful trajectories for subsequent data construction and makes later optimization stages more effective. This ordering is also practical because reasoning rewards, which are inherently more stable as described in Section 4.3.2, are well-suited for large-scale RL. Furthermore, reasoning capabilities can be applicable across many tasks, allowing the early RL phase to serve as a broad capability-building stage before later task-specific refinement. In this case, large-scale RL could be viewed as an approach to continued pre-training.

Since the optimization objective of each phase is different, the design of iterative RL can also be analyzed and understood from the perspective of multi-objective optimization (Wang et al., 2024a). In the context of multi-objective optimization, iterative RL for LLMs can be likened to interactive methods where the solution process is iterative, and preferences are actively defined and refined by the decision-maker during the search for the most preferred solutions (Miettinen et al., 2008; Deb et al., 2016). More specifically, in this scenario, RL can be seen as a decision-maker, iteratively refining and enhancing the different capabilities of the LLM across different phases.

5.3 Large-scale RL

While test-time scaling is instrumental in exploring more effective reasoning paths, its potential is restricted if the model has not learned to generate high-quality reasoning paths. In other words, in practice, test-time scaling struggles to achieve desired outcomes without foundational reasoning ability, no matter how extensive it is (Yuan et al., 2023; Zeng et al., 2025). Consequently, there has been a growing research interest in training LLMs specifically to develop reasoning abilities that mimic human-like processes. A straightforward approach is to use annotated reasoning data to train the LLM through SFT. However, a significant challenge in this approach is the high cost of annotations, especially since annotating detailed step-by-step reasoning paths is significantly more cost-intensive than annotating SFT data in other tasks like machine translation and summarization.

Another more advanced approach is to utilize large-scale RL, transitioning from human-annotated to self-reinforced learning processes. Unlike the RL used as a fine-tuning method discussed in Section 3, which typically involves training for a few dozen or hundreds of steps at a small scale, this approach applies RL at a larger scale with rewards to break free from the limitation of supervised reasoning data. This approach has enabled the development of robust reasoning models, such as OpenAI-o1 (OpenAI, 2024), DeepSeek-R1 (Guo et al., 2025), and Kimi-1.5 (Team et al., 2025b). Notably, DeepSeek-R1-Zero was able to develop a robust reasoning model by applying large-scale RL to a pre-trained LLM without the need for any annotated reasoning data. However, despite its successes, large-scale RL is not easy and introduces unique challenges that are not present at smaller scales, as follows.

One is that large-scale RL requires a highly generalizable reward model. As the scale of training increases, the model is trained on broader data, necessitating a reward model capable of effectively generalizing across this varied data. On the other hand, the extensive scope of training introduces significant variability in the model, leading to considerable changes in sampling behaviors. Consequently, it is crucial to ensure that the reward model possesses robust generalization capabilities to prevent overfitting and maintain the effectiveness of the learning process. There are several methods to achieve this. For example, Guo et al. (2025) incorporated rule-based rewards, as discussed in Section 4.3.2, such as format checking and answer verification in reasoning scenarios, which can provide a stable reward throughout the learning process. This suggests that in certain RL scenarios, prioritizing rule-based rewards may be beneficial if they can effectively describe human preferences. Yuan et al. (2024) introduced a self-rewarding framework that dynamically updates the reward model based on the currently optimized policy model so that this reward model can effectively evaluate the behaviors from the current policy model.

Another is that large-scale RL might be constrained by the reference model. To prevent the policy model from deviating too far from its initial state, an SFT LLM is typically employed as the reference model, adding a penalty that restricts updates to ensure the policy model operates within a desirable behavior region. However, reliance on a static reference model can limit the adaptability and innovation potential of the policy model in large-scale RL. A common strategy to mitigate this issue is to periodically update the reference model to better align with the improving capabilities and knowledge of the policy model, thus striking a balance between stability and adaptiveness during the training process (Gorbatovski et al., 2025).

5.4 On-Policy Distillation

While large-scale RL has shown strong potential for improving model capabilities, it often relies on large amounts of verifiable training data. Several approaches have been explored to relax this requirement. A straightforward solution is to construct proxy rewards from the model's own outputs. For example, we can use majority voting to derive pseudo-gold answers and compute accuracy-based rewards. A more promising direction is *on-policy distillation* (OPD), which introduces supervision from a stronger teacher model while preserving on-policy exploration of the student model[11]. The key idea is to let the student model generate reasoning trajectories on-policy and then use the teacher model to provide token-level distributional supervision at the states actually visited by the student.

Compared with conventional offline distillation, OPD adopts a simple but important design: the training samples are generated online by the current student policy rather than collected offline from the teacher model. This design mainly addresses the distribution mismatch in offline distillation, where the student is trained on teacher-generated trajectories that may differ from the states it encounters during its own inference. In contrast, OPD provides teacher supervision directly on the states visited by the student, allowing the teacher to correct the student's actual behaviors. For example, consider a mathematical reasoning problem where the student has already generated the partial trajectory "3x+7=22 $\rightarrow$ 3x=15". At this student-generated state, the teacher model can directly provide token-level supervision that assigns a higher probability to generating "x=5" next, rather than an incorrect continuation such as "x=4". In this

[11] In standard knowledge distillation settings, we typically refer to the "smaller" model as the *student model* and the "larger" model as the *teacher model*.

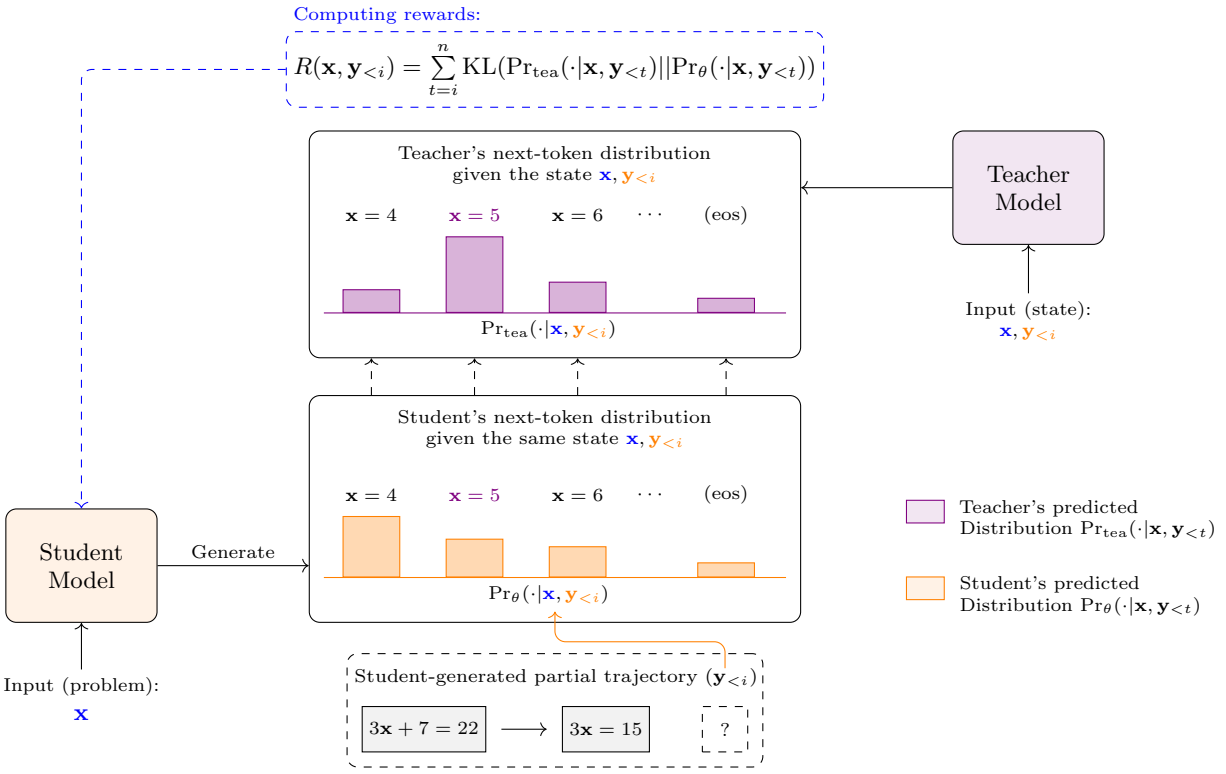


Figure 26: Illustration of OPD (Agarwal et al., 2024). The student model generates trajectories from its current policy, while the teacher model provides next-token distributional supervision at the states visited by the student. The divergence between the teacher and student distributions is used to construct dense token-level rewards for optimizing the student model.

way, OPD teaches the student how to continue correctly from the states it actually visits, rather than only imitating complete trajectories generated by the teacher model.

This implementation of OPD is relatively simple, as illustrated in Figure 26. Given an input $\mathbf{x}$, we first sample an output $\mathbf{y} = \{y_1, \ldots, y_n\}$ from the current student model $\mathrm{Pr}_{\theta}(\cdot|\mathbf{x})$. For each token position i, the prefix $(\mathbf{x}, \mathbf{y}_{<i})$ denotes a state actually visited by the student model. We then feed the same state into the teacher model $\mathrm{Pr}_{\mathrm{tea}}$ and obtain its probability distribution over the next token. The student model is optimized to match this teacher distribution. This OPD objective can be defined as:

$$\mathcal{L}_{\mathrm{opd}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{S}_x,\, \mathbf{y}\sim\mathrm{Pr}_{\theta}(\cdot|\mathbf{x})} \left[\frac{1}{n} \sum_{i=1}^{n} \mathrm{KL}\left(\mathrm{Pr}_{\mathrm{tea}}(\cdot|\mathbf{x}, \mathbf{y}_{<i}) \| \mathrm{Pr}_{\theta}(\cdot|\mathbf{x}, \mathbf{y}_{<i})\right) \right] \tag{66}$$

Here, the KL divergence measures the discrepancy between the teacher and student distributions at each student-visited state. For optimization, we can treat the negative KL divergence at each token position as a token-level reward, such that a smaller discrepancy between the student and teacher distributions yields a higher reward. Based on these dense token-level rewards, standard RL algorithms can then be directly applied to optimize the student model. From the perspective of reward construction, OPD provides more fine-grained supervision than conventional outcome-based approaches, which assign rewards only after the complete response is generated. In contrast, OPD provides token-level feedback throughout generation, making the supervision denser. However, this benefit comes with additional computational cost. The teacher model must evaluate student-generated states and produce next-token distributions during training, which requires extra forward passes and increases both memory and computation overhead.

6 Agentic RL

Using RL to train agents, often referred to as *agentic RL*, is not a new concept. In classical machine learning, agentic RL typically involves training a domain-specific decision model from scratch. For example, in tasks such as playing complex games like Atari and Go (Mnih et al., 2013; Silver et al., 2017) or controlling robotic locomotion (Lee et al., 2024), the goal is to explore a structured environment and learn an optimal policy starting from random initialization.

With the emergence of LLMs as core components of autonomous systems, the role of RL has shifted from learning policies from scratch to adapting pretrained models for sequential decision-making in dynamic and open-ended environments. In this context, agentic RL aims to address a key limitation of LLMs: although they contain extensive world knowledge, their outputs are not inherently optimized for long-horizon interactions. For example, an SFT-trained LLM may successfully generate code for calling an external API, but it may fail to recover from unexpected execution errors during interaction.

In this section, we focus on the emerging paradigm of agentic RL for LLM-based agents. We first discuss how RL enables agents to acquire fundamental capabilities, including long-horizon planning and tool-integrated reasoning. We then introduce the role of environment design and scaling, which provide the essential foundation for training and evaluating agentic RL systems. We also discuss credit assignment, which provides more fine-grained feedback for agent learning. Finally, we explore how agents can continuously improve from their interaction experiences through mechanisms, including memory management, skill optimization, and trajectory refinement.

6.1 Building Agent Capabilities

LLM-based agents extend LLMs from passive response generators to interactive decision-making systems. Instead of producing a single response, agents continuously observe their environments, select actions, and update their behaviors through interaction. This interaction loop enables agents to solve complex tasks that require long-horizon execution. Although LLMs already possess strong capabilities, such as instruction understanding and reasoning, these capabilities are mainly developed for static text generation and may not be sufficient for dynamic interaction scenarios. When deployed as agents in real-world environments, LLMs need to further adapt their reasoning abilities to sequential decision-making, where they must decompose high-level goals into executable steps, select appropriate actions, and recover from unexpected failures during execution.

Such agentic capabilities are difficult to acquire from static input-output demonstrations alone. The effectiveness of an individual decision often depends on its long-term consequences and the final task outcome. A locally reasonable action may eventually lead to failure after multiple interactions, while an unsuccessful attempt may provide valuable experience for future decisions. To address this challenge, RL has become a standard approach for training agentic systems, enabling agents to explore different behaviors and improve their abilities through environmental feedback (Singh et al., 2025c; Feng et al., 2025).

In this subsection, we focus on two key capabilities in agentic RL: *planning* and *tool use*. Planning enables agents to decompose complex goals into executable steps and make effective decisions over long horizons. Tool use allows agents to interact with external resources, such as knowledge sources and computational tools, thereby extending their capabilities beyond the information encoded in model parameters.

6.1.1 Planning

An agent's autonomy refers to its ability to independently determine the steps required to complete a given task, even when these steps are not explicitly specified in the input. This capability requires the agent to infer the necessary procedure from the task description, decompose the goal into executable subgoals, and decide when and how to invoke external tools. Such autonomy is often reflected in the generation of a

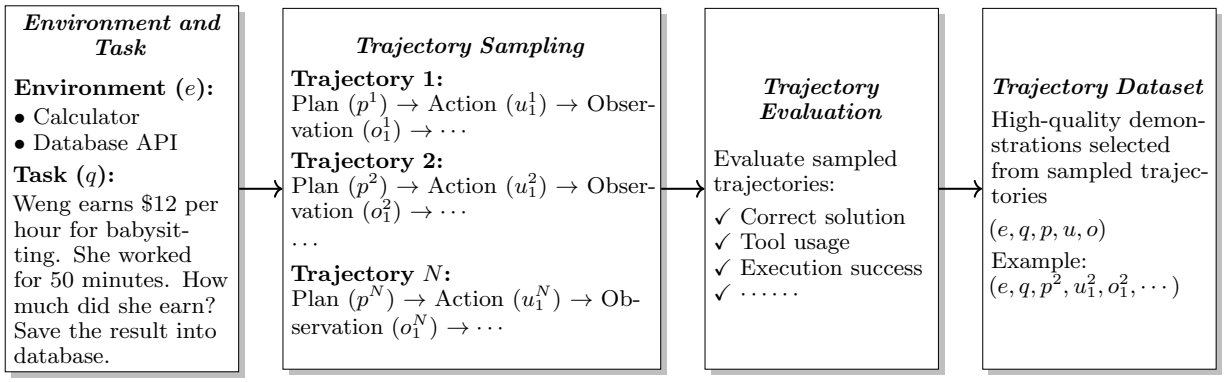


Figure 27: Overview of SFT data construction for improving agent planning.

structured plan that guides the execution trajectory. For example, given a specific task description, the agent may produce an actionable plan with tool-use decisions as follows:

Input *Environment*: Calculator, Database API
Task: Weng earns $12 per hour for babysitting. Yesterday, she babysat for 50 minutes. How much did she earn? Please save the result into the database.

Output 1. Extract the relevant numerical values and identify the required computation.
2. Call the `calculator` tool to evaluate the expression $12 \times (50/60)$.
3. Format the computed result into a valid JSON schema.
4. Invoke the `update_record` API to save the final earnings into the database.
Next, I will execute the plan step by step.
.....

Unlike single-turn response generation, agent planning requires a sequence of interdependent decisions, where early choices can affect subsequent tool use, environmental feedback, and final task success. Improving planning capability therefore requires not only teaching the model to generate reasonable intermediate steps, but also enabling it to adjust its behavior based on execution outcomes. To this end, researchers have explored various approaches to achieving this goal. Among them, one promising approach is RL, as it naturally formulates agent planning as sequential decision-making driven by environmental feedback and trajectory-level rewards. In practice, SFT provides a cold start by allowing the model to imitate high-quality planning trajectories, while RL further refines the agent through interaction-based feedback and trajectory-level optimization.

6.1.1.1 Supervised Planning Data

Before applying RL, SFT is commonly adopted to provide a cold start for LLM-based agents by teaching them basic planning and interaction behaviors (Chen et al., 2023a; Zhang et al., 2024b; Zeng et al., 2024a). As illustrated in Figure 27, constructing SFT data for agent planning typically involves three stages: (1) preparing diverse environments and planning tasks; (2) synthesizing expert-level trajectories, consisting of action-observation sequences, through agent-environment interactions; and (3) selecting high-quality trajectories based on predefined evaluation criteria. Specifically, expert trajectories can be generated by leveraging state-of-the-art LLMs as expert agents and retaining reliable demonstrations according to reward signals or task success criteria. Through this process, LLMs can learn planning behaviors from

demonstrations and improve their ability to generate structured execution plans.

$$\mathbf{x} = [e, q] \tag{67}$$

$$\mathbf{y} = [p, u_1, o_1, \cdots, u_T, o_T] \tag{68}$$

where e denotes the environment information, including available tools and external resources, and q represents the task description provided by users. p denotes the generated plan that decomposes the task into intermediate steps. u_t and o_t represent the agent's behavior and the corresponding observation from the environment at the t-th interaction step, respectively. Specifically, u_t denotes the executable behavior performed by the agent, such as tool invocation, while o_t denotes the feedback returned by the environment after executing u_t. Therefore, $\{u_1, o_1, \cdots, u_T, o_T\}$ forms a sequential interaction trajectory between the agent and the environment. It is worth noting that although u_t and a_t in Section 2 both represent "action", they are defined at different levels of abstraction. Specifically, u_t represents a high-level interaction step in agent planning, such as calling a specific tool, whereas a_t represents the low-level token-level action in the standard LLM-based RL formulation, corresponding to the generation of an individual token by the language model.

Similar to LLM instruction tuning (Longpre et al., 2023; Zhou et al., 2023), the scale and quality of trajectory data significantly influence the effectiveness of agent training. High-quality trajectories provide explicit supervision signals for agents to learn how to decompose complex tasks, select appropriate tools, and interact with dynamic environments. Therefore, recent studies have explored constructing trajectory-based instruction tuning datasets to enhance the planning capabilities of LLM-based agents.

A straightforward approach is to utilize generated trajectories for instruction tuning. However, trajectories collected from LLM-based agents may contain unsuccessful plans, ineffective tool usage, or execution failures, which can introduce noisy supervision. Therefore, existing methods typically employ filtering strategies based on task success criteria or heuristic rules to select high-quality demonstrations. For example, AgentTuning constructs AgentInstruct by collecting interaction trajectories and filtering unsuccessful samples, while FireAct investigates the impact of diverse trajectory data from multiple tasks and reasoning paradigms for improving agent fine-tuning (Zeng et al., 2024a; Chen et al., 2023a).

Beyond selecting high-quality trajectories, another challenge lies in effectively representing the complex interaction processes within trajectories. Unlike conventional instruction tuning, agent trajectories contain not only final responses but also intermediate planning processes and multi-step interactions with environments. Therefore, how to organize and annotate different components of trajectories becomes crucial for enabling agents to acquire planning and execution capabilities. To address this challenge, AgentLUMOS introduces a modular training framework that explicitly separates high-level planning and low-level execution (Yin et al., 2024). Specifically, its planning module learns to generate high-level subgoals, while its grounding module learns to translate these subgoals into executable actions through external tools. Such a modular design enables agents to acquire different capabilities from structured trajectory annotations.

Furthermore, even with effective trajectory selection and representation, scaling the generation of diverse and high-quality trajectory data remains challenging. Existing approaches often rely on manually designed environments and planning tasks, which limits the diversity and coverage of collected trajectories. Designing new environments requires substantial human expertise, while constructing tasks with appropriate difficulty levels remains difficult. To address this limitation, AGENTGEN explores automatically generating diverse environments and planning tasks with LLMs, enabling large-scale synthesis of trajectory data with varying task complexity for agent training (Hu et al., 2025).

6.1.1.2 Learning to Plan with RL

Although SFT can provide a useful cold start for agent planning, it mainly teaches the model to imitate offline demonstrations. This limits its ability to improve beyond the quality and coverage of the collected trajectories. In contrast, RL further optimizes planning through direct interaction with the environment, where the agent receives feedback based on the outcome of its generated plans and executed behaviors.

Under the agent planning formulation, the LLM-based agent can be viewed as a high-level policy that generates a plan and then produces a sequence of executable behaviors (Li et al., 2025b). Given the environment information e and task description q, the agent first generates a plan p and then interacts with the environment through a sequence of behaviors and observations:

$$\tau = [p, u_1, o_1, \cdots, u_T, o_T] \tag{69}$$

The goal of RL is to optimize the agent policy so that the generated trajectory receives higher feedback from the environment. This objective can be written as

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{\tau \sim \pi_\theta(\cdot|e,q)} \left[R(\tau) \right] \tag{70}$$

where π_θ denotes the agent policy parameterized by the LLM, and $R(\tau)$ denotes the trajectory-level reward that evaluates the overall quality of the planning process. In practice, this reward can be defined according to task success. The objective can then be optimized using standard RL algorithms introduced in Section 3, such as PPO or GRPO.

Compared with SFT, RL provides two important advantages for agent planning. First, it enables the agent to learn from execution outcomes rather than only from static demonstrations. If a plan leads to failed tool execution or poor environmental feedback, the agent can be penalized and encouraged to explore alternative behaviors. Second, RL supports trajectory-level credit assignment, allowing the model to adjust earlier planning decisions according to later outcomes. This is crucial for agent planning, where early subgoal decomposition or tool selection can substantially affect the final task success.

6.1.2 Tool Use

Tool use is another fundamental capability of LLM-based agents. Tools extend the agent beyond its internal parameters. They allow the agent to access external knowledge and execute actions in external systems. Early studies show that LLMs can use tools through prompt-based reasoning-action patterns (Chang et al., 2024; Yao et al., 2023b). ReAct is a representative example, where the model alternates between reasoning steps and tool-use actions (Yao et al., 2023b). However, prompt-based tool use is often unstable. It depends heavily on the capability of the model. Thus, SFT methods train LLMs on tool-use trajectories, so that the model can learn when and how to invoke tools from demonstrations (Komeili et al., 2022; Schick et al., 2023). Still, SFT mainly teaches imitation. It cannot directly optimize whether the agent should use a tool, which tool it should select, or how to balance tool benefit with tool cost. RL provides a natural framework for efficient tool use. We can define rewards based on task success, tool-use correctness, or tool-use completeness (Qian et al., 2025; Singh et al., 2025b).

6.1.2.1 Prompt-based Tool Use

Prompt-based tool use is a simple approach to enable LLM-based agents to interact with external tools without updating model parameters. However, the model may not naturally know when to invoke a tool or how to organize tool-use behaviors. Thus, we usually need to provide tool descriptions and a few demonstrations in the prompt. These demonstrations show the model how to alternate between reasoning, tool invocation, and observation. At each step, the model first generates a reasoning trace, then produces a tool-use action. After the tool returns an observation, the model uses this feedback to update its next decision. For example,

Tools `calculator(expression)`: return the numerical result of an expression.
`update_record(json)`: save a JSON record into the database.

Demonstration Task: Tom earns $15 per hour. He worked for 2 hours. How much did he earn? Please save the result into the database.
Thought: I need to compute the total payment.
Action: `calculator`(15×2)
Observation: 30
Thought: The payment is 30 dollars. I need to save it into the database.
Action: `update_record`({"earning": 30})
Observation: success
Answer: Tom earned $30, and the result has been saved into the database.

Current Task Weng earns $12 per hour for babysitting. Yesterday, she babysat for 50 minutes. How much did she earn? Please save the result into the database.

Output **Thought**: I need to compute the payment based on the hourly wage and working time.
Action: `calculator`$(12 \times 50/60)$
Observation: 10
Thought: The payment is 10 dollars. I need to save it into the database.
Action: `update_record`({"earning": 10})
Observation: success
Answer: Weng earned $10, and the result has been saved into the database.

This demonstration teaches the model a tool-use pattern. The model learns to reason about the task, invoke an external tool when necessary, read the returned observation[12], and continue the execution process.

6.1.2.2 Supervised Tool-use Data

To make tool use more reliable, we can further train LLMs on tool-use trajectories. In this setting, each training example contains a user task, available tool descriptions, and a demonstrated tool-use process. The model learns when to invoke a tool, which tool to select, how to construct valid arguments, and how to use the returned observation. For example, we can construct a training example as follows:

$$\mathbf{x} = \text{Task: Weng earns \$12 ... Tools: calculator(expression), ... , update\_record(json)}$$
$$\mathbf{y} = \text{Thought: compute the payment; Action: calculator}(12 \times 50/60)\text{; ...; Answer: Weng earned \$10.}$$

With such data, SFT teaches the model to imitate the demonstrated tool-use pattern. The model can learn the format of tool invocation and the role of observations in later generation.

A common way to construct such data is to manually annotate tool-use trajectories. However, this requires annotators to know which tool should be used, how to write valid arguments, and how to judge whether the returned observation is useful. This process is costly and difficult to scale to many tools and tasks.

A representative approach to addressing this data construction problem is Toolformer, which uses a self-supervised method to generate tool-use data (Schick et al., 2023). Instead of relying on large-scale human annotations, Toolformer starts from only a few demonstrations for each API. It then lets the language model annotate a large text corpus with possible API calls. After executing these calls, Toolformer keeps only the API calls that help the model better predict future tokens. The filtered data is then used to

[12]We refer to the response returned by an external tool as an "observation".

fine-tune the model. Through this process, the model learns when to call a tool, what arguments to pass, and how to incorporate the tool result into later generation.

Beyond learning to use a small set of tools, later studies further scale tool-use training to larger and more realistic API spaces. API-Bank builds a benchmark and training set for tool-augmented LLMs, where models need to plan, retrieve, and call APIs in executable environments (Li et al., 2023). ToolLLM constructs ToolBench with more than 16,000 real-world APIs and uses automatically generated instruction-solution trajectories to train ToolLLaMA (Qin et al., 2024). Gorilla focuses on API calling at scale and fine-tunes LLaMA-based models to generate accurate API calls. It also uses a document retriever to reduce API hallucination and adapt to changing API documents (Patil et al., 2024).

6.1.2.3 Learning to Use Tools with RL

Although SFT can teach LLMs to imitate tool-use demonstrations, it is still limited by the coverage and quality of offline trajectories. The model mainly learns the tool-use patterns that appear in the supervised data. As a result, it may fail to generalize to unfamiliar tools, complex tool combinations, or new interaction patterns. More importantly, SFT does not directly optimize whether a tool call is necessary, whether the selected tool is appropriate, or whether the returned observation improves the final answer. This limitation becomes more serious in multi-step tool-use scenarios, *where the model needs to decide when to call a tool, how to formulate the tool input, how to interpret the tool output, and when to stop using tools.*

To address this limitation, researchers have explored RL for enhancing tool use. Instead of only imitating fixed demonstrations, the model can interact with tools during rollout and receive feedback from task outcomes or tool execution results. Given a user task q and a set of available tools $\mathcal{T}$, the model generates a tool-use trajectory:

$$\tau = [h_1, u_1, o_1, \cdots, h_T, u_T, o_T, y] \tag{71}$$

where h_t denotes the reasoning state at the t-th step, u_t denotes the high-level tool-use action, o_t denotes the observation returned by the tool or environment, and y denotes the final answer. Here, h_t can include natural language reasoning, while u_t represents an executable behavior such as selecting a tool and providing arguments. The goal of RL is to optimize the policy so that the generated trajectory receives a higher reward:

$$\max_{\theta} \ \mathbb{E}_{\tau \sim \pi_\theta(\cdot|q,\mathcal{T})} \left[R(\tau) \right] \tag{72}$$

In practice, this objective can be optimized with standard RL algorithms such as PPO or GRPO. Recent studies show that such outcome-driven optimization can help LLMs learn more adaptive tool-use behaviors, such as deciding when to search, when to execute code, and how to revise a tool call after receiving an error message (Jin et al., 2025; Chen et al., 2025; Feng et al., 2025; Singh et al., 2025a).

However, RL-based tool use also introduces several unique challenges. First, the action space is more complex than ordinary text generation. A tool-use action may involve tool selection, argument generation, execution timing, and response integration. A small error in any part can lead to tool failure. Second, the reward signal is often sparse and delayed. The environment may only provide a final success signal after the whole trajectory is completed, making it difficult to identify which tool call causes success or failure. Third, tool use may introduce additional costs. Unnecessary tool calls can increase latency and computation cost, while insufficient tool use may lead to incorrect answers. Therefore, the model needs to learn not only how to use tools, but also how to use them efficiently.

A key direction is to design more informative rewards for tool use. Instead of using only a final-answer reward, we can decompose the reward into multiple components:

$$R(\tau) = R_{\text{ans}}(\tau) + \alpha R_{\text{fmt}}(\tau) + \beta R_{\text{exec}}(\tau) + \gamma R_{\text{tool}}(\tau) - \lambda C_{\text{tool}}(\tau) \tag{73}$$

where R_{ans} measures final answer correctness, R_{fmt} measures whether the model follows the required tool-use format, R_{exec} measures whether tool calls can be successfully executed, and R_{tool} measures whether the

model selects appropriate tools and passes valid arguments. C_{tool} denotes the cost of tool use, such as the number of tool calls, latency, or computational expense. The coefficients α, β, γ, and λ control the relative importance of different reward terms.

In application, this reward decomposition provides denser feedback than final-answer rewards. For example, if the model selects the correct tool but passes an invalid argument, the execution reward can penalize the invalid call while preserving credit for the correct tool selection. If the model reaches the correct answer but uses many unnecessary tool calls, the cost term can discourage inefficient behavior. ToolRL systematically studies this reward design problem and shows that fine-grained reward decomposition can make RL training more stable and effective for tool selection and tool application (Qian et al., 2025).

Another direction is to use RL to improve strategic tool use. For example, ReTool first builds a cold-start model with code-augmented reasoning traces and then applies RL with real-time code execution. During rollout, the model interleaves natural language reasoning with code execution, observes execution results, and revises its subsequent reasoning. This allows the model to discover when and how to invoke a code interpreter based on outcome feedback rather than human-written rules (Feng et al., 2025). Search-R1 and ReSearch follow a similar idea in retrieval-augmented reasoning. They train models to generate search queries during reasoning and use retrieved information to update later steps, rather than relying on fixed retrieval pipelines or hand-crafted search prompts (Jin et al., 2025; Chen et al., 2025).

So far, we have introduced how RL can enhance the core capabilities of LLM-based agents. Since LLM-based agents are built upon underlying LLMs, their RL training can naturally leverage the general LLM-based RL algorithms introduced in Section 3. However, agentic RL introduces additional challenges beyond standard response optimization. A key challenge is *how to obtain high-quality feedback from environments* and *how to effectively use these signals to improve agent behaviors*. As a result, recent work on agentic RL requires not only appropriate RL algorithms, but also specialized approaches for addressing the unique challenges introduced by agentic settings (Huo et al., 2026; Liu et al., 2026b).

6.2 Environment Design and Scaling

The previous subsections focus on optimizing the agent policy for planning and tool use. In this subsection, we turn to another important component of agentic RL: the environment that produces interaction experience. An environment is more than a testbed. It determines which actions the agent can perform, what observations it receives, how the underlying state changes after each action, and how task success is evaluated. More formally, we can represent an environment as

$$e = (\mathcal{S}, \mathcal{U}, \mathcal{O}, P_e, V_e) \tag{74}$$

where $\mathcal{S}$ is the state space, $\mathcal{U}$ is the set of available actions or tool invocations, and $\mathcal{O}$ is the observation space. The transition function P_e determines how an action changes the environment state, while the verifier V_e determines whether the resulting state satisfies the task objective. This formulation describes how the environment produces observations and feedback, rather than how the agent generates its actions.

The quality of an environment directly affects what an agent can learn. For example, an agent trained in a narrow environment may simply memorize specific APIs, task templates, or interaction patterns. An unreliable environment may return inconsistent observations or incorrect rewards, introducing noise into the learning process. Ideally, an effective environment should provide diverse tasks, executable interactions, reliable state transitions, and verifiable outcomes.

To encourage generalization, we can train the agent over a distribution of environments rather than within a single fixed environment:

$$\theta^* = \arg\max_{\theta} \mathbb{E}_{e\sim p(\mathcal{E}), q\sim p_e(\mathcal{Q})} \mathbb{E}_{\tau\sim\pi_\theta(\cdot|q,e)} \left[R_e(\tau)\right] \tag{75}$$

where $p(\mathcal{E})$ denotes the environment distribution, and $p_e(\mathcal{Q})$ denotes the task distribution within environment e. The interaction trajectory τ follows the definition introduced in the previous subsections. This objective

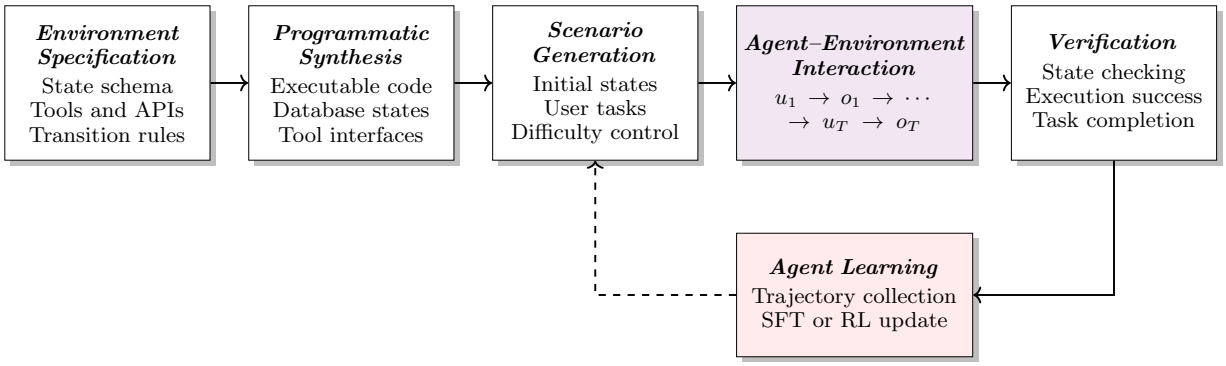


Figure 28: Overview of environment synthesis and interaction for agentic RL. Agentic RL enables continuous learning through an interaction-learning loop, where agents collect trajectories from synthesized environments and enhance their capabilities.

shows that agent performance depends not only on policy optimization, but also on the coverage and quality of the environments used for training.

However, constructing high-quality environments manually is expensive and difficult to scale. Real-world systems may be inaccessible and costly. Purely language-based simulations are easier to build, but they may generate inconsistent state transitions (Li et al., 2026). Thus, recent studies explore procedural and programmatic environment synthesis, where executable programs and structured states are used to provide a scalable and reliable interaction experience.

This process is illustrated in Figure 28. It typically consists of the following steps:

- **Environment Specification.** We first define the state schema[13], available tools and APIs, and transition rules of the environment.
- **Programmatic Synthesis.** We then implement the environment as an executable system. This process typically converts the structured specification into program code, database tables, and callable tool interfaces.
- **Scenario Generation.** Based on the environment specification, we generate diverse initial states and user tasks. The tasks should cover different goals, constraints, and difficulty levels. For example, in a shopping environment, a simple task may ask the agent to purchase a single item, while a more difficult task may require it to compare multiple products.
- **Agent–Environment Interaction.** The agent interacts with the generated environment through multiple rounds of actions and observations. At each step, the agent selects a tool and provides the required arguments. The environment executes the action and returns an observation.
- **Outcome Verification.** After the interaction is completed, a verifier checks whether the task has been successfully solved. The verifier may inspect tool execution results, database states, or other structured records.
- **Agent Learning.** Finally, the verified trajectories are used to improve the agent. On the one hand, successful trajectories can serve as demonstrations for SFT, while task-level or step-level feedback can support RL training. On the other hand, failed interactions are also valuable because they reveal weaknesses in the current policy and gaps in the task distribution. This can guide the generation of targeted scenarios that exercise the agent's weak capabilities and provide more

[13]A state schema specifies the structured variables used to represent the current status of an environment, together with their types and possible values. For example, in a travel-booking environment, it may define fields for available flights, user preferences, and existing reservations.

informative experience for subsequent training (Chen et al., 2026; Sun et al., 2026). More broadly, this process can be viewed as learning from agentic experience, which we will discuss in Section 6.4.

Despite this progress, several important questions remain. First, how can we scale both the number and diversity of environments? Second, how can we ensure that generated environments are executable and verifiable? Third, how can agents generalize across heterogeneous and previously unseen environments?

One direction focuses on scaling environment diversity. RandomWorld procedurally generates executable tools and compositional tasks for tool-use agents (Sullivan et al., 2025). Unlike static datasets that contain only predefined trajectories, these generated environments support online execution and allow agents to explore different tool combinations during training. AgentScaler further constructs heterogeneous simulated environments and separates the learning of general function-calling behaviors from domain-specific adaptation (Fang et al., 2026). At a larger scale, DeepSeek-V3.2 synthesizes diverse environments and complex tasks for agentic post-training, showing that broad interaction experience is important for long-tail generalization (DeepSeek-AI, 2025).

Another direction focuses on executability. EnvScaler first constructs environment skeletons that specify state schemas, available tools, and interaction rules. It then generates multiple task scenarios within each environment (Song et al., 2026). Agent World Model follows a similar idea by implementing synthetic environments with executable code and database-backed states (Wang et al., 2026g). Compared with purely language-based simulations, these programmatic environments produce more consistent state transitions and generate tool outputs through actual execution.

Verifiability is equally important because RL requires reliable feedback. Instead of evaluating only the textual final response, an environment can check whether the agent has produced the desired state change. Given the final environment state s_T^e and the task goal g, we can define a state-based reward as

$$R_e(\tau) = V_e(s_T^e, g) \tag{76}$$

where $V_e(\cdot)$ may be implemented using executable tests, database queries, or rule-based validators. For example, in a database operation task, the verifier can directly check whether the required record has been inserted correctly. This provides more reliable feedback than comparing the generated answer with a reference text. Both EnvScaler and Agent World Model use such mechanisms to connect agent actions with observable state changes.

Scaling environments also introduces substantial heterogeneity. Different environments may vary in task difficulty, available tools, and interaction length. These differences can make RL training unstable and cause the agent to overfit to frequently sampled or easier environments. AutoForge addresses this issue by synthesizing difficult but verifiable tasks and estimating learning signals at the environment level (Cai et al., 2025). This design reduces the influence of unstable simulated interactions and improves training across heterogeneous environments.

6.3 Credit Assignment

Credit assignment is not unique to LLM-based agents. It is a long-standing challenge in traditional RL (Sutton, 1984; Zhou et al., 2020). When rewards are delayed, the agent must decide which past actions should be reinforced or suppressed. Classical methods, such as temporal-difference learning and eligibility traces, address this issue by propagating reward information backward along the trajectory. In agentic RL, credit assignment becomes more challenging because each interaction step may involve high-level planning, tool invocation, and environmental feedback, rather than a single low-level action. Therefore, credit assignment often needs to operate at both the trajectory level and the step level.

This challenge is especially important for agent planning. In many agent tasks, the environment only provides a sparse reward after the entire trajectory is completed. However, the final success or failure may come from an earlier planning decision, an incorrect tool invocation, or an ineffective response to an

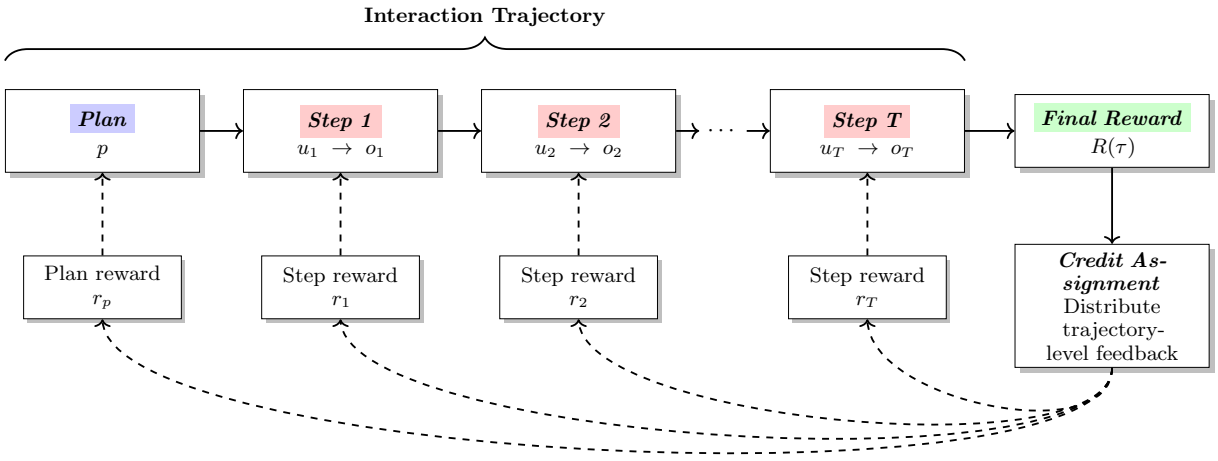


Figure 29: Illustration of credit assignment in agentic RL.

intermediate observation. If we directly assign the same trajectory-level reward to all interaction steps, the supervision can become noisy. The agent may fail to identify which decisions should be reinforced and which should be suppressed.

Formally, given a trajectory $\tau = [p, u_1, o_1, \cdots, u_T, o_T]$, a simple RL objective uses a trajectory-level reward $R(\tau)$ to optimize the whole planning process. However, such a reward only provides coarse feedback on the overall outcome. Therefore, credit assignment aims to decompose the trajectory-level reward into step-level signals:

$$R(\tau) \rightarrow \{r_p, r_1, \cdots, r_T\} \tag{77}$$

where r_p evaluates the quality of the generated plan, and r_t evaluates the contribution of the t-th interaction step (u_t, o_t) to the final task outcome.

The mechanism of credit assignment is illustrated in Figure 29. This decomposition enables the agent to distinguish whether failure comes from an unreasonable plan, an invalid tool call, or a poor adaptation to environmental feedback (Li et al., 2025b; Xi et al., 2026; Wang et al., 2026e). For example, consider the task of computing the babysitting payment and saving the result into a database. Suppose the agent generates a reasonable plan and correctly calls the calculator, but fails to invoke the database API with the required JSON format. If we only use a final trajectory-level reward, the whole trajectory may receive a low reward, e.g., $R(\tau) = 0.3$, even though the early planning and calculation steps are correct.

For illustration, we can define rule-based step-level rewards using predefined verification criteria, such as whether the plan includes all necessary steps, whether the calculation is correct, whether the JSON schema is valid, and whether the database update succeeds. These step-level rewards are diagnostic signals rather than a conservative numerical decomposition, so they need not sum to the trajectory-level reward. Then, we can use credit assignment to decompose the final trajectory-level feedback into step-level rewards:

$$R(\tau) = 0.3 \quad \Rightarrow \quad \{r_p = 0.4,\ r_1 = 1.0,\ r_2 = 1.0,\ r_3 = 0.7,\ r_4 = 0.0\} \tag{78}$$

where r_p indicates that the overall plan is reasonable, r_1 and r_2 indicate that the agent correctly extracts the numerical values and performs the calculation, r_3 indicates that the JSON formatting is partially correct, and r_4 indicates that the database update fails. In this way, the agent receives more precise feedback: it should preserve the correct planning and calculation steps while improving the database update step.

6.4 Learning from Agentic Experience

AI systems are gradually moving from an era dominated by human-generated data toward an era in which agents increasingly learn from their own experience (Silver & Sutton, 2025). As discussed in Section 6.1,

supervised data remains useful for teaching agents basic behaviors, such as planning, tool invocation, and response formatting. However, human demonstrations alone are unlikely to cover the full range of situations that an agent may encounter in complex and dynamic environments. One promising direction is to enable agents to learn from their own interaction experience. Through continuous interaction with environments, agents can collect successful and failed trajectories, and improve their future behaviors via these trajectories. In recent literature, this process is often referred to as *agent self-evolution*, where agents continuously enhance their capabilities by leveraging accumulated experience. Here we discuss three commonly used approaches for learning from agentic experience.

The first approach is **memory management** through agentic experience. In practice, an agent can store useful information from previous interactions. When facing a new task, the agent can retrieve relevant memories and use them as additional context to guide its decisions. From the perspective of in-context learning, this process can also be viewed as a form of *learning*, where the agent improves its behavior by incorporating previous experience into the current context. In this way, one example of learning from experience is to update the memory based on interaction outcomes. For example, given a task, the agent can store both successful and failed trajectories in its memory system. When solving similar tasks in the future, the agent can retrieve these experiences and avoid previous mistakes, thereby improving task performance. This approach is usually simple and does not require additional training of the agent.

The second approach is **skill optimization**. One challenge of using memory to learn from experience is scalability. As agent experience continuously accumulates, the memory size will keep growing. Moreover, a large amount of noisy and redundant information may also be introduced into the memory. As a result, efficiently storing and retrieving relevant information becomes increasingly difficult. Instead of storing individual experiences in memory, skill aims to identify common patterns across experiences and transform them into abstract behaviors. These skills provide a more compact representation of experience and allow the agent to transfer experience across different tasks. Similar to memory optimization, we can also update and refine the skills of agents to achieve learning from experience.

The third approach is to learn from experience via **trajectory refinement**. From this perspective, test-time scaling in agents can also be viewed as a form of learning from experience. Instead of updating the model parameters, the agent improves its current solution by leveraging immediate feedback during task execution. For example, the agent can use tool execution results or environment feedback to identify mistakes and refine its trajectory. Through repeated attempts and continuous refinement, the agent can gradually improve its decision-making process and achieve better task performance.

The three approaches mentioned above can be implemented through various techniques. Since this paper focuses on RL, we will introduce RL-based methods for these approaches in detail in the following sections. It is worth noting that, although these approaches do not always rely on traditional parametric updates, they can still be formulated under the RL paradigm described in Eqs. (69) and (70), where agents improve their behaviors via feedback from experience.

6.4.1 Memory Management

Memory management is a direct way for agents to learn from agentic experience. During interaction with an environment, an agent may observe useful information. If this information is discarded after the current task, the agent must solve similar problems from scratch in the future. Therefore, the goal of agent memory is to store useful information from previous interactions and retrieve it when needed. As illustrated in Figure 30, a typical memory system usually consists of three main stages (Chhikara et al., 2025):

- **Memory Retrieval.** The agent retrieves relevant memories from the memory bank to support the current interaction. The retrieved information can take various forms depending on how the memory bank is constructed. For example, the memory bank may contain successful trajectories from similar tasks, which can provide reusable solutions for the current decision. It may also store user-specific preferences, such as frequently selected hotels or preferred travel styles.

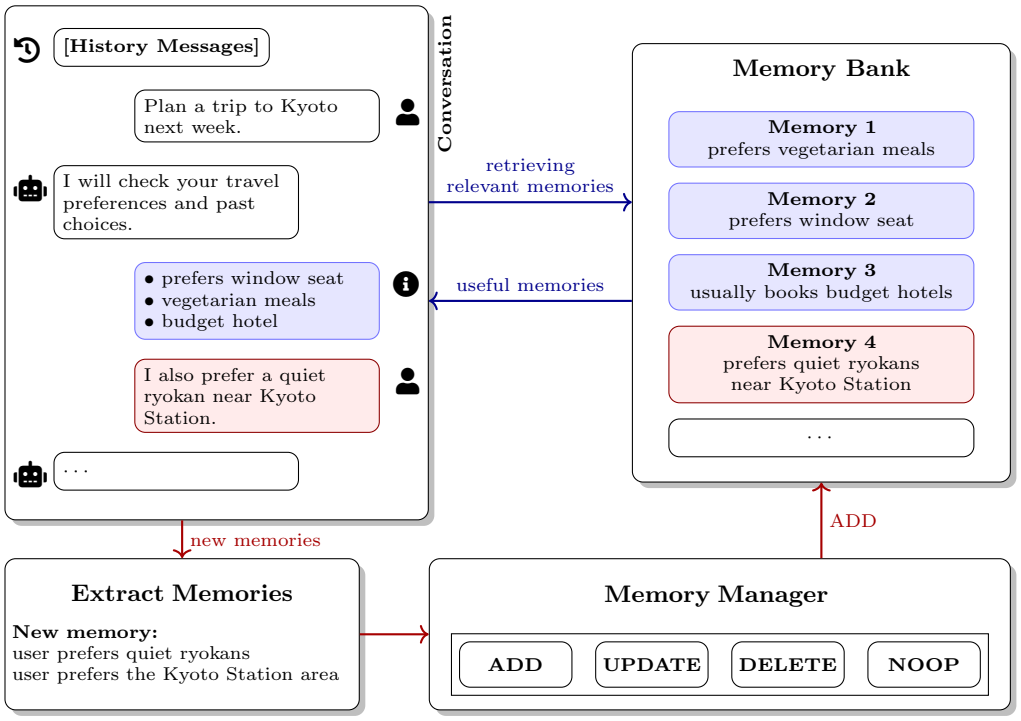


Figure 30: An overview of a memory system for agentic interaction. We take a travel-planning scenario as an example, where the user asks the agent to plan a trip to Kyoto. The agent first retrieves relevant user preferences from the memory bank, such as vegetarian meals, window seats, and budget hotels, and uses them to support the current conversation. During the interaction, the user provides new information, such as a preference for quiet ryokans near Kyoto Station. This new information is extracted as memory and passed to a memory manager, which updates the memory bank by selecting an operation such as `ADD`. Blue highlights denote retrieved memories, while red highlights denote newly extracted memories and their update path back to the memory bank.

- **Memory Extraction.** After the agent completes the current task based on retrieved information, useful information can be extracted from the interaction process and stored for future use. For example, if a user says that they are vegetarian, we can extract a memory such as "the user prefers vegetarian food". This process can be performed by the agent itself or by other agents.
- **Memory Update.** The agent updates the memory bank by integrating the newly extracted information with existing memories. A straightforward approach is to store all extracted information. However, this strategy is impractical because the amount of stored information continuously grows as the agent operates over time, making memory retrieval increasingly inefficient and introducing a large amount of redundant information. In contrast, a more advanced approach is to selectively update the memory. Specifically, the memory system can formulate memory maintenance as an operation selection problem. Given newly extracted information, a *memory manager* selects one of several operations: $\text{op} \in \mathcal{A}^m = \{\texttt{ADD}, \texttt{UPDATE}, \texttt{DELETE}, \texttt{NOOP}\}$, where `ADD` creates a new memory entry, `UPDATE` modifies an existing memory with newly observed information, `DELETE` removes outdated or contradictory memories, and `NOOP` keeps the memory bank unchanged. We can typically implement this memory manager by prompting an LLM to select appropriate operations based on the current interaction and existing memories.

In learning from experience, it is easy to observe that the performance of memory-based learning heavily depends on the accuracy of the memory manager. Although prompting an LLM can enable it to select memory operations, such memory systems still rely largely on the LLM's in-context decision-making ability

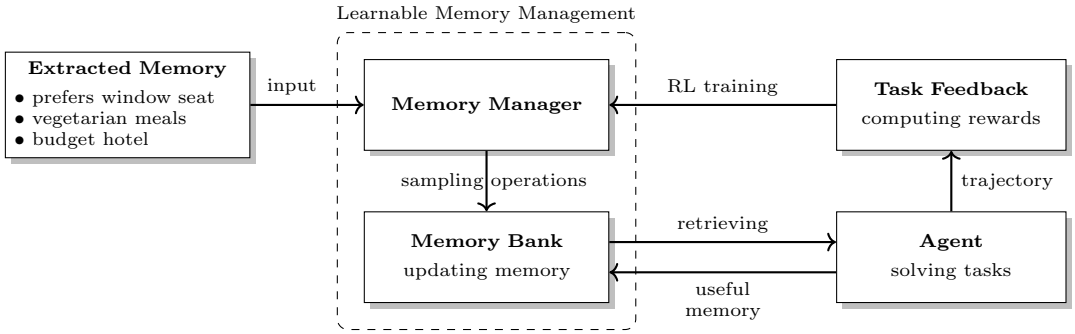


Figure 31: Overview of training a memory manager with RL.

or manually designed rules. As a result, they may struggle with complex memory updates and make incorrect decisions. For example, as shown in Yan et al. (2026)'s work, when a user first says "I adopted a dog named Buddy" and later adds "I adopted another dog named Scout", a vanilla memory system may incorrectly interpret the new information as a contradiction and perform a `DELETE`+`ADD` operation, overwriting the original memory. In contrast, a trained memory manager can recognize that the two statements are complementary and perform an `UPDATE` operation to consolidate the information into a more complete memory: "The user adopted two dogs, Buddy and Scout".

One promising approach to improve memory management is to optimize memory operations with RL (Yan et al., 2026). The key idea is to make memory management itself a learnable decision-making process. Given an extracted memory x^{mem} and an existing memory bank $\mathcal{M}_{\text{old}}$, the memory manager acts as a policy that selects a memory operation and generates the updated memory content:

$$(\text{op}, m') \sim \pi_\theta(\cdot \mid x^{\text{mem}}, \mathcal{M}_{\text{old}}) \tag{79}$$

where op denotes the selected memory operation from the operation set $\mathcal{A}^m$ and m' denotes the updated memory content. After applying the selected operation, the updated memory bank is provided to the agent for downstream task solving. By optimizing the memory manager with task-level feedback, the agent can learn when and how to update its memory based on the usefulness of stored experience. Figure 31 illustrates the training process of a memory manager with RL.

During optimization, the reward is defined according to the final task performance. If the updated memory helps the agent produce a correct answer, the memory manager receives a positive reward; otherwise, it receives a lower reward. A simple reward function can be defined as:

$$R_{\text{answer}} = \text{EM}(y_{\text{pred}}, y_{\text{gold}}) \tag{80}$$

where $\text{EM}(\cdot)$ denotes the Exact Match function, y_{pred} denotes the predicted answer, and y_{gold} denotes the ground-truth answer. This design avoids the need to manually annotate individual memory operations. Instead, the memory manager learns which operations are beneficial by directly optimizing their impact on downstream task performance.

After optimizing the memory manager, the agent can dynamically maintain its memory during future interactions. However, learning effective memory operations is only the first step toward experience-driven agent improvement. A key challenge is how to evaluate the long-term utility of accumulated memories. Existing memory optimization methods typically define rewards based on downstream task performance, i.e., whether the retrieved memories help the agent solve the current task. However, the usefulness of a memory is not always reflected immediately (Xu et al., 2025b). For example, a failure case collected from one interaction may not improve the current response, but it can help the agent avoid similar mistakes in future tasks. This indicates that memory can serve as an important component of agent learning, beyond simply storing past information.

Beyond optimizing memory update operations, another important question is how to construct and organize memories from accumulated experiences. Instead of treating memory construction as a fixed preprocessing step, we can enable an agent to learn effective memory construction strategies through RL. In this way, a straightforward approach is to formulate memory construction as a sequential decision-making process, where the agent learns how to select, organize, and transform interaction experiences into useful memories (Wang et al., 2025d).

6.4.2 Skill Optimization

A skill represents a higher-level abstraction of agentic experience. Different from memory, which focuses on storing useful information extracted from previous interactions, a skill aims to summarize recurring patterns across multiple experiences and transform them into reusable capabilities. Specifically, a skill can be viewed as a structured instruction package that describes when the skill should be invoked, what workflow it should follow, which tools are required, and how the final outcome should be verified. For example, a reasoning-oriented skill may be represented as follows:

Name: Math-Logic-Reasoning
Description: Solve complex mathematical, logical, tabular, and constraint reasoning tasks. Use when a task requires multi-step arithmetic, symbolic reasoning, calculator-backed verification, table analysis, optimization, scheduling or allocation constraints, Pareto trade-offs, proof checking, or translating word problems into equations, code, formulas, or executable checks.

Math Logic Reasoning
Use this skill for reasoning tasks where a wrong intermediate assumption can silently break the final answer. Prefer a compact formal model plus executable verification over unsupported mental arithmetic.

Workflow
1. Restate the target quantity, decision, or claim.
2. Extract givens, units, constraints, and hidden assumptions.
3. Convert the problem into equations, inequalities, tables, spreadsheet formulas, graphs, search spaces, constraint satisfaction problems, or short verification scripts.
...

Tool Choices
1. Use Python as a calculator for multi-step arithmetic, combinatorics, simulations, optimization search, or exact rational checks.
2. Use Fraction, Decimal, or sympy when exactness matters.
3. Use pandas for complex tables, joins, grouped summaries, rankings, or data-cleaning logic.
...

Verification Checklist
1. Recalculate the final numeric answer using a second method when feasible.
2. Test boundary cases: zero, negative values, equality limits, duplicate entries, and empty sets.
3. Confirm units after every transformation.
...

Output Style
Give the answer first when possible. Then include a concise derivation, the checked constraints, and any caveats.

In practice, the agent is provided with descriptions of all available skills in the prompt and learns to select appropriate skills based on the current task. These selected skills serve as high-level behavioral templates that guide the agent's planning and execution process. For example,

Available Skills:
Skill 1: Web Search
This skill enables the agent to retrieve and synthesize external information when the task requires up-to-date or domain-specific knowledge ···
Skill 2: Math-Logic-Reasoning
Solve complex mathematical, logical, tabular, and constraint reasoning tasks. Use when a task requires multi-step arithmetic, symbolic reasoning ···
{More skill descriptions}
User Query: Solve the equation $2x + 5 = 13$.
Selected Skill: Math-Logic-Reasoning

A straightforward approach to constructing skills for an agent is to manually write them according to specific task requirements. However, there can be many different ways to design a skill for the same task, similar to the process of writing prompts for LLMs. For example, one may define a skill with detailed instructions, including explicit workflows and execution procedures, while another may provide only high-level guidance and allow the agent to determine the detailed execution strategy. Such variations in skill design can significantly affect agent performance (Yu et al., 2026b).

One limitation of manual skill construction is that the quality and diversity largely depend on human experience. Therefore, if we want LLMs to handle a broad range of tasks, relying on human-annotated data for LLM fine-tuning is often inefficient. To address this limitation, an alternative approach is to automatically generate skills from agentic experiences. For example, we can prompt an LLM to summarize successful trajectories and extract reusable skills. These generated skills can then be added to the agent's skill bank and reused in future tasks.

The above way of generating skills often suffers from quality issues. First, a general LLM does not inherently know how to design effective skills without additional optimization. Second, the skill generator may optimize for the wrong objective because skill generation and skill utilization involve different contexts. Specifically, an LLM may consider a generated skill effective because it satisfies the instructions provided in the prompt, while another agent may fail to use this skill effectively in real-world tasks. As a result, skill generation should not only focus on producing well-structured skills, but also consider whether these skills can actually improve agent performance during task execution. Recent work has turned to RL to achieve this goal, where skill generation is optimized based on the actual performance of generated skills.

Here we consider **SkillRL** as an example to illustrate how to optimize skill generation through RL (Xia et al., 2026). The idea is that instead of directly generating skills from stored trajectories, we can first abstract reusable skills from existing trajectories and use them to guide agentic RL training. During the training process, the agent continuously discovers new skills and updates the skill bank, enabling the joint evolution of the agent and its skills. Figure 32 shows a schematic illustration of SkillRL. Here we give a brief outline of the key steps involved.

- Initially, we collect interaction trajectories from environments. These trajectories can include both successful and failed experiences, which provide diverse signals for skill discovery and optimization. Successful trajectories reveal effective behaviors, while failed trajectories help identify potential weaknesses and improvement opportunities.

- The collected trajectories are then used to extract reusable skills, which initialize the skill bank. Considering that agent capabilities usually include both general problem-solving strategies and task-specific procedures, we can organize skills at different levels of abstraction. Specifically, a hierarchical skill library can be constructed, where a general skill bank stores transferable skills learned across different tasks to improve generalization, while a task-specific skill bank preserves specialized strategies for solving particular tasks.

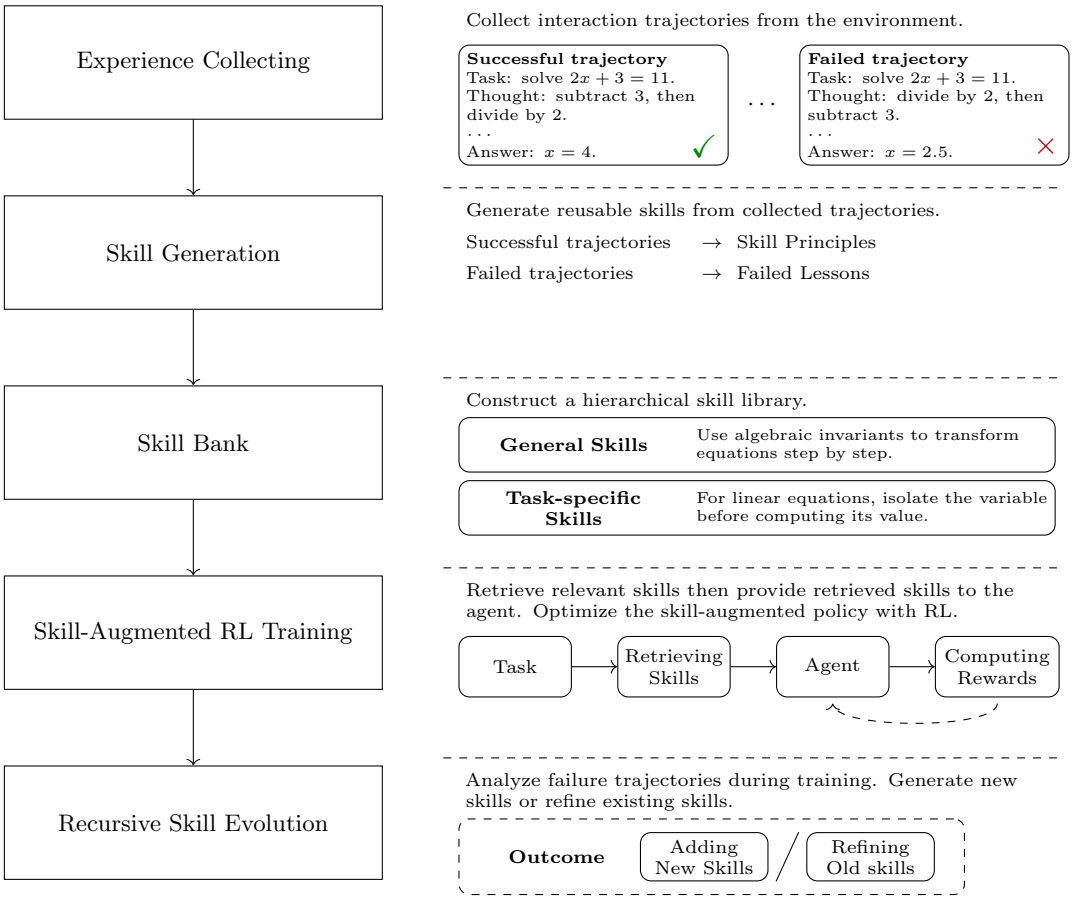


Figure 32: Illustration of SkillRL (Xia et al., 2026). SkillRL maintains a pool of agentic trajectories and initially abstracts reusable skills from these trajectories. The extracted skills are then used to guide RL training. As the agent improves through RL training, it generates higher-quality trajectories, which are further used to refine the skill bank.

- After initializing the skill bank, the agent retrieves relevant skills to guide the RL training process. Specifically, given task x, the agent selects suitable skills from the skill bank $\mathcal{S}$:

$$\mathcal{S}^* = \text{TopK}_{s_i \in \mathcal{S}} \text{Score}(x, s_i) \tag{81}$$

  where $\text{Score}(x, s_i)$ measures the relevance between the current task and each skill, and $\mathcal{S}^*$ denotes the selected skill set. The selected skills are then incorporated into the agent's decision-making process and used to guide RL optimization. These skills provide high-level behavioral priors, such as task decomposition and execution workflows, which help the agent explore more effective behaviors and improve the efficiency of RL training.
- Through interaction with the environment, the agent collects new trajectories. These trajectories are further analyzed to discover new skills or refine existing ones. In this way, the skill bank evolves together with the agent policy: the skill library provides prior knowledge to accelerate RL training, while improved trajectories continuously enrich and refine the skill library.

By repeating the above process, the agent can improve its skill library through accumulated experiences. Specifically, newly collected trajectories provide additional evidence for discovering new skills or refining existing ones, while the updated skill library further guides future RL training. This creates an iterative learning process in which the agent and its skills co-evolve over time. From this perspective, skill optimization provides a practical way to transform low-level interaction experiences into reusable capabilities, and has

become an important approach for enabling agents to continuously learn from their own experiences (Wang et al., 2026f; Fu et al., 2026).

6.4.3 Trajectory Refinement

Different from memory management and skill optimization, which focus on storing and abstracting experiences, trajectory refinement directly improves agent behaviors by revising generated trajectories. This idea is motivated by the self-refinement capability of LLMs. As discussed in Section 5.1, reasoning-enhanced RL training enables LLMs to evaluate their intermediate reasoning processes and revise incorrect solutions. For example, when solving mathematical problems, an LLM can verify intermediate results after each reasoning step and adjust its solution trajectory based on the verification outcomes.

A natural extension is to apply this self-refinement capability to LLM-based agents. During interaction with environments, agents continuously receive external feedback, such as tool execution results or environment states. Instead of blindly following the initial trajectory, agents can leverage these feedback signals to identify mistakes, revise previous decisions, and generate improved trajectories. In this way, agents can learn from their own interaction experiences by continuously refining their behaviors. One commonly used trajectory refinement approach is Reflexion (Shinn et al., 2023). The basic idea is to introduce an explicit reflection process into agent execution. Given a task, the agent first generates an initial trajectory and receives feedback from an evaluator. The agent then analyzes the feedback, summarizes potential errors or improvement strategies, and generates a reflection. Based on this reflection, the agent produces an improved trajectory in subsequent attempts. Here, we take mathematical reasoning as an example.

Task: Solve the equation $3x + 7 = 22$.

Initial Trajectory:
Thought: I need to isolate x by moving the constant term to the other side.
Action: Subtract 7 from both sides:

$$3x + 7 - 7 = 22 - 7$$

Observation: The equation becomes:

$$3x = 15$$

Thought: I need to divide both sides by 3 to obtain x.
Action: Calculate:

$$x = \frac{15}{3} = 4$$

Answer: $x = 4$.

Feedback: The answer is incorrect. This is because: $\frac{15}{3} = 5 \neq 4$.

Reflection: The previous trajectory failed because I made an error during intermediate calculation. For future tasks, I should carefully verify arithmetic operations and check the final answer by substituting it back into the original equation.

Refined Trajectory:
Thought: I need to isolate x and verify the final solution.
Action: Subtract 7 from both sides:

$$3x = 22 - 7 = 15$$

Action: Divide both sides by 3:

$$x = \frac{15}{3} = 5$$

Observation: Verify the solution:

$$3 \times 5 + 7 = 22$$

Answer: $x = 5$.

It is worth noting that we can obtain feedback from either internal evaluation or external evaluators. For example, we can use rule-based verifiers, LLM-based evaluators, or binary reward signals to provide feedback to the agent. Regardless of the feedback source or format, we need to convert it into a language-based representation, enabling the agent to understand errors and refine future behaviors. Here, inspired by the step-level feedback discussed in Section 5.1.2, we can further consider whether providing fine-grained feedback at each decision step can help agents generate better-refined trajectories. Compared with outcome-level feedback, step-level feedback provides more precise guidance about where and why the agent makes mistakes (Liu et al., 2024; Wang et al., 2025c). For example, when an agent fails to solve a mathematical reasoning problem, a binary reward only indicates that the final answer is incorrect, but it does not reveal which reasoning step contains the error or how the agent should revise its solution. However, obtaining step-level feedback is often expensive and challenging, as it requires evaluating intermediate decisions throughout the trajectory. To address this challenge, recent studies explore using methods such as MCTS to automatically construct step-level feedback signals (Yuan et al., 2025).

The above approaches mainly rely on prompting to enable trajectory refinement. Such approaches still face several limitations. First, their performance is bounded by the intrinsic refinement ability of the LLM. A pre-trained LLM may not naturally know how to interpret feedback and effectively revise its trajectory without additional optimization. Second, prompt-based refinement mainly improves the current trajectory at inference time, which limits its ability to accumulate and transfer experience across tasks. As discussed in Sections 6.4.1 and 6.4.2, failed trajectories contain valuable experiences about agent weaknesses and can provide useful learning signals for improving future behaviors. However, prompting-based approaches only leverage these experiences within the current interaction context. To address these limitations, recent studies explore training agents to acquire trajectory refinement abilities (Fu et al., 2025), either by updating model parameters or optimizing refinement behaviors based on collected experiences (Song et al., 2024). Interested readers can refer to these works for more information.

Although trajectory refinement does not explicitly update model parameters like traditional RL, it shares the fundamental principle of RL: improving agent behaviors based on feedback collected from interactions. From the perspective of in-context learning, the feedback obtained during trajectory refinement can be viewed as a temporary reward signal that modifies the agent's subsequent decision-making process. Instead of updating policy parameters, the agent updates its context with reflections, which potentially modifies the policy used for future actions within the current interaction. Therefore, trajectory refinement can be regarded as an in-context form of policy improvement. This perspective is closely related to the emerging research direction of *in-context RL* (namely ICRL), which studies how agents can adapt their behaviors through interaction feedback without explicit parameter updates (Monea et al., 2024).

7 Multimodal RL

Multimodal learning aims to build models that can process and integrate information from multiple modalities, such as vision and text. By combining complementary signals from different modalities, multimodal models can achieve more comprehensive perception and reasoning abilities than single-modality systems. These capabilities have enabled a wide range of applications, including image captioning, visual reasoning, speech understanding, and image generation (Baltrušaitis et al., 2018).

With the rapid development of LLMs, a natural direction is to extend their powerful reasoning and generation capabilities to multimodal domains. For example, visual language models can be constructed by aligning LLMs with visual encoders and training them on large-scale image-text pairs through supervised fine-tuning. However, extending LLM capabilities to new modalities introduces additional alignment challenges. Although an LLM may achieve strong alignment with human preferences in the language domain, such alignment does not necessarily transfer to other modalities. For example, a model may generate helpful textual responses while still producing visually inconsistent or misaligned outputs. Therefore, additional alignment strategies are required to optimize multimodal models according to modality-specific objectives.

RL provides a natural framework for addressing this challenge by optimizing multimodal models based on task-specific feedback signals. Compared with supervised fine-tuning, RL can directly optimize high-level objectives, such as human preferences, visual quality, and semantic consistency. In this section, we discuss the application of RL in different types of multimodal models. We first introduce RL for multimodal understanding models, where the output remains primarily textual but the input involves multiple modalities. We then discuss RL for multimodal generation models, including diffusion-based and flow matching-based models, where RL is used to optimize generated content quality.

7.1 Multimodal Understanding Models

Multimodal models can be roughly divided according to whether they generate non-textual content or use non-textual content as input. In this subsection, we focus on the latter case, namely multimodal understanding models. Although these models may process images, videos, audio, or other non-textual signals, their output is still usually a textual response. Therefore, from the perspective of RL training, they can still be viewed as conditional text generation models. The main difference from standard LLMs is that the condition now contains multimodal information in addition to a textual instruction. Once these inputs are encoded and passed into the LLM, the subsequent policy modeling, reward modeling, and RL training are largely consistent with the methods discussed in Section 3.

A representative example is the multimodal language model[14], especially the Visual Language Model (VLM). A VLM typically connects a visual encoder to an LLM through a linear projector or related alignment module, enabling the LLM to perform visual and language understanding. VLMs are currently the most explored extension in multimodal language research, and they form the foundation for many open-source multimodal language models, such as Qwen2.5-VL (Team, 2025) and LLaMA-3.2-11B-Vision (Grattafiori et al., 2024). For a VLM, the input usually consists of one or multiple images and a textual instruction, denoted by $(\mathbf{I}, \mathbf{x})$, where $\mathbf{I}$ represents the input images. These images are encoded into visual representations that are either concatenated with instruction embeddings or integrated into the LLM through cross-attention mechanisms. Since the model still generates a textual output $\mathbf{y}$, the RL training process for LLMs can be adapted to train VLMs with only minor changes (Yu et al., 2024a; Wang et al., 2025b; Zang et al., 2025; Ji et al., 2025).

Although the overall RL formulation can be reused, training VLMs with RL is still challenging in practice. The central difficulty is not the policy optimization algorithm itself, but the scarcity of high-quality multimodal preference data. Compared with textual preference data, visual preference data is more expensive to collect, harder to verify, and often more task-dependent. This makes it difficult to train a reliable visual reward model, which is a key component for applying RLHF-style methods to VLMs.

One straightforward way to alleviate this problem is to generate visual preference data through the automatic preference data generation method described in Section 4.1.1 (Yu et al., 2024b). Another alternative is a multi-stage training approach for the visual reward model, motivated by a simple idea: human preferences are already well captured in text, and these preferences can be transferred across modalities (Wang et al., 2025b). By leveraging textual preference data, this approach reduces the dependence on large-scale visual preference data when training a visual reward model. More specifically, as illustrated in Figure 33, we can train a visual reward model in the following three stages:

- Stage 1: pre-training with large-scale textual preference data. Given the transferability of human preferences across different modalities, we can begin by using large-scale textual preference data to pre-train the visual reward model. Note that the projector parameters are frozen without images.

[14]A multimodal language model is defined as a model that integrates an LLM with a multimodal encoder, such as CLIP (Radford et al., 2021), allowing the LLM to process inputs beyond text, such as images. Recent literature has also introduced the use of LLMs for generating outputs in non-text modalities, such as images and speech (Xu et al., 2025a; Zhang et al., 2024a). However, in this section, we focus on the former definition of multimodal language models.

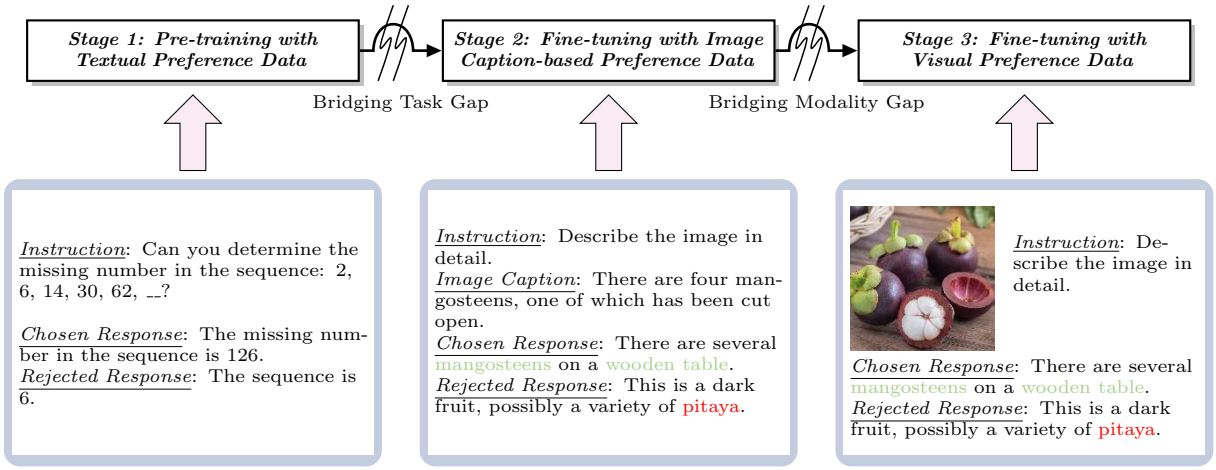


Figure 33: An overview of the multi-stage training approach for visual reward models.

This stage can be considered as providing a stronger starting point for training the visual reward model, as it enables the model to pre-learn general human preferences.

- Stage 2: fine-tuning with image caption-based preference data. Pre-learned human preferences cannot be directly applied to vision tasks due to both *task gap* and *modality gap*. To bridge the task gap, we fine-tune the reward model using image caption-based preference data in this stage. The rationale behind this approach is that general textual preference data does not cover vision-specific tasks, such as "Please describe the content in this image". We use such data to fine-tune the model, adapting it to vision-specific tasks. The projector parameters are frozen at this stage.
- Stage 3: fine-tuning with small-scale visual preference data. To further bridge the modality gap, we use visual preference data in the final stage to train the model. Note that in this phase, we also train the projector parameters.

In this process, not all preference data may align with the preferences used in subsequent stages, potentially leading to preference conflicts. To address this issue, we can enhance the visual reward model through data selection techniques, such as LESS (Xia et al., 2024). In fact, preference transfer is effective across modalities and has also been shown to work across different tasks and languages (Cheng et al., 2023; Wu et al., 2024). Interested readers can refer to these papers for more detailed discussions of these topics.

As discussed in Section 4.1.4, reward reasoning models have demonstrated superior performance in reward prediction. A natural question then arises: *can reward reasoning capabilities also be transferred from text to multimodal settings?* Recent work by Wang et al. (2026a) provides empirical evidence supporting this hypothesis. By following a similar multi-stage training paradigm, they show that reward reasoning capabilities can indeed be effectively transferred across modalities, further enhancing the performance of visual reward models.

Apart from constructing better preference data, another approach to improving the visual reward model is to integrate additional image content, such as image captions, into reward prediction (Sun et al., 2024b). This approach aims to achieve factually augmented reward prediction and reduce reward hacking. Specifically, in the original setup, the reward model predicts a reward based only on the image, input, and output; that is, the reward model's input is $[\mathbf{I}, \mathbf{x}, \mathbf{y}]$. In the factually augmented setup, the reward model also receives an additional textual image caption $\mathbf{C}$, resulting in an input of $[\mathbf{I}, \mathbf{C}, \mathbf{x}, \mathbf{y}]$. The basic idea is that the backbone of the visual reward model remains a well-trained LLM with in-context learning ability. By providing additional factual context about the image, we can help the reward model make more accurate and grounded reward predictions.

While our discussion primarily focuses on models for visual inputs in the context of visual language models, the RL techniques are adaptable across inputs of various modalities, such as video and audio.

7.2 Multimodal Generation Models

Unlike multimodal understanding models that typically produce textual responses, multimodal generation models aim to synthesize new content, such as images and videos. Recent advances in generative models, including diffusion models and flow matching models, have enabled high-quality content generation by learning complex data distributions. However, researchers have found that optimizing these models remains challenging because conventional maximum likelihood or reconstruction objectives cannot fully capture high-level human preferences, such as satisfying specific requirements (e.g., object quantities, colors, and visual layouts). To address this challenge, RL has been introduced to directly optimize multimodal generation models based on flexible reward signals (Lee et al., 2023; Fan et al., 2023). In this way, instead of relying solely on token-level or pixel-level reconstruction objectives, RL allows models to optimize task-specific criteria provided by humans or automatic evaluators. This property makes RL particularly suitable for multimodal generation, where generation quality is often difficult to describe through explicit supervision.

In this subsection, we describe the application of RL in two representative classes of multimodal generation models: diffusion models and flow matching models.

7.2.1 Diffusion Models

We first briefly introduce the basic generation process of diffusion-based models to provide the necessary notations. There are many aspects of diffusion models, such as image representations and mathematical formulations, that cannot be fully covered in this paper. Interested readers can refer to existing surveys on diffusion models for further details (Yang et al., 2023; Croitoru et al., 2023).

Specifically, diffusion models generate samples through a gradual denoising process (Sohl-Dickstein et al., 2015). Given a clean sample $\mathbf{x}_0$, the forward diffusion process progressively adds Gaussian noise to the sample. At each timestep t, the transition distribution is defined as:

$$Q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I}) \tag{82}$$

where $Q(\cdot)$ denotes the predefined forward diffusion process, β_t controls the noise magnitude at timestep t, and $\mathcal{N}(\mu, \sigma^2)$ denotes a Gaussian distribution with mean μ and variance σ^2. Through this forward process, the original data distribution is gradually transformed into a simple Gaussian noise distribution.

The generation process performs the reverse denoising procedure, where a neural network learns to recover clean samples from noisy inputs:

$$U_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \tag{83}$$

where $U_\theta(\cdot)$ denotes the learned reverse denoising process used for generation, and $\mu_\theta(\cdot)$ and $\Sigma_\theta(\cdot)$ denote the learned mean and variance of the reverse transition. By iteratively applying the denoising process from timestep T to 0, diffusion models can generate high-quality samples from random noise.

During training, diffusion models mainly focus on recovering the original data distribution, which can be regarded as a supervised learning objective. Similar to SFT in LLMs, such training paradigms optimize models toward the provided training data but do not explicitly consider human preferences. As a result, diffusion models may generate high-quality samples while still failing to satisfy fine-grained user requirements. For example, as described in Lee et al. (2023), although text-to-image diffusion models can generate visually realistic images, they may struggle with specific attributes, such as generating a desired number of objects. Therefore, how to incorporate additional learning objectives to better align diffusion models with human preferences has become an important research topic.

To address this challenge, researchers have explored RL as an effective approach for aligning diffusion models with human preferences. A straightforward approach is to treat generated samples as optimization targets and use external reward signals to guide the generation process. Specifically, given a text prompt $\mathbf{z}$, a reward function $R_{\text{dm}}(\mathbf{x}_0, \mathbf{z})$ evaluates the generated image $\mathbf{x}_0$, and the diffusion model is optimized to maximize the expected reward:

$$\max_{\theta} \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z}), \mathbf{x}_0 \sim p_\theta(\mathbf{x}_0|\mathbf{z})}[R_{\text{dm}}(\mathbf{x}_0, \mathbf{z})] \tag{84}$$

Here, the reward function can be instantiated by different types of evaluators depending on the optimization objective. For example, it can measure semantic alignment between the generated image and the text prompt using a vision-language model or measure human preferences through a learned reward model. Given a prompt requiring "three red apples on a table", a reward function can assess whether the generated image contains the correct object number and color.

However, directly optimizing this objective with the RL formulation introduced in Section 3.1 is not straightforward. This is because diffusion models generate samples through an iterative denoising process rather than an autoregressive generation process. Therefore, we need to redefine the RL formulation according to the characteristics of diffusion generation. Taking DPOK (Fan et al., 2023) as an example, recent studies observe that the reverse diffusion process naturally forms a multi-step trajectory, where each denoising step can be viewed as an action conditioned on the current noisy state. Based on this observation, the diffusion generation process can be formulated as an MDP. Specifically, we can consider the denoising procedure as a multi-step MDP and apply a policy gradient-based RL algorithm to optimize the reward obtained from generated images. The state corresponds to the current noisy latent representation, while the action represents the next denoising step:

$$s_t = (\mathbf{z}, \mathbf{x}_{T-t}), \quad a_t = \mathbf{x}_{T-t-1} \tag{85}$$

The policy is defined as the reverse diffusion transition:

$$\Pr_\theta(a_t|s_t) = U_\theta(\mathbf{x}_{T-t-1}|\mathbf{x}_{T-t}, \mathbf{z}) \tag{86}$$

where the final generated image receives the reward signal from the reward model. Based on this formulation, the optimization objective in Eq. (84) can be optimized using a simple policy gradient loss function:

$$\mathcal{L}_{\text{dmrl}}(\theta) = -\mathbb{E}_{\mathbf{z}\sim S_z, \tau_{\text{dm}}\sim \Pr_\theta(\cdot)}\left[\sum_{t=0}^{T-1} \log U_\theta(\mathbf{x}_{T-t-1}|\mathbf{x}_{T-t}, \mathbf{z}) R_{\text{dm}}(\mathbf{x}_0, \mathbf{z})\right] \tag{87}$$

where S_z denotes the training set, τ_{dm} denotes the denoising trajectory $\{\mathbf{x}_T, \mathbf{x}_{T-1}, \cdots, \mathbf{x}_0\}$. Here, many RL techniques developed for LLM alignment can be naturally extended to diffusion model optimization, such as incorporating KL regularization (Fan et al., 2023), introducing a reference diffusion model for importance sampling (Black et al., 2024), and applying GRPO-based optimization (Xue et al., 2025).

7.2.2 Flow Matching Models

Different from diffusion models, flow matching models perform multimodal generation based on ordinary differential equations (ODEs), where data transformation is modeled as a continuous-time flow. Instead of gradually adding and removing noise through a stochastic diffusion process, flow matching learns a continuous vector field that transports samples from a simple prior distribution to the target data distribution. In this subsection, we briefly introduce the training and generation processes of flow matching models, and then discuss how RL can be applied to optimize them. Note that we do not provide a detailed introduction to the underlying principles of flow matching models in this subsection. Interested readers can refer to existing tutorials for further details (Xiao et al., 2026).

Let $\mathbf{x}_0 \sim X_0$ denote a data sample from the target distribution and $\mathbf{x}_1 \sim X_1$ denote a noise sample from the prior distribution. Flow matching constructs an intermediate state by interpolating between the data

and noise distributions:

$$\mathbf{x}_t = (1-t)\mathbf{x}_0 + t\mathbf{x}_1 \tag{88}$$

where $t \in [0,1]$ denotes the continuous time variable. Based on this interpolation process, flow matching trains a neural network to predict the velocity field that describes how samples move along the trajectory:

$$\mathcal{L}_{\text{fm}}(\theta) = \mathbb{E}_{t,\mathbf{x}_0\sim X_0,\mathbf{x}_1\sim X_1}\left[\|\mathbf{v}_\theta(\mathbf{x}_t,t) - \mathbf{v}\|^2\right] \tag{89}$$

where $\mathbf{v} = \mathbf{x}_1 - \mathbf{x}_0$ denotes the target velocity field and $\mathbf{v}_\theta(\mathbf{x}_t,t)$ denotes the learned velocity function. After training, the generation process starts from a noise sample $\mathbf{x}_1$ and follows the learned continuous flow by solving the ordinary differential equation:

$$\frac{d\mathbf{x}_t}{dt} = \mathbf{v}_\theta(\mathbf{x}_t,t) \tag{90}$$

Although the continuous generation process of flow matching models can also be formulated as an MDP to enable RL optimization, this formulation faces a unique difficulty. Diffusion models naturally support stochastic sampling because Gaussian noise is progressively involved in the generation process. In contrast, flow matching models typically generate samples through deterministic ODEs, where the intermediate states are uniquely determined once the initial noise is given. This deterministic process creates two difficulties for online RL: it makes the transition probability required by policy-gradient optimization difficult to compute, and, more importantly, provides limited randomness for policy exploration.

To make RL applicable, Flow-GRPO converts the original ODE-based flow matching model into an equivalent stochastic differential equation (SDE) that preserves the marginal distributions while introducing stochasticity into the generation process (Liu et al., 2025a). The drift function of the converted SDE can be constructed from the original flow velocity field as:

$$\mathbf{v}_\theta^d(\mathbf{x}_t,t,\mathbf{z}) = \mathbf{v}_\theta(\mathbf{x}_t,t,\mathbf{z}) + \frac{\sigma_t^2}{2t}\left[\mathbf{x}_t + (1-t)\mathbf{v}_\theta(\mathbf{x}_t,t,\mathbf{z})\right] \tag{91}$$

where $\mathbf{v}_\theta(\cdot)$ denotes the original velocity field and $\mathbf{v}_\theta^d(\cdot)$ denotes the drift function after the ODE-to-SDE conversion. Therefore, the stochastic generation process can be written as:

$$d\mathbf{x}_t = \mathbf{v}_\theta^d(\mathbf{x}_t,t,\mathbf{z})dt + \sigma_t d\mathbf{w} \tag{92}$$

where $d\mathbf{w}$ denotes the Wiener process increment, i.e., the infinitesimal Gaussian noise added along the continuous sampling path.

In practice, we discretize the above SDE for trajectory sampling. Using Euler–Maruyama discretization, each sampling step can be written as:

$$\mathbf{x}_{t+\Delta t} = \mathbf{x}_t + \mathbf{v}_\theta^d(\mathbf{x}_t,t,\mathbf{z})\Delta t + \sigma_t\sqrt{\Delta t}\boldsymbol{\epsilon} \quad \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}) \tag{93}$$

The last term injects stochasticity into each generation step, allowing the model to sample diverse trajectories from the same condition and thereby enabling effective exploration for online RL. After discretization, each generation step corresponds to a stochastic Gaussian transition, which makes it possible to compute the transition probabilities required by GRPO.

Specifically, by discretizing the continuous flow trajectory, we can consider the generation process as a sequence of decision steps, where the model determines how to transform the current state toward the target data distribution (Liu et al., 2025a). Under this formulation, the state, action, transition, initial state distribution, and reward function are defined as follows. The state at timestep t is defined as:

$$s_t = (\mathbf{z}, t, \mathbf{x}_t) \tag{94}$$

The action corresponds to the next state predicted by the flow model:

$$a_t = \mathbf{x}_{t-\Delta t} \tag{95}$$

Since the flow model deterministically predicts the velocity field, the policy can be represented as:

$$\Pr_\theta(a_t|s_t) = \delta(a_t - (\mathbf{x}_t - \Delta t\, \mathbf{v}_\theta(\mathbf{x}_t, t, \mathbf{z}))) \tag{96}$$

where $\delta(\cdot)$ denotes the Dirac delta distribution, indicating that the next state is deterministically determined by the current state and the learned velocity field. After the ODE-to-SDE conversion and discretization described above, the transition probability becomes:

$$\Pr_\theta(a_t|s_t) = \mathcal{N}(a_t; \mu_\theta(\mathbf{x}_t, t, \mathbf{z}), \sigma_t^2 \mathbf{I}) \tag{97}$$

where $\mu_\theta(\cdot)$ is determined by the learned velocity field and σ_t controls the stochasticity during sampling. Based on this stochastic MDP formulation, RL algorithms can be applied to optimize flow matching models. Taking GRPO as an example, given a text condition $\mathbf{z}$, the flow model samples a group of generation trajectories $\{\tau_{\text{fm}}^i\}_{i=1}^G$, where each trajectory represents a complete sampling process from the initial noise to the final generated sample. We evaluate the final generated sample of each trajectory using the reward function and obtain a group of rewards: $\mathbf{R}_{\text{fm}} = \{R_{\text{fm}}(\mathbf{x}_0^1, \mathbf{z}), R_{\text{fm}}(\mathbf{x}_0^2, \mathbf{z}), \cdots, R_{\text{fm}}(\mathbf{x}_0^G, \mathbf{z})\}$, where R_{fm} denotes the reward function for evaluating the generated sample. Similar to diffusion models, the reward function can be instantiated with different types of evaluators depending on the optimization objective, such as vision-language models for measuring text-image alignment. The advantage of each trajectory is then estimated based on its relative reward within the sampled group:

$$\hat{A}_i = \frac{R_{\text{fm}}(\mathbf{x}_0^i, \mathbf{z}) - \text{mean}(\mathbf{R}_{\text{fm}})}{\text{std}(\mathbf{R}_{\text{fm}})} \tag{98}$$

Based on these estimated advantages, we formulate the GRPO objective as follows:

$$\mathcal{L}_{\text{fmgrpo}}(\theta) = -\mathbb{E}_{\mathbf{z}\sim S_z, \tau_{\text{fm}}\sim \Pr(\cdot)} \frac{1}{G}\sum_{i=1}^{G}\left[\frac{1}{T}\sum_t \min(\frac{\Pr_\theta(a_t^i|s_t^i)}{\Pr_{\theta_{\text{old}}}(a_t^i|s_t^i)}\hat{A}_i, \text{clip}(\frac{\Pr_\theta(a_t^i|s_t^i)}{\Pr_{\theta_{\text{old}}}(a_t^i|s_t^i)}, 1-\epsilon, 1+\epsilon)\hat{A}_i)\right] \tag{99}$$

8 Conclusions and Future Directions

In this paper, we have introduced the fundamental concepts of RL from the perspective of LLM training. We began with basic RL formulations and algorithms, including policy gradients, advantage estimation, importance sampling, and reward modeling, and explained them through LLM training examples. We then discussed recent advances in RL. We further extended the discussion to reasoning models, agentic systems, and multimodal models, showing how RL has evolved from a general policy optimization framework into a key paradigm for enhancing LLM reasoning, interactive decision-making, and multimodal generation capabilities.

While RL has become well established in LLM training and has achieved remarkable results, several promising directions remain for future exploration:

- **RL for pre-training.** Most existing studies focus on applying RL during post-training, while its role in pre-training remains less explored. Extending reward-driven learning to the pre-training stage may provide a way to shape model capabilities earlier in the training process. Recent studies have begun to demonstrate the potential of this direction (Dong et al., 2025), but achieving stable and scalable optimization at pre-training scale remains an important challenge.
- **More efficient RL training.** Recent methods, such as GRPO, have simplified RL training by removing components such as the critic model. However, RL still relies heavily on online sampling, which introduces substantial computational and time costs. Improving sampling efficiency and reducing the overall training cost therefore remain important directions for future research.

- **Predicting the scaling limits of RL.** The final gains from RL depend on many factors, including model capability, training data, reward quality, and optimization settings. In practice, we often need to complete expensive RL runs before knowing the final performance. Developing methods to predict the potential gains and performance limits of RL before full-scale training could significantly improve the efficiency of model development.
- **Continual and self-evolving RL.** Most current RL pipelines optimize a model on a fixed training distribution and stop once training is completed. One more ambitious direction is to enable models to continuously learn from their own interactions, feedback, and accumulated experiences after deployment. This requires new mechanisms for experience selection, memory and skill evolution, and stable policy updating without catastrophic forgetting.

Acknowledgements

This work was supported in part by the National Science Foundation of China (Nos. 62276056 and U24A20334), the Yunnan Fundamental Research Projects (No.202401BC070021), the Yunnan Science and Technology Major Project (No. 202502AD080014), the Liaoning Provincial Science and Technology Plan Project (No. 2026JH40/10100033), the Fundamental Research Funds for the Central Universities (Nos. N25BSS054 and N25BSS094), and the Program of Introducing Talents of Discipline to Universities, Plan 111 (No.B16009). The authors thank Ziming Zhu, Yuzhang Wu, Yifu Huo, Xihan Yang, and Kaiwei Wang for their valuable comments and discussions, which helped improve the manuscript.

Appendix: Datasets and Systems

Dataset Name	Use Case			Scale	Modality	Feedback / Reward Signal	
	RM	Rsn.	Agent			Source	Category
Anthropic/hh-rlhf	✔	✗	✗	169K		Human	Pairwise preference
stanfordnlp/SHP-2	✔	✗	✗	4.8M		Human / crowd	Pairwise preference
H4/stack-exchange-preferences	✔	✗	✗	10.7M		Crowd score	Score / ranking
Summarize from Feedback	✔	✗	✗	64.8K		Human	Pairwise preference
UltraFeedback	✔	✗	✗	64K / 340K		GPT-4	Scores, critiques, pairs
berkeley-nest/Nectar	✔	✗	✗	183K / 3.8M		GPT-4	7-way ranking
Skywork-Reward-Preference-80K	✔	✗	✗	80K		Mixed / curated	Pairwise preference
nvidia/HelpSteer	✔	✗	✗	37K		Human	Attribute scores
nvidia/HelpSteer2	✔	✗	✗	10K pairs		Human	Attribute scores / preference
nvidia/HelpSteer3	✔	✗	✗	40K pref.		Human	Preference, feedback, edits
Arena Human Preference 55K	✔	✗	✗	55K		Human users	Arena battle preference
VLFeedback	✔	✗	✗	80K / 380K		GPT-4V	Multimodal preference
RLHF-V-Dataset	✔	✗	✗	5.7K		Human	Fine-grained correction
openbmb/RLAIF-V-Dataset	✔	✗	✗	83K		AI feedback	Pairwise preference
OpenGVLab/MMPR-v1.2	✔	✔	✗	3M		AI / verifier	Multimodal reasoning preference
open-r1/OpenR1-Math-220k	✗	✔	✗	220K		Verifier / judge	Math reasoning traces
AI-MO/NuminaMath	✗	✔	✗	900K		Rule / GPT-4	Math CoT / TIR
PRIME-RL/Eurus-2-RL-Data	✗	✔	✗	482K		Verifier	Math/code outcome reward
AgentGym-RL-Data	✗	✔	✔	184K		Env.	Multi-turn agent reward
ALFWorld	✗	✗	✔	3.5K train		Env.	Text-world task reward
WebShop	✗	✗	✔	12K tasks		Env.	Web shopping reward
Search-R1 Data	✗	✔	✔	170K train		Retriever / rule	Search-tool QA reward
Sokoban	✗	✔	✔	Env. gen.		Env.	Puzzle-solving reward
Gym Cards	✗	✔	✔	Env. gen.		Env.	Logic-game reward
ToolBench	✗	✔	✔	126K		API execution	Tool-use trajectories

Table 4: Preference, reasoning, and trainable agentic RL datasets and environments for LLMs. Here, RM denotes reward modeling, Rsn. denotes reasoning, Env. denotes environment-based feedback, Env. gen. denotes procedurally generated environments, and TIR denotes tool-integrated reasoning.

System Name	Supported Modality				Supported Training Approaches
	Text	Image	Video	Audio	
TRL	✔	✔	✗	✗	SFT, Reward Modeling, PPO, GRPO, RLOO, DPO, KTO, ORPO, Online-DPO; Agentic RL via OpenEnv/Harbor.
OpenRLHF	✔	✔	✗	✗	SFT, RM, PPO, REINFORCE++, GRPO, RLOO, Dr.GRPO, DPO, KTO; Agentic RL with single-turn/multi-turn executors.
verl	✔	✔	✗	✗	SFT, PPO, GRPO, GSPO, ReMax, REINFORCE++, RLOO, DAPO, Dr.GRPO; Agentic RL via multi-turn tool calling.
verl-agent	✔	✔	✗	✗	GiGPO, GRPO, PPO, DAPO, GSPO, RLOO, REINFORCE++, LoRA; purpose-built Agentic RL for long-horizon LLM/VLM agents.
EasyR1	✔	✔	✗	✗	GRPO, DAPO, REINFORCE++, ReMax, RLOO, GSPO, CISPO; no native Agentic RL.
LLaMA-Factory	✔	✔	✔	✔	Pre-training, SFT, RM, PPO, DPO, KTO, ORPO, SimPO; no native Agentic RL.
VeOmni	✔	✔	✔	✔	Single-/multi-modal pre-training and post-training, SFT-style training, DPO, RL trainer backend; no native Agentic RL environment stack.
ms-swift	✔	✔	✔	✔	Pre-training, SFT, RM, PPO, GRPO, DPO, KTO, ORPO, CPO, SimPO, GKD; Agentic RL via multi-turn GRPO/tool-use training.
Align-Anything	✔	✔	✔	✔	SFT, RM, PPO, DPO, KTO, ORPO, SimPO, rule-based RL; Agent RL on roadmap.
OpenRLHF-M	✔	✔	✗	✗	PPO, GRPO, RLOO, Online-RLHF, Rejection Sampling for multimodal models; no native Agentic RL.
DeepSpeed-Chat	✔	✗	✗	✗	SFT, Reward Modeling, PPO-based RLHF; no native Agentic RL.
ROLL	✔	✔	✗	✗	SFT, DPO, distillation, PPO, GRPO, REINFORCE++, GSPO, RAFT++, StarPO, GiGPO; Agentic RL.
slime	✔	✗	✗	✗	PPO, GRPO, GSPO, REINFORCE++, OPD; Agentic RL via custom generation, tools, sandboxes, and verifier rewards.
OpenClaw-RL	✔	✔	✗	✗	Binary RL/GRPO, OPD, Hybrid RL, LoRA training; asynchronous Agentic RL for personalized agents, terminal, GUI, SWE, and tool-call settings.
SkyRL	✔	✗	✗	✗	GRPO, DAPO, async RL, Tinker-compatible training; Agentic RL for tool-use, search, SQL, and long-horizon tasks.
Agent Lightning	✔	✗	✗	✗	RL, APO, SFT-style optimization hooks; purpose-built Agentic RL for existing agent frameworks.
RAGEN	✔	✗	✗	✗	PPO/StarPO-style multi-turn optimization, rollout filtering, environment rewards; Agentic RL.
Search-R1	✔	✗	✗	✗	PPO, GRPO, REINFORCE for reasoning-search interleaved models; Agentic RL for search/tool use.
AReaL	✔	✗	✗	✗	PPO/GRPO-style asynchronous RL with agent-framework integration; Agentic RL.

Table 5: RL systems for LLM-based and multimodal post-training, including their supported modalities and corresponding training approaches.

References

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Afra Amini, Tim Vieira, and Ryan Cotterell. Direct preference optimization with an offset. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 9954–9972, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.592/.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Rémi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. A general theoretical paradigm to understand learning from human preferences. In Sanjoy Dasgupta, Stephan Mandt, and Yingzhen Li (eds.), *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume 238 of *Proceedings of Machine Learning Research*, pp. 4447–4455. PMLR, 2024. URL https://proceedings.mlr.press/v238/gheshlaghi-azar24a.html.

Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron C. Courville, and Yoshua Bengio. An actor-critic algorithm for sequence prediction. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=SJDaqqveg.

Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence*, 41(2):423–443, 2018.

Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

Kevin Black, Michael Janner, Yilun Du, Ilya Kostrikov, and Sergey Levine. Training diffusion models with reinforcement learning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=YCWjhGrJFD.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. *Advances in neural information processing systems*, 6, 1993.

Shihao Cai, Runnan Fang, Jialong Wu, Baixuan Li, Xinyu Wang, Yong Jiang, Liangcai Su, Liwen Zhang, Wenbiao Yin, Zhen Zhang, Fuli Feng, Pengjun Xie, and Xiaobin Wang. Autoforge: Automated environment synthesis for agentic reinforcement learning, 2025. URL https://arxiv.org/abs/2512.22857.

Kaiyan Chang, Songcheng Xu, Chenglong Wang, Yingfeng Luo, Xiaoqian Liu, Tong Xiao, and Jingbo Zhu. Efficient prompting methods for large language models: A survey, 2024. URL https://arxiv.org/abs/2404.01077.

Baian Chen, Chang Shu, Ehsan Shareghi, Nigel Collier, Karthik Narasimhan, and Shunyu Yao. Fireact: Toward language agent fine-tuning, 2023a. URL https://arxiv.org/abs/2310.05915.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. Accelerating large language model decoding with speculative sampling, 2023b. URL https://arxiv.org/abs/2302.01318.

Jianming Chen, Yawen Wang, Junjie Wang, Xiaofei Xie, Shoubin Li, Qing Wang, and Fanjiang Xu. Where did it go wrong? capability-oriented failure attribution for vision-and-language navigation agents. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 28132–28150, San Diego, California, United States, 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. URL https://aclanthology.org/2026.findings-acl.1402/.

Lichang Chen, Shiyang Li, Jun Yan, Hai Wang, Kalpa Gunaratna, Vikas Yadav, Zheng Tang, Vijay Srinivasan, Tianyi Zhou, Heng Huang, and Hongxia Jin. Alpagasus: Training a better alpaca with fewer data. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=FdVXgSJhvz.

Mingyang Chen, Linzhuang Sun, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. Research: Learning to reason with search for llms via reinforcement learning. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/7b3a0d3a0864b66f229c0a9c84f9e950-Abstract-Conference.html.

Pengyu Cheng, Jiawen Xie, Ke Bai, Yong Dai, and Nan Du. Everyone deserves a reward: Learning customized human preferences, 2023. URL https://arxiv.org/abs/2309.03126.

Prateek Chhikara, Dev Khant, Saket Aryan, Taranjeet Singh, and Deshraj Yadav. Mem0: Building production-ready ai agents with scalable long-term memory, 2025. URL https://arxiv.org/abs/2504.19413.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 4299–4307, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/d5e2c0adad503c91f91df240d0cd4e49-Abstract.html.

Thomas Coste, Usman Anwar, Robert Kirk, and David Krueger. Reward model ensembles help mitigate overoptimization. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dcjtMYkpXx.

Florinel-Alin Croitoru, Vlad Hondru, Radu Tudor Ionescu, and Mubarak Shah. Diffusion models in vision: A survey. *IEEE transactions on pattern analysis and machine intelligence*, 45(9):10850–10869, 2023.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. ULTRAFEEDBACK: boosting language models with scaled AI feedback. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=BOorDpKHiJ.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Bingxiang He, Wei Zhu, Yuan Ni, Guotong Xie, Ruobing Xie, Yankai Lin, Zhiyuan Liu, and Maosong Sun. ULTRAFEEDBACK: boosting language models with scaled AI feedback. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=BOorDpKHiJ.

Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. Multi-objective optimization. In *Decision sciences*, pp. 161–200. CRC Press, 2016.

Deepseek. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

DeepSeek-AI. Deepseek-v3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Mingkai Deng, Jianyu Wang, Cheng-Ping Hsieh, Yihan Wang, Han Guo, Tianmin Shu, Meng Song, Eric Xing, and Zhiting Hu. Rlprompt: Optimizing discrete text prompts with reinforcement learning. In *Proceedings of the 2022 conference on empirical methods in natural language processing*, pp. 3369–3391, 2022.

Hongxin Ding, Baixiang Huang, Yue Fang, Weibin Liao, Zheng Li, Jinyang Zhang, Zhijing Wu, Junfeng Zhao, and Yasha Wang. Evorubrics: Dynamic rubrics as rewards via adversarial co-evolution for llm reinforcement learning, 2026. URL https://arxiv.org/abs/2606.23038.

Qingxiu Dong, Li Dong, Yao Tang, Tianzhu Ye, Yutao Sun, Zhifang Sui, and Furu Wei. Reinforcement pre-training, 2025. URL https://arxiv.org/abs/2506.08007.

Yann Dubois, Chen Xuechen Li, Rohan Taori, Tianyi Zhang, Ishaan Gulrajani, Jimmy Ba, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Alpacafarm: A simulation framework for methods that learn from human feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/5fc47800ee5b30b8777fdd30abcaaf3b-Abstract-Conference.html.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, DJ Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, et al. Helping or herding? reward model ensembles mitigate but do not eliminate reward hacking, 2023. URL https://arxiv.org/abs/2312.09244.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. Kto: Model alignment as prospect theoretic optimization, 2024. URL https://arxiv.org/abs/2402.01306.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, 2018.

Ying Fan, Olivia Watkins, Yuqing Du, Hao Liu, Moonkyung Ryu, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, Kangwook Lee, and Kimin Lee. Dpok: Reinforcement learning for fine-tuning text-to-image diffusion models. *Advances in neural information processing systems*, 36:79858–79885, 2023.

Runnan Fang, Shihao Cai, Baixuan Li, Jialong Wu, Guangyu Li, Wenbiao Yin, Xinyu Wang, Xiaobin Wang, Liangcai Su, Zhen Zhang, Shibin Wu, Zhengwei Tao, Yong Jiang, Pengjun Xie, Ningyu Zhang, Fei Huang, Wentao Zhang, and Jingren Zhou. Towards general agentic intelligence via environment scaling. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 17610–17621, San Diego, California, United States, 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. URL https://aclanthology.org/2026.findings-acl.872/.

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms, 2025. URL https://arxiv.org/abs/2504.11536.

Evan Frick, Tianle Li, Connor Chen, Wei-Lin Chiang, Anastasios Nikolas Angelopoulos, Jiantao Jiao, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. How to evaluate reward models for RLHF. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=cbttLtO94Q.

Dayuan Fu, Keqing He, Yejie Wang, Wentao Hong, Zhuoma Gongque, Weihao Zeng, Wei Wang, Jingang Wang, Xunliang Cai, and Weiran Xu. Agentrefine: Enhancing agent generalization through refinement tuning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=FDimWzmcWn.

Zenghuang Fu, Zhaoyang Li, Qiuyuan Ai, Haoyu Wu, Minghui Wu, Chenxu Zhao, Ante Wang, Guannan He, and Changwei Wang. Self-play meets skill evolution: Self-evolving search agents that pose, solve, and remember, 2026. URL https://arxiv.org/abs/2607.29468.

Víctor Gallego. Refined direct preference optimization with synthetic data for behavioral alignment of llms, 2024. URL https://arxiv.org/abs/2402.08005.

Leo Gao, John Schulman, and Jacob Hilton. Scaling laws for reward model overoptimization. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10835–10866. PMLR, 2023. URL https://proceedings.mlr.press/v202/gao23h.html.

Alexey Gorbatovski, Boris Shaposhnikov, Alexey Malakhov, Nikita Surnachev, Yaroslav Aksenov, Ian Maksimov, Nikita Balagansky, and Daniil Gavrilov. Learn your reference model for real good alignment. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=H0qIWXXLUR.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

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

Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, volume 2024, pp. 34133–34156, 2024.

Alex Havrilla, Yuqing Du, Sharath Chandra Raparthy, Christoforos Nalmpantis, Jane Dwivedi-Yu, Maksym Zhuravinskyi, Eric Hambro, Sainbayar Sukhbaatar, and Roberta Raileanu. Teaching large language models to reason with reinforcement learning, 2024. URL https://arxiv.org/abs/2403.04642.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Jiwoo Hong, Noah Lee, and James Thorne. ORPO: Monolithic preference optimization without reference model. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 11170–11189, Miami, Florida, USA, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-main.626/.

Jian Hu. Reinforce++: A simple and efficient approach for aligning large language models, 2025. URL https://arxiv.org/abs/2501.03262.

Mengkang Hu, Pu Zhao, Can Xu, Qingfeng Sun, Jian-Guang Lou, Qingwei Lin, Ping Luo, and Saravan Rajmohan. Agentgen: Enhancing planning abilities for large language model based agent via environment and task generation. In Yizhou Sun, Flavio Chierichetti, Hady W. Lauw, Claudia Perlich, Wee Hyong Tok, and Andrew Tomkins (eds.), *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining, V.1, KDD 2025, Toronto, ON, Canada, August 3-7, 2025*, pp. 496–507. ACM, 2025. URL https://doi.org/10.1145/3690624.3709321.

Yifu Huo, Shunjie Xing, Chenglong Wang, Peinan Feng, Qiaozhi He, Yan Ding, Anxiang Ma, Yuxin Gao, Tongran Liu, Tong Xiao, and Jingbo Zhu. Learning from environmental feedback: Credit assignment across multiple timescales for agentic reinforcement learning, 2026. URL https://arxiv.org/abs/2608.08255.

Jiaming Ji, Xinyu Chen, Rui Pan, Han Zhu, Conghui Zhang, Jiahao Li, Donghai Hong, Boyuan Chen, Jiayi Zhou, Kaile Wang, et al. Safe rlhf-v: Safe reinforcement learning from human feedback in multimodal large language models, 2025. URL https://arxiv.org/abs/2503.17682.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning, 2025. URL https://arxiv.org/abs/2503.09516.

Mojtaba Komeili, Kurt Shuster, and Jason Weston. Internet-augmented dialogue generation. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8460–8478, Dublin, Ireland, 2022. Association for Computational Linguistics. URL https://aclanthology.org/2022.acl-long.579/.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. Training language models to self-correct via reinforcement learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=CjwERcAU7w.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. RewardBench: Evaluating reward models for language modeling. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1755–1797, Albuquerque, New Mexico, 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. URL https://aclanthology.org/2025.findings-naacl.96/.

Joonho Lee, Marko Bjelonic, Alexander Reske, Lorenz Wellhausen, Takahiro Miki, and Marco Hutter. Learning robust autonomous navigation and locomotion for wheeled-legged robots. *Science Robotics*, 9 (89):eadi9641, 2024.

Kimin Lee, Hao Liu, Moonkyung Ryu, Olivia Watkins, Yuqing Du, Craig Boutilier, Pieter Abbeel, Mohammad Ghavamzadeh, and Shixiang Shane Gu. Aligning text-to-image models using human feedback, 2023. URL https://arxiv.org/abs/2302.12192.

Minghao Li, Yingxiu Zhao, Bowen Yu, Feifan Song, Hangyu Li, Haiyang Yu, Zhoujun Li, Fei Huang, and Yongbin Li. API-bank: A comprehensive benchmark for tool-augmented LLMs. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 3102–3116, Singapore, 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-main.187/.

Xuefeng Li, Haoyang Zou, and Pengfei Liu. Limr: Less is more for rl scaling, 2025a. URL https://arxiv.org/abs/2502.11886.

Yixia Li, Hongru Wang, Jiahao Qiu, Zhenfei Yin, Dongdong Zhang, Cheng Qian, Zeping Li, Xiaoteng Ma, Guanhua Chen, and Heng Ji. From word to world: Can large language models be implicit text-based world models? In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8084–8111, San Diego, California, United States, 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.366/.

Zhiwei Li, Yong Hu, and Wenqing Wang. Encouraging good processes without the need for good answers: Reinforcement learning for LLM agent planning. In Saloni Potdar, Lina Rojas-Barahona, and Sebastien Montella (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 1654–1666, Suzhou (China), 2025b. Association for Computational Linguistics. ISBN 979-8-89176-333-3. URL https://aclanthology.org/2025.emnlp-industry.116/.

Ziniu Li, Tian Xu, Yushun Zhang, Zhihang Lin, Yang Yu, Ruoyu Sun, and Zhi-Quan Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=Stn8hXkpe6.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=v8L0pN6EOi.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=v8L0pN6EOi.

Jie Liu, Gongye Liu, Jiajun Liang, Yangguang Li, Jiaheng Liu, Xintao Wang, Pengfei Wan, Di Zhang, and Wanli Ouyang. Flow-grpo: Training flow matching models via online RL. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025a. URL http://papers.nips.cc/paper_files/paper/2025/hash/3a10c46572628d58cb44fb705f25cbbf-Abstract-Conference.html.

Tianci Liu, Ran Xu, Tony Yu, Ilgee Hong, Carl Yang, Tuo Zhao, and Haoyu Wang. OpenRubrics: Towards scalable synthetic rubric generation for reward modeling and LLM alignment. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 17417–17437, San Diego, California, United States, 2026a. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.791/.

Xiaoqian Liu, Ke Wang, Yuchuan Wu, Fei Huang, Yongbin Li, Jianbin Jiao, and Junge Zhang. Agentic reinforcement learning with implicit step rewards. In *International Conference on Learning Representations*, volume 2026, pp. 129271–129291, 2026b.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: NLG evaluation using gpt-4 with better human alignment. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, Singapore, 2023. Association for Computational Linguistics. URL https://aclanthology.org/2023.emnlp-main.153/.

Yantao Liu, Zijun Yao, Rui Min, Yixin Cao, Lei Hou, and Juanzi Li. Rm-bench: Benchmarking reward models of language models with subtlety and style. In *The Thirteenth International Conference on*

Learning Representations, ICLR 2025, Singapore, April 24-28, 2025. OpenReview.net, 2025b. URL https://openreview.net/forum?id=QEHrmQPBdd.

Zhiwei Liu, Weiran Yao, Jianguo Zhang, Rithesh Murthy, Liangwei Yang, Zuxin Liu, Tian Lan, Ming Zhu, Juntao Tan, Shirley Kokane, Thai Hoang, Juan Carlos Niebles, Shelby Heinecke, Huan Wang, Silvio Savarese, and Caiming Xiong. PRACT: Optimizing principled reasoning and acting of LLM agent. In Libby Barak and Malihe Alikhani (eds.), *Proceedings of the 28th Conference on Computational Natural Language Learning*, pp. 442–446, Miami, FL, USA, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.conll-1.33/.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. The flan collection: Designing data and methods for effective instruction tuning. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pp. 22631–22648. PMLR, 2023. URL https://proceedings.mlr.press/v202/longpre23a.html.

Dakota Mahan, Duy Van Phung, Rafael Rafailov, Chase Blagden, Nathan Lile, Louis Castricato, Jan-Philipp Fränken, Chelsea Finn, and Alon Albalak. Generative reward models, 2024. URL https://arxiv.org/abs/2410.12832.

Saumya Malik, Valentina Pyatkin, Sander Land, Jacob Morrison, Noah Smith, Hanna Hajishirzi, and Nathan Lambert. Rewardbench 2: Advancing reward model evaluation. In *International Conference on Learning Representations*, volume 2026, pp. 144839–144866, 2026.

Saeed Masoudnia and Reza Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42:275–293, 2014.

Igor Melnyk, Youssef Mroueh, Brian Belgodere, Mattia Rigotti, Apoorva Nitsure, Mikhail Yurochkin, Kristjan H. Greenewald, Jirí Navrátil, and Jarret Ross. Distributional preference alignment of llms via optimal transport. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/bce96750dcb64f3257ceabadf0b9c9bf-Abstract-Conference.html.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e099c1c9699814af0be873a175361713-Abstract-Conference.html.

Kaisa Miettinen. *Nonlinear multiobjective optimization*, volume 12. Springer Science & Business Media, 1999.

Kaisa Miettinen, Francisco Ruiz, and Andrzej P Wierzbicki. Introduction to multiobjective optimization: interactive approaches. In *Multiobjective optimization: interactive and evolutionary approaches*, pp. 27–57. Springer, 2008.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In

Maria-Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pp. 1928–1937. JMLR.org, 2016. URL http://proceedings.mlr.press/v48/mniha16.html.

Giovanni Monea, Antoine Bosselut, Kianté Brantley, and Yoav Artzi. Llms are in-context reinforcement learners. 2024.

Tetsuro Morimura, Mitsuki Sakamoto, Yuu Jinnai, Kenshi Abe, and Kaito Ariu. Filtered direct preference optimization. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 22729–22770, Miami, Florida, USA, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-main.1266/.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. Webgpt: Browser-assisted question-answering with human feedback, 2021. URL https://arxiv.org/abs/2112.09332.

OpenAI. Learning to reason with llms, 2024. URL https://openai.com/index/learning-to-reason-with-llms/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, 2022. URL http://papers.nips.cc/paper_files/paper/2022/hash/b1efde53be364a73914f58805a001731-Abstract-Conference.html.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model connected with massive apis. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/e4c61f578ff07830f5c37378dd3ecb0d-Abstract-Conference.html.

Reiner Pope, Sholto Douglas, Aakanksha Chowdhery, Jacob Devlin, James Bradbury, Jonathan Heek, Kefan Xiao, Shivani Agrawal, and Jeff Dean. Efficiently scaling transformer inference. In Dawn Song, Michael Carbin, and Tianqi Chen (eds.), *Proceedings of the Sixth Conference on Machine Learning and Systems, MLSys 2023, Miami, FL, USA, June 4-8, 2023*. mlsys.org, 2023. URL https://proceedings.mlsys.org/paper_files/paper/2023/hash/c4be71ab8d24cdfb45e3d06dbfca2780-Abstract-mlsys2023.html.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/97c5b2707228e7e3fb67e4ecc2e0e607-Abstract-Conference.html.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li,

Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=dHng2O0Jjr.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 2021. URL http://proceedings.mlr.press/v139/radford21a.html.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D. Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/a85b405ed65c6477a4fe8302b5e06ce7-Abstract-Conference.html.

MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Yunzhong He, and Afra Feyza Akyürek. Online rubrics elicitation from pairwise comparisons, 2025. URL https://arxiv.org/abs/2510.07284.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

John Schulman, Sergey Levine, Pieter Abbeel, Michael I. Jordan, and Philipp Moritz. Trust region policy optimization. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 1889–1897. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/schulman15.html.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1506.02438.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Amrith Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. Rewarding progress: Scaling automated process verifiers for LLM reasoning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=A6Y7AqlzLW.

Ruichen Shao, Bei Li, Gangao Liu, Yang Chen, ZhouXiang, Jingang Wang, Xunliang Cai, and Peng Li. Earlier tokens contribute more: Learning direct preference optimization from temporal decay perspective. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=OspqtLVUN5.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

William F. Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. Rethinking rubric generation for improving llm judge and reward modeling for open-ended tasks, 2026. URL https://arxiv.org/abs/2602.05125.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

David Silver and Richard S Sutton. Welcome to the era of experience. *Google AI*, 1:11, 2025.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial intelligence*, 299:103535, 2021.

Joykirat Singh, Raghav Magazine, Yash Pandya, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning, 2025a. URL https://arxiv.org/abs/2505.01441.

Joykirat Singh, Yash Pandya, Pranav Vajreshwari, Raghav Magazine, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*, 2025b.

Joykirat Singh, Yash Pandya, Pranav Vajreshwari, Raghav Magazine, and Akshay Nambi. Agentic reasoning and tool integration for llms via reinforcement learning. In *First Workshop on Foundations of Reasoning in Language Models*, 2025c.

Prasann Singhal, Tanya Goyal, Jiacheng Xu, and Greg Durrett. A long way to go: Investigating length correlations in rlhf, 2023. URL https://arxiv.org/abs/2310.03716.

Jascha Sohl-Dickstein, Eric A. Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In Francis R. Bach and David M. Blei (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 2256–2265. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/sohl-dickstein15.html.

Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. EnvScaler: Scaling tool-interactive environments for LLM agent via programmatic synthesis. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 8326–8357, San Diego, California, United States, 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. URL https://aclanthology.org/2026.findings-acl.407/.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization of LLM agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1:*

Long Papers), pp. 7584–7600, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.409/.

Michael Sullivan, Mareike Hartmann, and Alexander Koller. Procedural environment generation for tool-use agents. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 18544–18562, Suzhou, China, 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. URL https://aclanthology.org/2025.emnlp-main.936/.

Mingjie Sun, Zhuang Liu, Anna Bair, and J. Zico Kolter. A simple and effective pruning approach for large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024a. URL https://openreview.net/forum?id=PxoFut3dWW.

Xueqiao Sun, Xiaohan Wang, Ludwig Schmidt, Serena Yeung-Levy, and Yuhui Zhang. Learning from failure: Inference-time self-improvement for computer-use agents, 2026. URL https://arxiv.org/abs/2606.31270.

Zhiqing Sun, Sheng Shen, Shengcao Cao, Haotian Liu, Chunyuan Li, Yikang Shen, Chuang Gan, Liangyan Gui, Yu-Xiong Wang, Yiming Yang, Kurt Keutzer, and Trevor Darrell. Aligning large multimodal models with factually augmented RLHF. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 13088–13110, Bangkok, Thailand, 2024b. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.775/.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction (2nd ed.)*. The MIT Press, 2018.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. University of Massachusetts Amherst, 1984.

Csaba Szepesvári. Algorithms for reinforcement learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 4(1):1–103, 2010.

Sijun Tan, Siyuan Zhuang, Kyle Montgomery, William Yuan Tang, Alejandro Cuadron, Chenguang Wang, Raluca A. Popa, and Ion Stoica. Judgebench: A benchmark for evaluating llm-based judges. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=G0dksFayVq.

Kimi Team, Yifan Bai, Yiping Bao, Y Charles, Cheng Chen, Guanduo Chen, Haiting Chen, Huarong Chen, Jiahao Chen, Ningxin Chen, et al. Kimi k2: Open agentic intelligence, 2025a. URL https://arxiv.org/abs/2507.20534.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms, 2025b. URL https://arxiv.org/abs/2501.12599.

Qwen Team. Qwen2.5-vl, 2025. URL https://qwenlm.github.io/blog/qwen2.5-vl/.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh

Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. Llama 2: Open foundation and fine-tuned chat models, 2023. URL https://arxiv.org/abs/2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.

Chenglong Wang, Hang Zhou, Kaiyan Chang, Tongran Liu, Chunliang Zhang, Quan Du, Tong Xiao, and Jingbo Zhu. Learning evaluation models from large language models for sequence generation, 2023a. URL https://arxiv.org/abs/2308.04386.

Chenglong Wang, Hang Zhou, Kaiyan Chang, Bei Li, Yongyu Mu, Tong Xiao, Tongran Liu, and JingBo Zhu. Hybrid alignment training for large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 11389–11403, Bangkok, Thailand, 2024a. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.676/.

Chenglong Wang, Hang Zhou, Yimin Hu, Yifu Huo, Bei Li, Tongran Liu, Tong Xiao, and Jingbo Zhu. ESRL: efficient sampling-based reinforcement learning for sequence generation. In Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (eds.), *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, pp. 19107–19115. AAAI Press, 2024b. URL https://doi.org/10.1609/aaai.v38i17.29878.

Chenglong Wang, Yang Gan, Yifu Huo, Yongyu Mu, Qiaozhi He, Murun Yang, Bei Li, Tong Xiao, Chunliang Zhang, Tongran Liu, and JingBo Zhu. GRAM: A generative foundation reward model for reward generalization. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, volume 267 of *Proceedings of Machine Learning Research*. PMLR / OpenReview.net, 2025a. URL https://proceedings.mlr.press/v267/wang25ad.html.

Chenglong Wang, Yang Gan, Yifu Huo, Yongyu Mu, Murun Yang, Qiaozhi He, Tong Xiao, Chunliang Zhang, Tongran Liu, and Jingbo Zhu. Rovrm: A robust visual reward model optimized via auxiliary textual preference data. In Toby Walsh, Julie Shah, and Zico Kolter (eds.), *AAAI-25, Sponsored by the Association for the Advancement of Artificial Intelligence, February 25 - March 4, 2025, Philadelphia, PA, USA*, pp. 25336–25344. AAAI Press, 2025b. URL https://doi.org/10.1609/aaai.v39i24.34721.

Chenglong Wang, Yifu Huo, Yang Gan, Qiaozhi He, Qi Meng, Bei Li, Yan Wang, Junfu Liu, Tianhua Zhou, Jingbo Zhu, et al. Msrl: Scaling generative multimodal reward modeling via multi-stage reinforcement learning, 2026a. URL https://arxiv.org/abs/2603.25108.

Chenglong Wang, Yifu Huo, Yang Gan, Yongyu Mu, Qiaozhi He, Murun Yang, Bei Li, Chunliang Zhang, Tongran Liu, Anxiang Ma, Zhengtao Yu, Jingbo Zhu, and Tong Xiao. Probing preference representations: A multi-dimensional evaluation and analysis method for reward models. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 33404–33412. AAAI Press, 2026b. URL https://doi.org/10.1609/aaai.v40i39.40627.

Chenglong Wang, Yongyu Mu, Hang Zhou, Yifu Huo, Ziming Zhu, Jiali Zeng, Murun Yang, Bei Li, Xiaoyang Hao, Chunliang Zhang, Fandong Meng, Jingbo Zhu, and Tong Xiao. Gram-r^2: Self-training generative foundation reward models for reward reasoning. In Sven Koenig, Chad Jenkins, and Matthew E. Taylor (eds.), *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pp. 33395–33403. AAAI Press, 2026c. URL https://doi.org/10.1609/aaai.v40i39.40626.

Chenglong Wang, Ziming Zhu, Yifu Huo, Bei Li, Qiaozhi He, Yan Ding, Xiaoyang Hao, Yuxin Gao, Tianhua Zhou, Xiaojia Chang, Tongran Liu, and Jingbo Zhu. Rrc: Unlocking generative reward models in llm reinforcement learning via ranking-based reward construction, 2026d. URL https://arxiv.org/abs/2608.06310.

Daoyu Wang, Qingchuan Li, Mingyue Cheng, Jie Ouyang, Shuo Yu, Qi Liu, and Enhong Chen. Steppo: Step-aligned policy optimization for agentic reinforcement learning, 2026e. URL https://arxiv.org/abs/2604.18401.

Fusheng Wang, Jianhao Yan, Fandong Meng, and Jie Zhou. Selective knowledge distillation for neural machine translation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 6456–6466, Online, 2021. Association for Computational Linguistics. URL https://aclanthology.org/2021.acl-long.504/.

Hanlin Wang, Jian Wang, Chak Tou Leong, and Wenjie Li. STeCa: Step-level trajectory calibration for LLM agent learning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 11597–11614, Vienna, Austria, 2025c. Association for Computational Linguistics. ISBN 979-8-89176-256-5. URL https://aclanthology.org/2025.findings-acl.604/.

Jiongxiao Wang, Qiaojing Yan, Yawei Wang, Yijun Tian, Soumya Smruti Mishra, Zhichao Xu, Megha Gandhi, Panpan Xu, and Lin Lee Cheong. Reinforcement learning for self-improving agent with skill library. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1529–1550, San Diego, California, United States, 2026f. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.69/.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Lingpeng Kong, Qi Liu, Tianyu Liu, and Zhifang Sui. Large language models are not fair evaluators. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9440–9450, Bangkok, Thailand, 2024c. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.511/.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce LLMs step-by-step without human annotations. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 9426–9439, Bangkok, Thailand, 2024d. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.510/.

Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/7a8e7fd295aa04eac4b470ae27f8785c-Abstract-Conference.html.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. How far can camels go? exploring the state of instruction tuning on open resources. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023b. URL http://papers.nips.cc/paper_files/paper/2023/hash/ec6413875e4ab08d7bc4d8e225263398-Abstract-Datasets_and_Benchmarks.html.

Yu Wang, Ryuichi Takanobu, Zhiqi Liang, Yuzhen Mao, Yuanzhe Hu, Julian McAuley, and Xiaojian Wu. Mem-{\alpha}: Learning memory construction via reinforcement learning, 2025d. URL https://arxiv.org/abs/2509.25911.

Zhaoyang Wang, Canwen Xu, Boyi Liu, Yite Wang, Siwei Han, Zhewei Yao, Huaxiu Yao, and Yuxiong He. Agent world model: Infinity synthetic environments for agentic reinforcement learning, 2026g. URL https://arxiv.org/abs/2602.10090.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. Finetuned language models are zero-shot learners. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022a. URL https://openreview.net/forum?id=gEZrGCozdqR.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022b.

Zeqiu Wu, Yushi Hu, Weijia Shi, Nouha Dziri, Alane Suhr, Prithviraj Ammanabrolu, Noah A. Smith, Mari Ostendorf, and Hannaneh Hajishirzi. Fine-grained human feedback gives better rewards for language model training. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/b8c90b65739ae8417e61eadb521f63d5-Abstract-Conference.html.

Zhaofeng Wu, Ananth Balashankar, Yoon Kim, Jacob Eisenstein, and Ahmad Beirami. Reuse your rewards: Reward model transfer for zero-shot cross-lingual alignment. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1332–1353, Miami, Florida, USA, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-main.79/.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, Tao Ji, Tao Gui, Qi Zhang, and Xuanjing Huang. Agentprm: Process reward models for LLM agents via step-wise promise and progress. In Hakim Hacid, Yoelle Maarek, Francesco Bonchi, Ido Guy, and Emine Yilmaz (eds.), *Proceedings of the ACM Web Conference 2026, WWW 2026, Dubai, United Arab Emirates, originally scheduled for April 13-17, 2026, rescheduled for June 29 - July 3, 2026*, pp. 4184–4195. ACM, 2026. URL https://doi.org/10.1145/3774904.3792551.

Mengzhou Xia, Sadhika Malladi, Suchin Gururangan, Sanjeev Arora, and Danqi Chen. LESS: selecting influential data for targeted instruction tuning. In *Forty-first International Conference on Machine*

Learning, ICML 2024, Vienna, Austria, July 21-27, 2024. OpenReview.net, 2024. URL https://openreview.net/forum?id=PG5fV50maR.

Peng Xia, Jianwen Chen, Hanyang Wang, Jiaqi Liu, Kaide Zeng, Yu Wang, Siwei Han, Yiyang Zhou, Xujiang Zhao, Haifeng Chen, et al. Skillrl: Evolving agents via recursive skill-augmented reinforcement learning, 2026. URL https://arxiv.org/abs/2602.08234.

Teng Xiao, Yige Yuan, Huaisheng Zhu, Mingxiao Li, and Vasant G. Honavar. Cal-dpo: Calibrated direct preference optimization for language model alignment. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/cf8b2205e39f81726a8d828ecbe00ad0-Abstract-Conference.html.

Tong Xiao and Jingbo Zhu. Introduction to transformers: an nlp perspective, 2023. URL https://arxiv.org/abs/2311.17633.

Tong Xiao and Jingbo Zhu. Foundations of large language models, 2025. URL https://arxiv.org/abs/2501.09223.

Tong Xiao, Junhao Ruan, Bei Li, Zhengtao Yu, Min Zhang, and Jingbo Zhu. Ordinary differential equations in vision and language. 2026.

Tian Xie, Zitian Gao, Qingnan Ren, Haoming Luo, Yuqian Hong, Bryan Dai, Joey Zhou, Kai Qiu, Zhirong Wu, and Chong Luo. Logic-rl: Unleashing llm reasoning with rule-based reinforcement learning, 2025. URL https://arxiv.org/abs/2502.14768.

Haoran Xu, Amr Sharaf, Yunmo Chen, Weiting Tan, Lingfeng Shen, Benjamin Van Durme, Kenton Murray, and Young Jin Kim. Contrastive preference optimization: Pushing the boundaries of LLM performance in machine translation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=51iwkioZpn.

Jin Xu, Zhifang Guo, Jinzheng He, Hangrui Hu, Ting He, Shuai Bai, Keqin Chen, Jialin Wang, Yang Fan, Kai Dang, et al. Qwen2. 5-omni technical report, 2025a. URL https://arxiv.org/abs/2503.20215.

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tao Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable llm post-training, 2026. URL https://arxiv.org/abs/2602.01511.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025b. URL http://papers.nips.cc/paper_files/paper/2025/hash/19909c36f51abc4856b4560aff3d36d6-Abstract-Conference.html.

Zeyue Xue, Jie Wu, Yu Gao, Fangyuan Kong, Lingting Zhu, Mengzhao Chen, Zhiheng Liu, Wei Liu, Qiushan Guo, Weilin Huang, et al. Dancegrpo: Unleashing grpo on visual generation, 2025. URL https://arxiv.org/abs/2505.07818.

Sikuan Yan, Xiufeng Yang, Zuchao Huang, Ercong Nie, Zifeng Ding, Zonggen Li, Xiaowen Ma, Jinhe Bi, Kristian Kersting, Jeff Z. Pan, Hinrich Schuetze, Volker Tresp, and Yunpu Ma. Memory-r1: Enhancing large language model agents to manage and utilize memories via reinforcement learning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Proceedings of the 64th Annual Meeting of*

the Association for Computational Linguistics (Volume 1: Long Papers), pp. 12805–12825, San Diego, California, United States, 2026. Association for Computational Linguistics. ISBN 979-8-89176-390-6. URL https://aclanthology.org/2026.acl-long.583/.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM computing surveys*, 56(4):1–39, 2023.

Rui Yang, Ruomeng Ding, Yong Lin, Huan Zhang, and Tong Zhang. Regularizing hidden states enables learning generalizable reward model for llms. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/71f7154547c748c8041505521ca433ab-Abstract-Conference.html.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023a. URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023b. URL https://openreview.net/pdf?id=WE_vluYUL-X.

Da Yin, Faeze Brahman, Abhilasha Ravichander, Khyathi Chandu, Kai-Wei Chang, Yejin Choi, and Bill Yuchen Lin. Agent lumos: Unified and modular training for open-source language agents. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12380–12403, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.acl-long.670/.

Fangxu Yu, Tao Feng, Dehai Min, Zinan Lin, Weijia Xu, Michael Xu, Philip S. Yu, Ge Liu, and Tianyi Zhou. Reinforcement learning with evolving rubrics as rewards for audio reasoning, 2026a. URL https://arxiv.org/abs/2608.02831.

Jianxiang Yu, Jiapeng Zhu, Bochen Lin, Qier Cui, Zichen Ding, and Xiang Li. Skill is not one-size-fits-all: Model-aware skill alignment for llm agents, 2026b. URL https://arxiv.org/abs/2605.30723.

Tianyu Yu, Yuan Yao, Haoye Zhang, Taiwen He, Yifeng Han, Ganqu Cui, Jinyi Hu, Zhiyuan Liu, Hai-Tao Zheng, and Maosong Sun. RLHF-V: towards trustworthy mllms via behavior alignment from fine-grained correctional human feedback. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2024, Seattle, WA, USA, June 16-22, 2024*, pp. 13807–13816. IEEE, 2024a. URL https://doi.org/10.1109/CVPR52733.2024.01310.

Tianyu Yu, Haoye Zhang, Yuan Yao, Yunkai Dang, Da Chen, Xiaoman Lu, Ganqu Cui, Taiwen He, Zhiyuan Liu, Tat-Seng Chua, et al. Rlaif-v: Aligning mllms through open-source ai feedback for super gpt-4v trustworthiness, 2024b. URL https://arxiv.org/abs/2405.17220.

Siyu Yuan, Zehui Chen, Zhiheng Xi, Junjie Ye, Zhengyin Du, and Jiecao Chen. Agent-r: Training language model agents to reflect via iterative self-training, 2025. URL https://arxiv.org/abs/2501.11425.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024. URL https://openreview.net/forum?id=0NphYCmgua.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. Scaling relationship on learning mathematical reasoning with large language models, 2023. URL https://arxiv.org/abs/2308.01825.

Yuhang Zang, Xiaoyi Dong, Pan Zhang, Yuhang Cao, Ziyu Liu, Shengyuan Ding, Shenxi Wu, Yubo Ma, Haodong Duan, Wenwei Zhang, Kai Chen, Dahua Lin, and Jiaqi Wang. InternLM-XComposer2.5-reward: A simple yet effective multi-modal reward model. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6547–6563, Vienna, Austria, 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. URL https://aclanthology.org/2025.findings-acl.340/.

Aohan Zeng, Mingdao Liu, Rui Lu, Bowen Wang, Xiao Liu, Yuxiao Dong, and Jie Tang. AgentTuning: Enabling generalized agent abilities for LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3053–3077, Bangkok, Thailand, 2024a. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.181/.

Yongcheng Zeng, Guoqing Liu, Weiyu Ma, Ning Yang, Haifeng Zhang, and Jun Wang. Token-level direct preference optimization. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024b. URL https://openreview.net/forum?id=1RZKuvqYCR.

Zhiyuan Zeng, Qinyuan Cheng, Zhangyue Yin, Yunhua Zhou, and Xipeng Qiu. Revisiting the test-time scaling of o1-like models: Do they truly possess test-time scaling capabilities? In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4651–4665, Vienna, Austria, 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. URL https://aclanthology.org/2025.acl-long.232/.

Duzhen Zhang, Yahan Yu, Jiahua Dong, Chenxing Li, Dan Su, Chenhui Chu, and Dong Yu. MM-LLMs: Recent advances in MultiModal large language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 12401–12430, Bangkok, Thailand, 2024a. Association for Computational Linguistics. URL https://aclanthology.org/2024.findings-acl.738/.

Jianguo Zhang, Tian Lan, Rithesh Murthy, Zhiwei Liu, Weiran Yao, Ming Zhu, Juntao Tan, Thai Hoang, Zuxin Liu, Liangwei Yang, et al. Agentohana: Design unified data and training pipeline for effective agent learning, 2024b. URL https://arxiv.org/abs/2402.15506.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a. URL https://openreview.net/forum?id=Ccwp4tFEtE.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025b. URL https://openreview.net/forum?id=Ccwp4tFEtE.

Yilong Zhao, Chien-Yu Lin, Kan Zhu, Zihao Ye, Lequn Chen, Size Zheng, Luis Ceze, Arvind Krishnamurthy, Tianqi Chen, and Baris Kasikci. Atom: Low-bit quantization for efficient and accurate LLM serving. In Phillip B. Gibbons, Gennady Pekhimenko, and Christopher De Sa (eds.), *Proceedings of the Seventh Annual Conference on Machine Learning and Systems, MLSys 2024, Santa Clara, CA, USA, May 13-16, 2024*. mlsys.org, 2024. URL https://proceedings.mlsys.org/paper_files/paper/2024/hash/5edb57c05c81d04beb716ef1d542fe9e-Abstract-Conference.html.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. LIMA: less is more for alignment. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (eds.), *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/ac662d74829e4407ce1d126477f4a03a-Abstract-Conference.html.

Enyu Zhou, Guodong Zheng, Binghai Wang, Zhiheng Xi, Shihan Dou, Rong Bao, Wei Shen, Limao Xiong, Jessica Fan, Yurong Mou, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. RMB: comprehensively benchmarking reward models in LLM alignment. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=kmgrlG9TR0.

Hang Zhou, Chenglong Wang, Yimin Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. Prior constraints-based reward model training for aligning large language models. In Sun Maosong, Liang Jiye, Han Xianpei, Liu Zhiyuan, and He Yulan (eds.), *Proceedings of the 23rd Chinese National Conference on Computational Linguistics (Volume 1: Main Conference)*, pp. 1395–1407, Taiyuan, China, 2024. Chinese Information Processing Society of China. URL https://aclanthology.org/2024.ccl-1.107/.

Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (eds.), *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/8977ecbb8cb82d77fb091c7a7f186163-Abstract.html.