

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 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.

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.

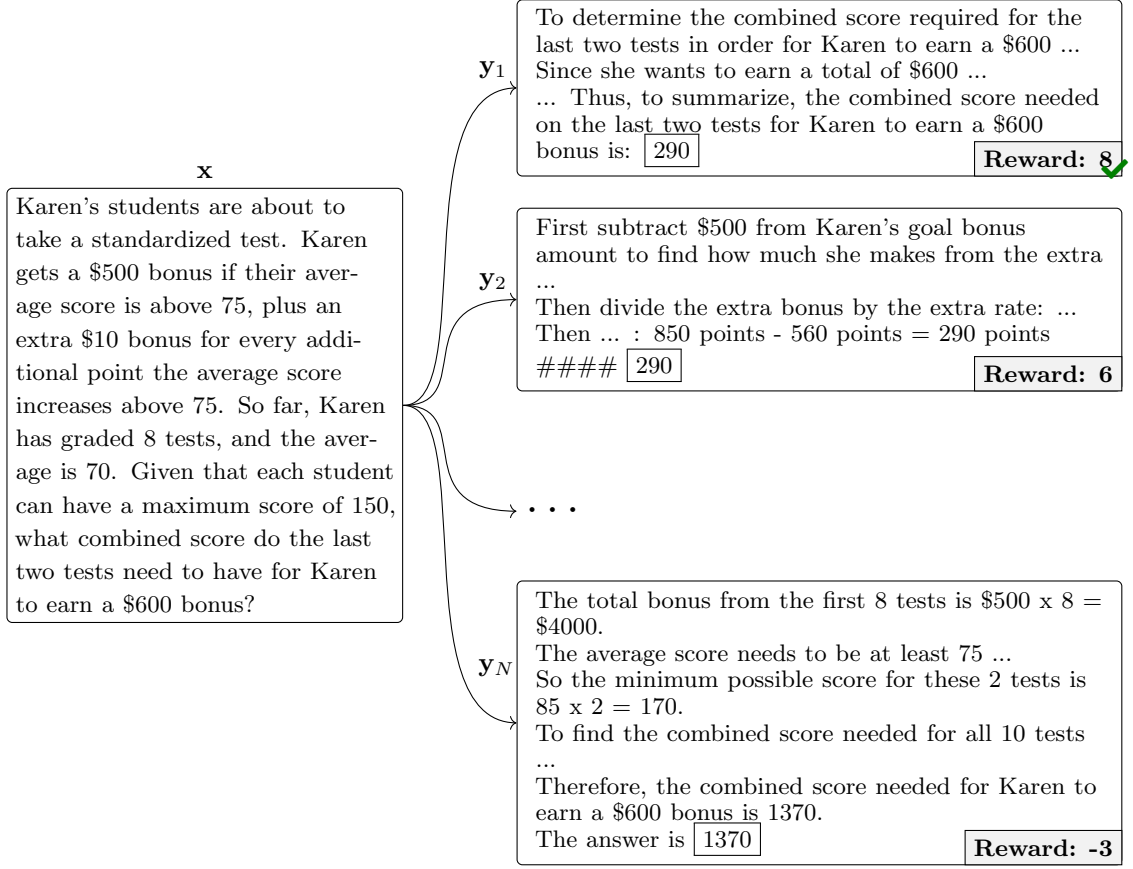


Figure 1: Best-of-n sampling. Given an input, multiple candidate outputs are sampled, and a reward model is used to select the best output. A majority vote can also determine the final output in scenarios without a reward model. For example, in this figure, most of the outputs suggest an answer of 290. Therefore, we can consider any one of the outputs resulting in 290 to be the final output.

As illustrated in Figure 1, in the BoN sampling, we first sample N different outputs $\{\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N\}$ for the input $\mathbf{x}$:

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

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:

$$\mathbf{y}_{\text{best}} = \arg \max_{\mathbf{y}_i \in \{\mathbf{y}_1, \dots, \mathbf{y}_N\}} R_{\phi}(\mathbf{x}, \mathbf{y}_i) \quad (2)$$

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 path¹. 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., 2024). 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., 2024b).

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 ?? . Here, consider a reasoning path including n_s steps, represented as $\mathbf{y} = \{\bar{\mathbf{y}}_1, \dots, \bar{\mathbf{y}}_{n_s}\}$. 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

$$R_\phi(\mathbf{x}, \bar{\mathbf{y}}_{\leq k}) = \log \Pr_\phi(\text{correct} | \mathbf{x}, \bar{\mathbf{y}}_{\leq k}) \quad (3)$$

¹We 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).

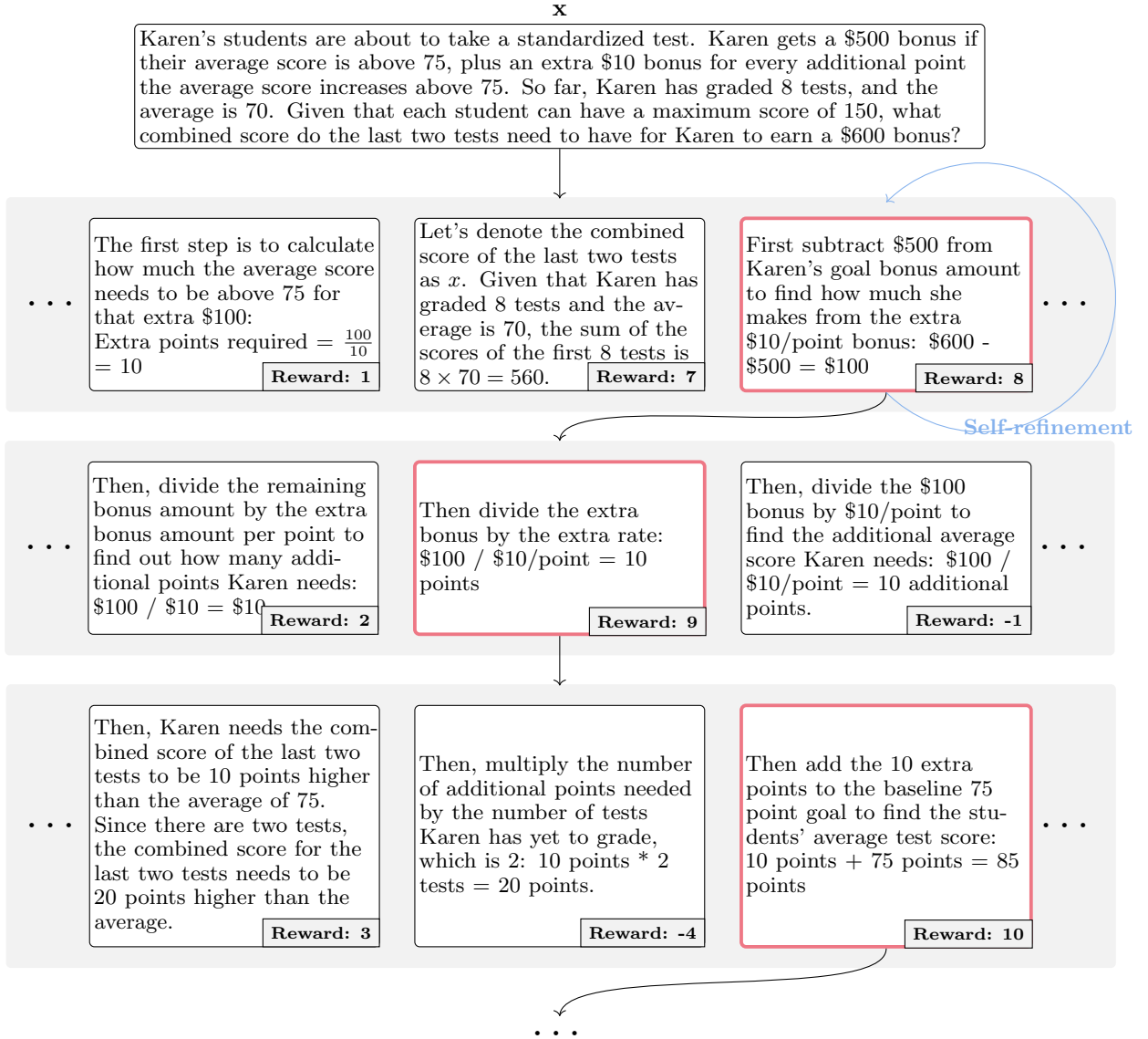


Figure 2: Step-by-step verification with greedy search. This process can be readily extended to beam search by retaining the top steps at each step from all candidate paths for further scaling up test-time computation. As the blue line indicates, the reasoning steps can be refined using verification feedback, such as rewards, to enhance accuracy.

where $\Pr_{\phi}(\text{correct}|\mathbf{x}, \bar{\mathbf{y}}_{\leq k})$ 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 ??, 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., 2025). 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 2, 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., 2023).

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 3, 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)². More specifically, at each step k , among the sampled candidate reasoning steps, we aim to select the one that maximizes the UCT objective:

$$\bar{\mathbf{y}}_k^* = \arg \max_{\bar{\mathbf{y}}_k} (\bar{R}(\bar{\mathbf{y}}_k) + c \sqrt{\frac{\ln N_p}{N_{\bar{\mathbf{y}}_k}}}) \quad (4)$$

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.

- **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. (4), reflecting how 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 two-phase approach to train LLMs in reasoning may lead to significant knowledge forgetting as the model

²When 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).

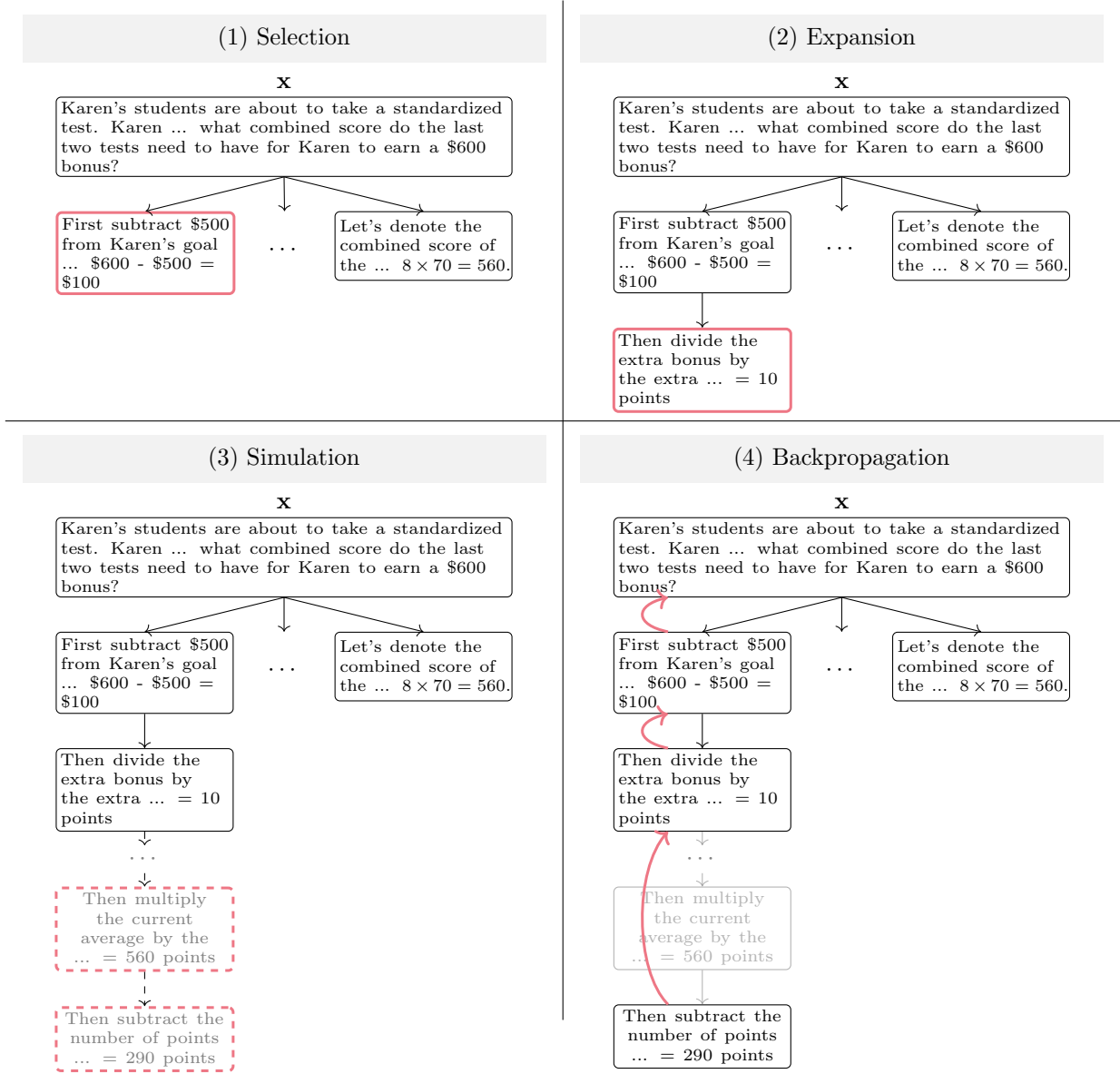


Figure 3: Illustration of step-by-step verification with MCTS. The process consists of four phases: (1) Selection, where the algorithm navigates the decision tree to select the most promising node based on strategies like UCT; (2) Expansion, where new potential nodes are added to the tree for further 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.

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

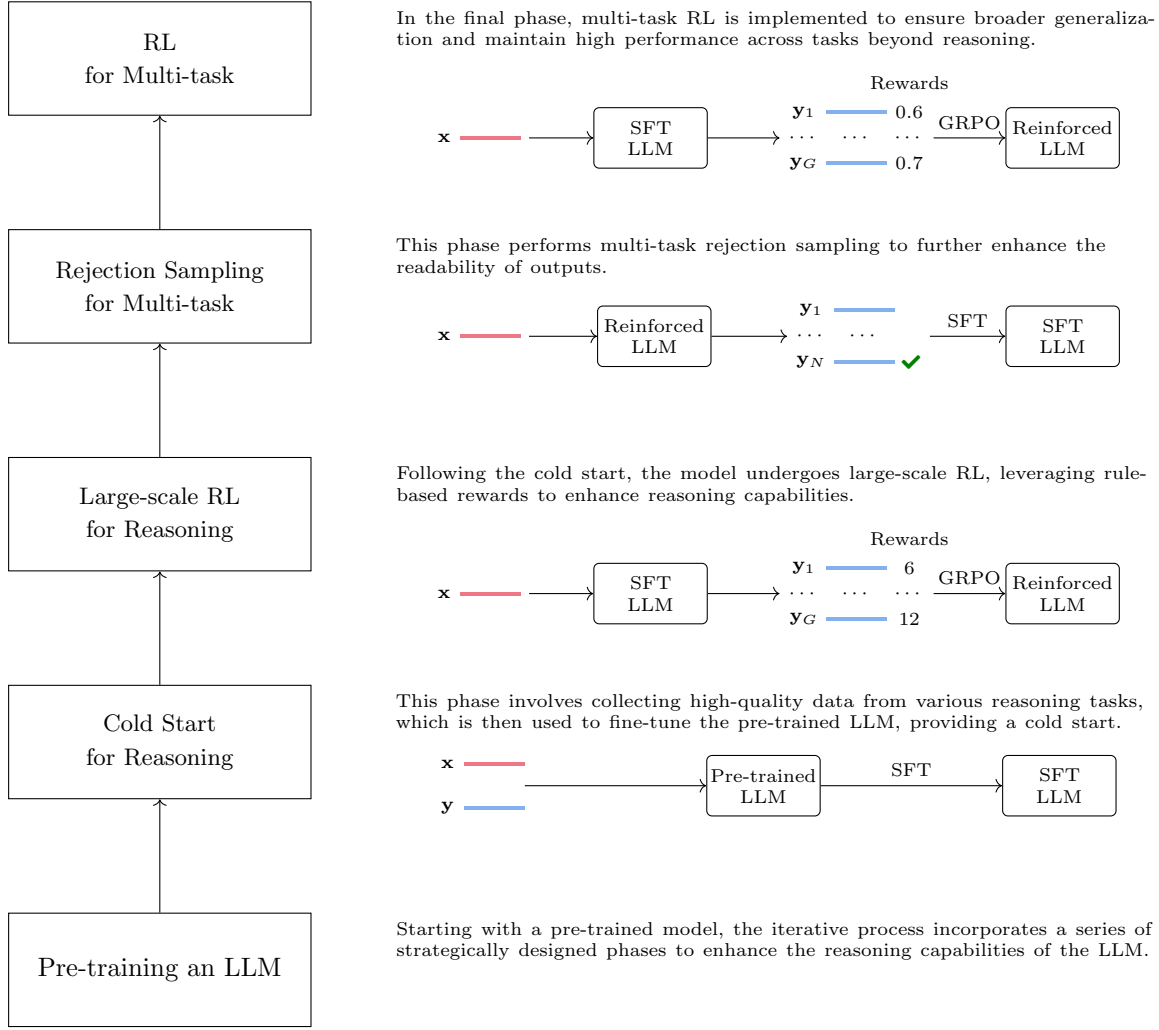


Figure 4: Illustration of iterative RL in DeepSeek-R1 (Guo et al., 2025). This method maintains multi-phase RL to develop a robust reasoning LLM with a GRPO algorithm. Initially, a pre-trained LLM is fine-tuned using a small set of high-quality labeled reasoning data with SFT as a cold start. Then, large-scale 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.

that is able to generate clearer and more comprehensible reasoning paths. Figure 4 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 ??, 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 ??, 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., 2025). 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 ??, 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 model³. 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 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 5. Given an input $\mathbf{x}$, we first sample an output $\mathbf{y} = \{y_1, \dots, y_n\}$ from the current student model $\text{Pr}_\theta(\cdot|\mathbf{x})$. 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 Pr_{tea} and obtain its probability distribution over the next token. The student model is

³In standard knowledge distillation settings, we typically refer to the “smaller” model as the *student model* and the “larger” model as the *teacher model*.

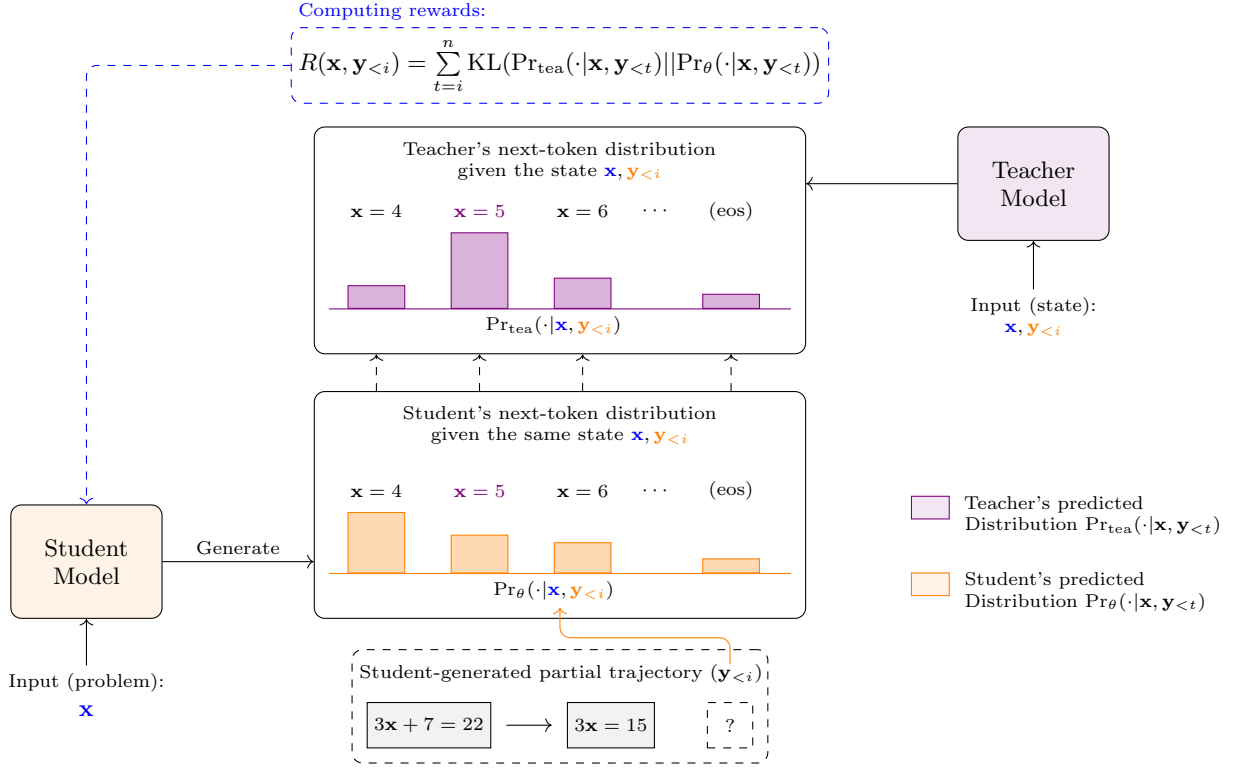


Figure 5: Illustration of OPD (Agarwal et al., 2024). The student model generates trajectories from its current policy, while the teacher model provides next-token distributional supervision at the states visited by the student. The divergence between the teacher and student distributions is used to construct dense token-level rewards for optimizing the student model.

optimized to match this teacher distribution. This OPD objective can be defined as:

$$\mathcal{L}_{\text{opd}}(\theta) = \mathbb{E}_{\mathbf{x} \sim \mathcal{S}_x, \mathbf{y} \sim \text{Pr}_{\theta}(\cdot | \mathbf{x})} \left[\frac{1}{n} \sum_{i=1}^n \text{KL}(\text{Pr}_{\text{tea}}(\cdot | \mathbf{x}, \mathbf{y}_{<i}) || \text{Pr}_{\theta}(\cdot | \mathbf{x}, \mathbf{y}_{<i})) \right] \quad (5)$$

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.

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>.
- Kalyanmoy Deb, Karthik Sindhya, and Jussi Hakanen. Multi-objective optimization. In *Decision sciences*, pp. 161–200. CRC Press, 2016.
- 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 Gorbатовski, 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>.
- 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>.
- 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, 2024. URL <https://openreview.net/forum?id=v8LOpN6EOi>.
- 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.
- 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/>.
- 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.
- 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>.

- 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.
- 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, 2025. URL <https://arxiv.org/abs/2501.12599>.
- 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>.
- 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, 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/>.
- 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, 2024b. Association for Computational Linguistics. URL <https://aclanthology.org/2024.acl-long.510/>.
- 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*, 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/271db9922b8d1f4dd7aaef84ed5ac703-Abstract-Conference.html.
- 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=ONphYCMgua>.
- 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>.
- 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/>.
- 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, 2025. URL <https://openreview.net/forum?id=Ccwp4tFEtE>.