A Survey on Complex Reasoning of Large Language Models through the Lens of Self-Evolution

Tao He^{1,†} **Hao Li**^{1,†} **Jingchang Chen**^{1} **Runxuan Liu**^{1} **Yixin Cao**^{2} **Lizi Liao**^{3}
Zihao Zheng^{1} **Zheng Chu**^{1} **Jiafeng Liang**^{1} **Ming Liu**^{1*} **Bing Qin**^{1}
^{1}Harbin Institute of Technology ^{2}Fudan University ^{3}Singapore Management University
{the,haoli,jcchen,rxliu,zhzheng,zchu,jfliang,mliu,qinb}@ir.hit.edu.cn
yxcao@fudan.edu.cn
lzliao@smu.edu.sg

Abstract

The release of OpenAI's O1 and subsequent challengers (i.e., DeepSeek R1) has significantly advanced research on complex reasoning in Large Language Models (LLMs), capturing the attention of both academia and industry. These developments have spurred efforts to replicate and build upon their achievements. To further propel research in this domain, this paper systematically analyzes existing techniques from the angle of self-evolution. Our approach is organized into three interconnected components: data evolution, model evolution, and self-evolution. The data evolution section examines efforts to enhance reasoning training data, including advancements in task evolution and inference-time computing to improve the quality of Chain-of-Thought (CoT) reasoning. The model evolution section highlights methods for optimizing model modules through training to augment the system' complex reasoning capabilities. Finally, the self-evolution section explores evolutionary strategies and patterns. We further discuss the scaling law of self-evolution and analyze representative O1-like works from the perspective of self-evolution. By systematically investigating relevant research, we summarize advanced methods and provide a forward-looking perspective on potential future directions. This paper aims to inspire further research within the LLM complex reasoning community and to foster deeper exploration into the advancement of LLMs' reasoning abilities.

1 Introduction

The rapid advancements in large language models (LLMs) over the past few years have been nothing short of extraordinary. Not only have they surpassed expectations in domains such as reading comprehension, story generation, and conversational abilities, but they have also demonstrated remarkable performance in tasks demanding intricate logical reasoning, including code generation and mathematical problem-solving. A pivotal moment in LLM research occurred in the latter half of last year with the release of OpenAI's O1 [OpenAI, 2024a], which marked a significant milestone in the study of complex reasoning. The O1 series of models can generate extensive reasoning processes, adeptly deconstructing problems and, when faced with challenges, autonomously clarifying, reflecting, and correcting potential errors, as well as exploring alternative solutions—emulating the meticulous, reflective reasoning processes characteristic of human thought [OpenAI, 2024b].

Both industry and academia have focused on reproducing O1, resulting in a wave of technical reports. In the industry, a series of similar products have emerged, such as DeepSeek R1 [DeepSeek-AI et al., 2025] (Shorted as R1), Kimi k1.5 [Team et al., 2025], and QwQ [Team, 2024b], each

[*] Corresponding Author: Ming Liu.

Preprint. Under review.

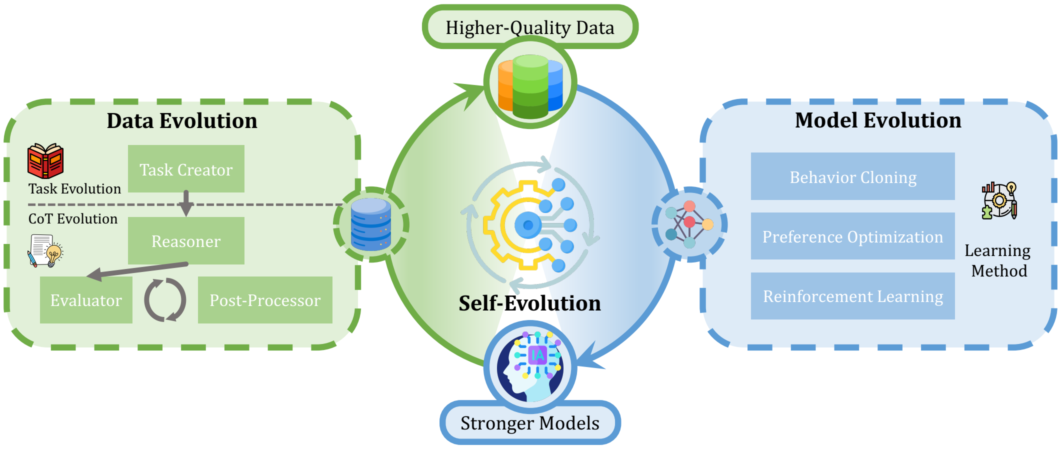


Figure 1: A conceptual framework for self-evolving complex reasoning capacities in LLMs. We identify three components in a complete self-evolution framework: data evolution, model evolution, and evolutionary strategies and patterns.

releasing their models or technical reports. These products not only match or even surpass O1 but are also commendable for their open-source contributions. Moreover, advanced technologies like the Scaling Reinforcement Learning (RL) highlighted in these technical reports have further expanded the directions for studying O1-like works. This advancement has significantly propelled the research progress of complex reasoning for LLMs and provided valuable insights for academic research within the community. In academia, several replication studies have been conducted from different perspectives. For example, O1 Journey [Qin et al., 2024, Huang et al., 2024] extensively discusses chain-of-thought formatting and distillation but offers limited insights into the continuous optimization methods. Meanwhile, OpenR [Wang et al., 2024e], O1-Coder [Zhang et al., 2024j], and other works primarily approach O1 research through the lens of RL, but they overlook discussions of reflective and corrective reasoning operations. On the other hand, Slow Thinking series works [Jiang et al., 2024a, Min et al., 2024] focus on inference-time computation, attempting to enhance reasoning performance via tree search techniques. Besides, rStar-Math [Guan et al., 2025] achieves close performance to O1 by jointly training the reasoner and Process Reward Model (PRM) using the self-evolution framework, underscoring the potential of iterative optimization in advancing reasoning capabilities.

Despite gaining valuable insights from various technical reports, we have observed the shortcoming that each report has a limited scope, focusing on its own technical approach, leading to a fragmented and complex landscape of methodologies. Thus, a systematic and higher-level review of these methodologies is urgently needed. The O1 blog [OpenAI, 2024a] and system card [OpenAI, 2024b] hint that O1 employs RL and inference-time compute. This naturally leads us to draw parallels with another prominent AI system, AlphaGo Zero [Silver et al., 2017]. AlphaGo Zero achieved self-evolution through self-play and iterative learning via Monte Carlo Tree Search (MCTS) and policy models [Silver et al., 2017]. This process enhanced its performance without human intervention, inspiring a similar technique could elevate complex reasoning abilities beyond human-level performance. In this analogy, training the policy model corresponds to optimization for reasoning, while MCTS search is exactly the inference-time compute. Self-evolution cycles these two stages to enable autonomous evolution of reasoning capabilities. Moreover, the scarcity of high-quality data underscores the urgent need for automated data synthesis frameworks [Sutskever, 2024, Wang et al., 2024f], a challenge that is even more pronounced in reasoning tasks due to their higher demands for logical rigor. Self-evolution could not only leverage synthetic data to enhance system capabilities without human intervention but also leverage improved systems to synthesize even higher-quality data, creating a virtuous cycle of advancement.

In light of these considerations, this paper hopes to provide a comprehensive review of LLM complex reasoning from the lens of self-evolution. Self-evolution, also known as self-improvement, for LLM complex reasoning requires to autonomously synthesize training data and continuously improve reasoning capabilities in a closed-loop reasoning system [Tao et al., 2024, Hu et al., 2024]. Expect iteration [Polu et al., 2022, Zhao et al., 2024b] is known as a classical self-evolution paradigm. Its

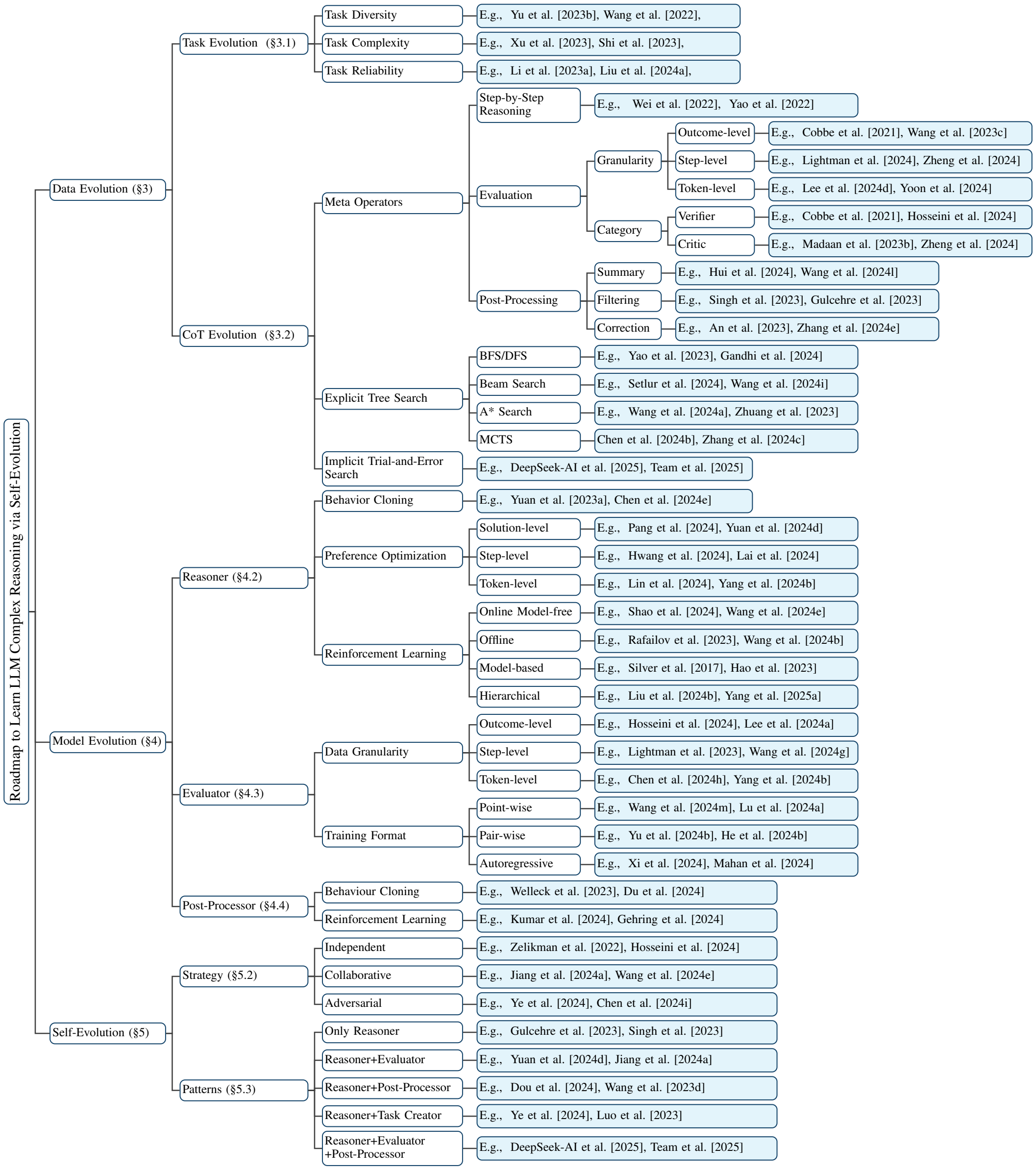

Roadmap to Learn LLM Complex Reasoning via Self-Evolution
Data Evolution (§3)
Task Evolution (§3.1)
Task Diversity
E.g., Yu et al. [2023b], Wang et al. [2022],
Task Complexity
E.g., Xu et al. [2023], Shi et al. [2023],
Task Reliability
E.g., Li et al. [2023a], Liu et al. [2024a],
CoT Evolution (§3.2)
Meta Operators
Step-by-Step Reasoning
E.g., Wei et al. [2022], Yao et al. [2022]
Evaluation
Granularity
Outcome-level
E.g., Cobbe et al. [2021], Wang et al. [2023c]
Step-level
E.g., Lightman et al. [2024], Zheng et al. [2024]
Token-level
E.g., Lee et al. [2024d], Yoon et al. [2024]
Category
Verifier
E.g., Cobbe et al. [2021], Hosseini et al. [2024]
Critic
E.g., Madaan et al. [2023b], Zheng et al. [2024]
Post-Processing
Summary
E.g., Hui et al. [2024], Wang et al. [2024l]
Filtering
E.g., Singh et al. [2023], Gulcehre et al. [2023]
Correction
E.g., An et al. [2023], Zhang et al. [2024e]
Explicit Tree Search
BFS/DFS
E.g., Yao et al. [2023], Gandhi et al. [2024]
Beam Search
E.g., Setlur et al. [2024], Wang et al. [2024i]
A* Search
E.g., Wang et al. [2024a], Zhuang et al. [2023]
MCTS
Chen et al. [2024b], Zhang et al. [2024c]
Implicit Trial-and-Error Search
E.g., DeepSeek-AI et al. [2025], Team et al. [2025]
Model Evolution (§4)
Reasoner (§4.2)
Behavior Cloning
E.g., Yuan et al. [2023a], Chen et al. [2024e]
Preference Optimization
Solution-level
E.g., Pang et al. [2024], Yuan et al. [2024d]
Step-level
E.g., Hwang et al. [2024], Lai et al. [2024]
Token-level
E.g., Lin et al. [2024], Yang et al. [2024b]
Reinforcement Learning
Online Model-free
E.g., Shao et al. [2024], Wang et al. [2024e]
Offline
E.g., Rafailov et al. [2023], Wang et al. [2024b]
Model-based
E.g., Silver et al. [2017], Hao et al. [2023]
Hierarchical
E.g., Liu et al. [2024b], Yang et al. [2025a]
Evaluator (§4.3)
Data Granularity
Outcome-level
E.g., Hosseini et al. [2024], Lee et al. [2024a]
Step-level
E.g., Lightman et al. [2023], Wang et al. [2024g]
Token-level
E.g., Chen et al. [2024h], Yang et al. [2024b]
Training Format
Point-wise
E.g., Wang et al. [2024m], Lu et al. [2024a]
Pair-wise
E.g., Yu et al. [2024b], He et al. [2024b]
Autoregressive
E.g., Xi et al. [2024], Mahan et al. [2024]
Post-Processor (§4.4)
Behaviour Cloning
E.g., Welleck et al. [2023], Du et al. [2024]
Reinforcement Learning
E.g., Kumar et al. [2024], Gehring et al. [2024]
Self-Evolution (§5)
Strategy (§5.2)
Independent
E.g., Zelikman et al. [2022], Hosseini et al. [2024]
Collaborative
E.g., Jiang et al. [2024a], Wang et al. [2024e]
Adversarial
E.g., Ye et al. [2024], Chen et al. [2024i]
Patterns (§5.3)
Only Reasoner
E.g., Gulcehre et al. [2023], Singh et al. [2023]
Reasoner+Evaluator
E.g., Yuan et al. [2024d], Jiang et al. [2024a]
Reasoner+Post-Processor
E.g., Dou et al. [2024], Wang et al. [2023d]
Reasoner+Task Creator
E.g., Ye et al. [2024], Luo et al. [2023]
Reasoner+Evaluator +Post-Processor
E.g., DeepSeek-AI et al. [2025], Team et al. [2025]

Figure 2: Taxonomy of Advanced Methods, including data evolution, model evolution, and self-evolution.

main idea involves that the model first generates reasoning trajectories, filters them based on golden answers to obtain correct solutions, and finally fine-tunes the model using these solutions to improve its capabilities. This process is repeated iteratively until the model converges. Besides, the online RL framework also embodies self-evolution thoughts. The agent initially explores and generates trajectories, which are then used for optimizing itself, enabling the agent to discover higher-quality trajectories in subsequent learning cycles.

The structure of this survey, shown in Figure 1, consists of three sections: **data evolution**, **model evolution**, and **self-evolution**. Data evolution explores to synthesize high-quality data and consists of two stages: (1) task evolution generates tasks that the reasoning system cannot yet handle effectively, and (2) CoT evolution scales LLM's performance boundaries through scaling inference-time compute [Snell et al., 2024] and generates better solutions via Chain-of-Thought (CoT) [Wei et al., 2022] reasoning. However, the improved performance may stem from heuristic search algorithms rather than the model's inherent capabilities, as evidenced by its inability to consistently generate high-quality CoTs. Model evolution addresses them by promoting system modules. On the one hand, models are specifically trained on tasks they struggled with. On the other hand, they selectively learn from the collected data to genuinely expand their capability boundaries. The first two evolution represent an exploratory and divergent effort to investigate promising technologies and challenges to implementing data and model evolution. This lays the technical foundation of data synthesis strategies and optimization methods for self-evolution. In the third section, we focus on the self-evolution framework for the reasoning system. By iteratively conducting data evolution and model evolution, the reasoning system achieves self-evolution: data evolution generates more targeted and higher-quality data based on the current model, and model evolution further strengthens the model using collected data, providing a stronger foundation for the next round of data evolution.

Our contributions can be summarized as follows: (1) **Comprehensive Survey**: This is the first comprehensive survey dedicated for the self-evolution of LLM reasoning; (2) **Taxonomy**: We introduce a meticulous taxonomy in Figure 2. (3) **Theory**: We collect related underlying theory and discuss the scaling law of self-evolution; (4) **Frontier and Future**: We analyze the latest open-source studies within the self-evolution framework and shed light on future research.

2 Preliminaries

2.1 Background

This survey focuses on complex reasoning tasks facilitated by LLMs. Specifically, we focus on CoT reasoning, where the LLM generates a step-by-step reasoning process, i.e., CoT, before predicting the final answer.

To facilitate subsequent discussion, we formalize the task and its solution process as follows: given a task q, the LLM p_{LLM} first generates a step-by-step CoT y and then predicts the final answer z based on y. This process can be mathematically expressed as:

$$p_{LLM}(z|q) = p_{LLM}(z|q, y) \cdot p_{LLM}(y|q). \tag{1}$$

Given that y and z often appear sequentially in practice, we may occasionally use y to denote the solution, i.e., CoT. Sometimes, we may abuse y to represent encompassing both the CoT and the final answer.

2.2 Framework Elements

Drawing insights from existing research on reasoning, we first delineate the essential components that a closed-loop self-evolution reasoning framework should incorporate. Specifically, we identify four critical modules as follows:

Task Creator: A reasoning system necessitates tasks as input. The most straightforward implementation of a Task Creator involves sampling from a fixed task set. However, unlike single-round reasoning improvement, self-evolution requires continuous enhancement of reasoning capabilities through iterative optimization. Fixed task sets may result in rapid performance convergence Jiang et al. [2024a], as the system learns to identify "shortcuts" tailored to specific tasks, thereby diminishing model generalization. Consequently, generating diverse tasks is essential to mitigate this issue and promote self-evolution.

Reasoner: The Reasoner model serves as the core component of the system, tasked with receiving inputs from the Task Creator and generating solutions through step-by-step reasoning. In this study, the Reasoner is implemented using an LLM.

Evaluator: The Evaluator is responsible for assessing and verifying the reasoning process generated by the Reasoner. This auxiliary module serves several critical functions: during the training phase, it provides score-based feedback to fine-tune the Reasoner through methods such as rejection fine-tuning or RL; during the inference phase, it evaluates the reasoning process, thereby guiding the inference-time compute and subsequent post-processing steps.

Post-Processor: The Post-Processor processes the solutions generated by the Reasoner based on evaluation feedback from the Evaluator. The simplest operation is filtering out incorrect solutions directly; however, such a method risks data wastage and does not fully align with human approaches facing errors. The post-processing occurs in two stages: during generation, it can refine the partial CoT by correcting error steps or backtracking; after generation, it leverages the system's correction capabilities to refine the complete solutions.

It is important to note that these modules are logically distinct, not physically. Owing to the powerful instruction-following capabilities of LLMs, a single model can simultaneously fulfill multiple roles during implementation. In the following sections, we explore their significant roles in data evolution (working together to generate high-quality data), model evolution (optimizing each module), and self-evolution (joint evolution of modules).

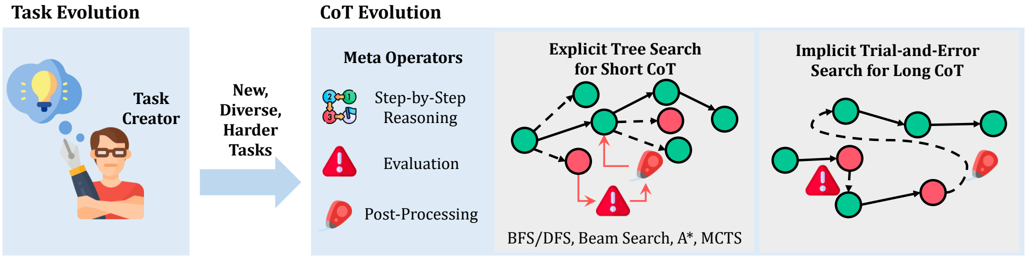

Task Evolution
Task Creator
New, Diverse, Harder Tasks
CoT Evolution
Meta Operators
Step-by-Step Reasoning
Evaluation
Post-Processing
Explicit Tree Search for Short CoT
BFS/DFS, Beam Search, A*, MCTS
Implicit Trial-and-Error Search for Long CoT

Figure 3: The pipeline of Data Evolution consists of Task Evolution and CoT Evolution. In CoT Evolution, we define three meta operators, which enable two search paradigms to generate higher-quality CoTs.

3 Data Evolution

As illustrated in Figure 3, the data evolution is divided into two distinct phases: task evolution and CoT evolution. Task evolution is centered on enhancing the quality of tasks by addressing key factors such as difficulty, diversity, and effectiveness. By refining these dimensions, the system avoids being confined to a limited range of tasks, thereby significantly enhancing its generalization capabilities. CoT evolution, commonly referred to as inference-time compute in most contexts, aims to improve the quality of the reasoning process during the inference phase. This improvement is primarily evident in the accuracy of reasoning, logical coherence, and the system's ability to autonomously evaluate and correct errors.

3.1 Task Evolution

Task evolution focuses on generating new tasks, enhancing the diversity and difficulty of the training data to bolster the model's reasoning and generalization capabilities. This approach is akin to students improving their skills by exercising various kinds of questions. Despite its critical role, we observe that current research on self-evolution for reasoning seldom explores this dimension. To address this gap and establish the foundation for a closed-loop system, we prioritize task evolution as our starting point. As a pivotal component of data synthesis, task evolution often involves the creation of new tasks, such as leveraging a more advanced LLM to formulate new challenges Li et al. [2024a]. In this section, we concentrate on enhancing task diversity, complexity, and reliability to robustly support task evolution.

Task Diversity To enhance task diversity, Haluptzok et al. [2022], Madaan et al. [2023a] prompt LLMs to modify data types and logical operations of reference problems, generating structurally similar but logically distinct tasks. And Yu et al. [2023b] use LLMs to rephrase reference problems to create new ones. However, this approach is inherently limited by its reliance on reference data, which restricts the generation of entirely novel tasks and curtails both diversity and creativity. To address this limitation, sampling data from high-variance distributions or incorporating diversity-focused prompts can serve as effective strategies. For instance, Liu et al. [2023] employ temperature sampling and diversity-focused prompts to enrich question generation, and Xu et al. [2023] explicitly instruct the LLMs to create rare, domain-specific questions. Besides, Self-Instruct [Wang et al., 2022] generates new task instructions by combining human-written and model-generated tasks, using specific prompt templates to guide generation.

Task Complexity Xu et al. [2023] propose several methods to generate more complex tasks based on example problems: 1) Adding Constraints: Introducing additional constraints or requirements to increase task difficulty, improving model flexibility and adaptability; 2) Deepening: Expanding the depth and breadth of queries within an example to enhance the model's reasoning ability; 3) Concretizing: Replacing general concepts in the problem with more specific ones, making instructions clearer and improving the accuracy and relevance of responses; 4) Increasing Reasoning Steps: Reformulating simple problems to require additional reasoning steps, thereby strengthening the model's logical thinking capacity; 5) Increasing Input Complexity: Modifying problem conditions to incorporate structured data or specific input formats (e.g., code, tables, XML), this transformation shifts problems from being directly computable to requiring additional data parsing or manipulation, thereby enhancing the model's robustness and generalizability. Additionally, Shi et al. [2023] increase reasoning difficulty by introducing irrelevant conditions, forcing the model to identify and focus on key conditions, while Mitra et al. [2024] reformulate problems into declarative statements by embedding the answer within the question and subsequently guide the generation of more complex problems through automated suggestions, such as the introduction of additional variables.

Task Reliability Automatically generating tasks risks producing unsolved tasks or incorrect answers. To address this, Li et al. [2023a] employ fine-tuned LLMs to score and select high-quality tasks. Similarly, Liu et al. [2024a], Xu et al. [2023] generate various tasks based on original questions and filter out inconsistent ones by validating the answers. Haluptzok et al. [2022], Liu et al. [2023] ensure quality by leveraging Python interpreters and predefined rules to verify correctness, such as checking for appropriate length or the presence of numerical content. Kreber and Hahn [2021] propose a GAN [Goodfellow et al., 2014] with a Transformer-based encoder for task synthesis, using random noise to generate symbolic tasks. The critic assesses the similarity between generated tasks and real data, providing feedback to refine the generator and enhance task reliability. Additionally, Wei et al. [2023], Lu et al. [2024b] explore reverse task generation by leveraging LLMs to derive problems from solutions. Specifically, Lu et al. [2024b] iteratively generate new answers from a mathematical reference solution, defines constraints and logical relations, and translates these answers into tasks, ensuring the reliability of the generated problems. Similarly, Wei et al. [2023] utilize high-quality open-source code to create programming tasks through LLM-based generation.

3.2 CoT Evolution

Before starting the reasoning process, it's important to conceptualize what an effective reasoning CoT should look like and what meta-operations it should consist of. A well-designed CoT format determines the upper bound of the system's reasoning capabilities. In this section, we begin by defining three meta-operators to construct more powerful CoTs, including **Step-by-Step Reasoning**, **Evaluation**, and **Post-Processing**.

We further review inference-time compute methods, typically implemented via search, to generate higher-quality CoTs. These methods are categorized into **explicit tree search** and **implicit trial-and-error search**. Early research mainly focused on explicit tree search. However, with the emergence of O1 and subsequent open-source efforts such as R1 [DeepSeek-AI et al., 2025], Kimi k1.5 [Team et al., 2025], and T1 [Hou et al., 2025], attention has gradually shifted toward trial-and-error search. Analyzing the CoT examples provided by O1 [OpenAI, 2024a] reveals that O1 can self-correct or backtrack to earlier steps upon detecting errors, recording the entire reasoning process within the CoT. This behavior mimics human-like long-time cognitive thinking before responding to a question. The O1 Journey [Qin et al., 2024] earlier discussed this, introducing the term Shortcut Learning [Geirhos et al., 2020] to describe the pursuit of CoTs that each step is correct and Journey Learning [Qin et al., 2024] to characterize the process of spontaneous verification, error detection, and correction during reasoning. Kimi k1.5 [Team et al., 2025] and Redstar [Xu et al., 2025] further explore this concept, referring to its outcome as **Long CoT**. In alignment with this terminology, we designate the results of Shortcut Learning as **Short CoT**.

3.2.1 Meta Operators

The potential of CoT reasoning has been widely explored. While vanilla CoT performs well on simple tasks, it performs insufficiently for more complex ones. An approach to enhance CoT reasoning is to design more sophisticated and efficient reasoning chains inspired by human cognition. Observations from O1-like systems [Qin et al., 2024, Zeng et al., 2024b] have sparked discussions on operations such as decomposition, step-by-step reasoning, self-evaluation, self-correction, and backtracking. Therefore, we summarize and categorize the three key meta-operations: **Step-by-step Reasoning**, **Evaluation**, and **Post-processing**.

Focusing on these three meta-operators and reviewing the modules outlined in the previous section(§2.2), we can establish the correlations between the CoT format and the reasoning system modules. The Reasoner generates the reasoning process through step-by-step decomposition, with search algorithms acting as the extension technique. Simultaneously, the Evaluator and Post-Processor manage the evaluation (reflection) and correction processes within the CoT, respectively. By integrating these three modules, we can effectively incorporate all three meta-operators into the CoT, ensuring a comprehensive and robust reasoning system.

Step-by-Step Reasoning

Step-by-step reasoning decomposes a problem into sequentially dependent steps, which requires robust planning capabilities, and then solves it incrementally through a chain-based reasoning process Chu et al. [2023]. Moreover, the decomposition process should be recursive, enabling the system to iteratively decompose complex sub-problems.

CoT Wei et al. [2022] represents a straightforward linear search approach, utilizing few-shot or zero-shot prompting to solve problems step-by-step. Plan-and-Solve [Wang et al., 2023b] adopts zero-shot prompting to guide the model in generating a plan within a single generation process, followed by chain-based reasoning grounded in the generated plan. Least-to-Most Prompting [Zhou et al., 2022] takes a two-phase approach: in the first phase, the problem is explicit decomposed into multiple sub-problems, and in the second phase, these sub-problems are solved sequentially. At each step, the outcomes of previous steps are appended to the context, prompting the model to solve the next sub-problem. In contrast to planning-based methods, Successive Prompting [Dua et al., 2022] employs an iterative decomposition process. In each iteration, a new sub-problem is proposed and solved within the current step. This two-step process is repeated until the entire problem is solved. ReACT [Yao et al., 2022] combines iterative reasoning with actions. At each step, the model generates an action based on its reasoning. The action may involve calling external tools, such as a calculator, or interacting with the environment. The model then uses feedback from these external tools or the environment to proceed to the next step until the final goal is reached. By incorporating actions, ReACT enables the model to interact with external systems, thereby enhancing the reasoning process of LLMs.

Evaluation

A robust reasoning system must possess self-evaluation capabilities, enabling it to evaluate its reasoning process both during and after task execution. During reasoning, the system should identify and terminate erroneous exploratory paths for post-processing. In heuristic searches, evaluation results further serve to guide the search. Upon completing the reasoning process, multiple candidate answers may be generated, necessitating thorough evaluation to assess and verify different solutions effectively. We systematically review existing research at three granularities: **outcome-level**, **step-level**, and **token-level**.

- **Outcome-level Evaluation** Early works primarily focused on outcome-level evaluation, where the evaluation is performed on the complete solution after reasoning [Cobbe et al., 2021, Wang et al., 2023c, Lee et al., 2024a]. The main differences in these methods lie in the form and purpose of the evaluation. During the training phase, when the correct answers are available, some works conduct straightforward correctness assessments of the solutions against the ground truth [Cobbe et al., 2021, Hosseini et al., 2024]. Beyond mere answer accuracy, R1 [DeepSeek-AI et al., 2025] and T1 [Hou et al., 2025] incorporate format-based outcome rewards to guide reasoning format learning. In the inference phase, Cobbe et al. [2021], Hosseini et al. [2024] utilize trained verifiers to score and rank candidate solutions, thereby selecting the optimal one. Moreover, some methods use LLMs to produce natural language feedback on solutions. For example, Madaan et al. [2023b], Zhang et al. [2024b] directly generate critiques, while Peng et al. [2023], Shinn et al. [2023], Gou et al. [2024] include both internal and external environmental information in their critique generation. Additionally, Ankner et al. [2024b], Yu et al. [2024b] merge natural language critiques with scoring mechanisms to improve the reliability and interpretability of the evaluations. Some studies also adopt consistency-based evaluation frameworks. For example, Wang et al. [2023c] employ a voting system to ascertain the final answer from multiple solution candidates, while Jiang et al. [2024b], Weng et al. [2023] evaluate answer quality by ensuring the consistency between forward and backward reasoning processes.

- **Step-level Evaluation** Outcome-level evaluation, while straightforward to implement, exhibits limited applicability in practice, where more granular evaluation is often necessary.

Among these, step-level evaluation has emerged as a particularly prominent approach. This evaluation paradigm emphasizes the assessment of individual reasoning steps, as demonstrated in recent studies [Lightman et al., 2024, Wang et al., 2024g,m, Gao et al., 2024a, Lu et al., 2024a, Li et al., 2023b]. In the context of tree search algorithms, process evaluation has been extensively utilized to direct search trajectories. For instance, Tian et al. [2024] employ state scoring within MCTS to guide the search process, whereas Xie et al. [2023] implement state scoring in beam search to optimize path selection. Furthermore, step-level evaluation has proven effective in both error correction and reasoning step summarization. Notably, Zheng et al. [2024], Xi et al. [2024] have developed methodologies capable of pinpointing inaccuracies at specific reasoning steps, thereby furnishing more precise and actionable feedback for comprehensive evaluation.

- **Token-level Evaluation** Some studies argue that step-level evaluation granularity still remains insufficient for comprehensive reasoning assessment [Yoon et al., 2024, Chen et al., 2024h]. This has spurred the development of token-level evaluation frameworks, which offer finer-grained analysis. Yoon et al. [2024] introduce a methodology leveraging a powerful LLM to iteratively modify CoT reasoning at the token level. Their approach assigns distinct rewards to tokens based on their modification operations and utilizes these rewards to train a token-level reward model. Similarly, Chen et al. [2024h] propose a two-stage framework, first training a correction model to identify and rectify erroneous reasoning steps. By associating low generation probabilities with incorrect tokens and high probabilities with correct ones, their method enables the construction of precise token-level reward signals. Furthermore, Lee et al. [2024d] propose a token-supervised value model, which supervises individual tokens to provide a more accurate assessment of solution correctness. Meanwhile, Yang et al. [2024b] derive a token-level evaluation scheme grounded in maximum entropy reinforcement learning principles. Their approach computes token-level values through rank-based truncation, assigning discrete rewards of +1, 0, or -1 to each token, thereby enabling fine-grained optimization of reasoning processes.

Based on the presentation format of evaluation feedback, existing evaluation methods can be categorized into two distinct paradigms: **verifier** and **critic**. The verifier focuses on quantifying solution quality through scalar scoring, while the critic provides verbal feedback in natural language.

- **Verifier** The verifier paradigm assesses the correctness of a solution by assigning a quantitative score. For instance, Cobbe et al. [2021] employ a verifier to estimate the probability of a solution being correct, while Hosseini et al. [2024] leverage a trained DPO verifier to generate likelihood scores that reflect solution validity. Furthermore, [Lightman et al., 2024, Wang et al., 2024g, Lu et al., 2024a] adopt a step-level scoring mechanism, assigning scores to individual reasoning steps and aggregating them using metrics such as the minimum or mean value to derive an overall solution quality assessment. [Tian et al., 2024, Xie et al., 2023] assign scores to each state within a tree search process to optimize the search path. For finer granularity, [Yoon et al., 2024, Chen et al., 2024h, Lee et al., 2024d, Yang et al., 2024b] introduce token-level scoring mechanisms, assigning continuous or discrete scores (e.g., neural, correct, or wrong) to individual tokens.
- **Critic** The critic paradigm generates natural language feedback to facilitate error clarification and enhance the interpretability of scoring mechanisms. For instance, Madaan et al. [2023b] exploit the model's inherent ability to produce critical feedback on its own solutions, enabling iterative refinement. Meanwhile, [Peng et al., 2023, Shinn et al., 2023, Gou et al., 2024] extend this approach by incorporating both internal model states and external environmental information to generate comprehensive critiques, which not only identify errors but also guide subsequent improvements. Further advancing this line of work, [Zheng et al., 2024, Xi et al., 2024] conduct granular, step-by-step critical analyses to pinpoint and rectify errors at a finer level of detail. [Ankner et al., 2024b, Yu et al., 2024b] integrate critique generation with scoring mechanisms. By producing natural language critiques prior to assigning scores, these methods enhance the transparency and reliability of the evaluation process, offering a more interpretable and robust framework for assessing solution quality. Besides, MCTS-Judge Wang et al. [2025b] also models the self-evaluation as a series of subtasks and uses MCTS to decompose problems into simpler, multi-perspective evaluation tasks.

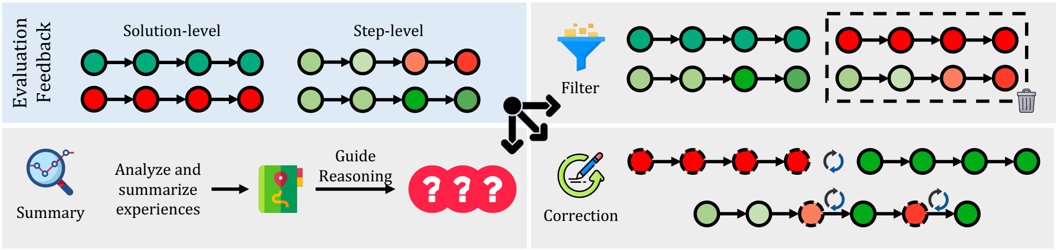


Figure 4: Three post-processing methods following evaluation: Filter, Summary, and Correction.

Challenges of Evaluation: Reward Hacking Reward hacking is defined as the situation where the policy model exploits ambiguities or loopholes in the reward definition to obtain high rewards, without actually learning the desired capacities [Weng, 2024]. Corresponding to specific phases, there are two main paths to mitigate the reward hacking. During the reward modeling phase, designing more complex process rewards may help mitigate this issue. However, overly complex reward signals could also shift the convergence objectives. An alternative approach is to forgo fine-grained PRM and rely solely on ORM, which is particularly suitable for reasoning tasks. For instance, R1 DeepSeek-AI et al. [2025] and T1 Hou et al. [2025] only adpot rule-based outcome rewards based on answer correctness and format compliance, effectively mitigating the reward hacking issue using PRM. Besides, using larger-scale LLMs as the base reward model could improve its generalization ability and reduce the risk of exploiting loopholes. Meanwhile, during RL training, mechanisms such as clipping and reward shaping can help alleviate this issue to some extent [Gao et al., 2024b].

Challenges of Evaluation: Generalization Furthermore, the generalization capability of reward models is equally critical. Parameterized evaluators, such as reward models, are typically trained on specific data distributions, which limits their applicability to out-of-distribution (OOD) tasks. This limitation can lead to biased or unstable evaluations upon novel tasks, further hindering task generalization DeepSeek-AI et al. [2025], Cui et al. [2025]. Therefore, enhancing the generalization ability of reward models to provide reliable feedback across a broader range of tasks is essential for improving task generalization. On one hand, non-parameterized evaluators, such as answer correctness or format accuracy, can be prioritized to mitigate these issues DeepSeek-AI et al. [2025], Hou et al. [2025]. On the other hand, if the parameterized evaluator is necessary, ensuring its continuous updates is crucial. A key challenge lies in efficiently and cost-effectively constructing training data for these evaluators.

Although works like R1 DeepSeek-AI et al. [2025] have circumvented issues such as reward hacking and generalization limitations in existing evaluators by leveraging rule-based outcome rewards, they have also revealed new challenges, including excessively long CoT, inefficient reflection, and overthinking. These issues suggest that relying solely on outcome rewards may be insufficient. A more fine-grained, step-level evaluation could potentially address these shortcomings. Combining the strengths of PRMs and ORMs to achieve fine-grained evaluation while mitigating reward hacking and ensuring generalization remains a significant challenge for future research.

Post-Processing

The reasoning solutions after evaluation can be further processed to enhance their quality and reliability, as shown in Figure 4. First, key information from the reasoning process can be distilled and summarized using Summary Knowledge. For low-quality reasoning solutions, common processing methods include Filtering and Correcting. Filtering directly removes unreliable solutions, while Correcting optimizes the reasoning process by correcting errors or reverting to a correct state. Both methods have their advantages, with Correcting particularly effective in preserving useful information while improving the accuracy of reasoning. By incorporating these post-processing operations, the system can effectively avoid dead ends and iterative reasoning failures, thereby enhancing overall robustness and reliability in complex problem-solving scenarios. The following discussion will delve into these core strategies in more detail.

Summarizing Knowledge from CoT

To improve model performance in reasoning tasks, some studies focus on summarizing the experiences from previous solutions to guide subsequent reasoning. For example, Zhang et al. [2024k] incorporates a reflection component in training instances, such as alternative solutions or problem extensions through analogy and reasoning, guiding the model to understand the problem from different angles and accumulate diverse reasoning experiences. While Wang et al. [2024l] integrate reflection insights into the codebook module via training alignment, allowing the model to actively retrieve relevant reflections to assist reasoning during the process. In tree search reasoning, Hui et al. [2024] identify important nodes and reflects on the subsequent actions and results, generating task-level guidelines to optimize search efficiency and avoid repetitive errors. Meanwhile, Liu et al. [2024c] introduce textual principles for action selection, continually refining these principles through iterative reflection to flexibly guide action execution. Additionally, Zhang et al. [2025a] propose the CoT-based Synthesizer, which improves reasoning by combining complementary information from multiple candidate solutions, generating better solutions even when all candidate solutions are flawed.

Filtering Low-quality CoT

When low-quality solutions are identified during the evaluation phase, the simplest approach is direct filtering. For example, when ground truth is available, low-quality solutions can be filtered based on answer correctness [Singh et al., 2023, Gulcehre et al., 2023]. In the absence of ground answer, filtering strategies can be refined based on consistency, such as perplexity [Min et al., 2024], voting-based consistency [Wang et al., 2023c, Chen et al., 2023a], forward-backward consistency [Jiang et al., 2024b, Weng et al., 2023], or evaluating solution consistency by constructing follow-up questions tailored to the nature of multiple-choice questions [Ankner et al., 2024a, Lee et al., 2024b]. Additionally, learnable verifier [Cobbe et al., 2021, Yu et al., 2023a, Stiennon et al., 2020] can be leveraged to further enhance the filtering process. While simple filtering is both efficient and straightforward to implement, it often results in significant reasoning data waste.

Correcting Low-quality CoT

In addition to direct filtering, incorrect solutions can be corrected to maximize data utilization, making the enhancement of low-quality solutions a critical area of research. Early approaches primarily relied on models' intrinsic capabilities for solution refinement. For instance, Madaan et al. [2023b] iteratively optimize initial outputs using self-generated feedback, while Zhang et al. [2024g] employ LLMs to compare multiple solutions, aggregating their differences into a checklist to improve the stability of self-reflection. However, intrinsic correction methods often fall short of adequately optimizing solutions, highlighting the need for external information to achieve more effective refinement. Recent advancements have introduced various strategies to enhance correction performance. Ferraz et al. [2024], Wu et al. [2024b] refine the correction process by decomposing problems into fine-grained constraints for stepwise critique and optimization or by verifying key conditions through backward reasoning. Gou et al. [2024] integrate feedback from external tools such as Python interpreters and search engines to assist in corrections, while Li et al. [2024b], Gao et al. [2024c], Chen et al. [2023b], Yuan et al. [2024a] utilize Python interpreters to execute code and analyze results, iterating until successful evaluation. Ramji et al. [2024] employ predefined policy metrics as external quality feedback, iteratively refining answers until they meet required standards. Similarly, Wu et al. [2024c] train a PSV model to identify and correct erroneous steps iteratively, ensuring all steps are accurate. Shridhar et al. [2024] train an Asker model to generate sub-questions, aiding in determining whether further corrections are necessary. To systematically enhance models' critique and refinement capabilities, several studies focus on enabling autonomous generation of critical reviews. Zheng et al. [2024], Xi et al. [2024], Yan et al. [2024], Zhang et al. [2024i] propose methods to strengthen these capabilities by training general or specialized models to provide critiques, thereby facilitating the refinement process.

From a theoretical perspective, iterative correction methods can be viewed as a Markov Decision Process (MDP) conducted via linear search, where solutions represent states and corrections represent actions enabling transitions between solutions. While simple linear search often delivers mediocre performance, more sophisticated tree search methods can theoretically yield better outcomes. For example, Zhang et al. [2024e,d] combine Monte Carlo Tree Search (MCTS) with self-evolution mechanisms to optimize solutions for complex mathematical reasoning tasks. Their algorithm initializes a root node and uses a value function Q to select the most promising node for expansion. During the self-evolution stage, model feedback corrects answers and generates higher-quality solutions, which are then scored through self-assessment. These scores are backpropagated to update the tree's value estimates, and the Upper Confidence Bound for Trees (UCT) values of nodes are updated using an improved UCT formula. This iterative process continues until termination criteria are met, simultaneously improving solution quality and exploring new possibilities.

Others

Besides the aforementioned error correction, filtering, and summarization, other post-processing operations like backtracking [Qin et al., 2024, Yang et al., 2025b] can also be conducted: when an error is detected, the system can backtrack to a previous state and explore alternative reasoning paths.

3.2.2 Explicit Tree Search for Short CoT

In this subsection, we define the concept of Short CoT, defined as the CoT that exclusively consists of logical reasoning steps, excluding behaviors such as evaluation and correction. Short CoT represents a streamlined reasoning process from the task prompt directly to the golden answer, with each intermediate step expected to be correct. While traditional CoT often suffers from low accuracy, an enhanced approach leverages heuristic search algorithms—primarily tree search—to improve the efficiency of exploring Short CoT. We summarize these search-based methods under the framework of **Explicit Tree Search**.

Although Short CoT is primarily composed of logical reasoning steps and excludes behaviors such as evaluation, this does not preclude the incorporation of such behaviors during the search process. During the search, an evaluation function is utilized to guide the exploration direction and perform pruning, ensuring that the most promising paths are prioritized. This search method significantly improves search efficiency while maintaining the accuracy and conciseness of the generated CoTs. Based on their underlying search strategies, explicit tree search algorithms can be classified into several categories: naive BFS/DFS, Beam Search, A*, and MCTS.

Search with BFS/DFS Tree-of-Thoughts (ToT) [Yao et al., 2023] decomposes a problem into multiple thought nodes and leverages classic search algorithms, that is, Breadth-First Search (BFS) and Depth-First Search (DFS) to explore diverse reasoning paths, significantly enhancing the problem-solving capability of language models in complex tasks. Qin et al. [2023] combine the search process with tool utilization, employing DFS to handle tool composition and error management, thereby improving the model's performance in real-world tasks. The above methods rely on external programs (e.g., Python code) to define search logic. However, these passive search methods suffer from low efficiency and limited flexibility. Autonomous Tree-Search [Zhang et al., 2023b] guides LLM directly through prompts to perform BFS or DFS independently, enabling the exploration of multiple solution paths and increasing reasoning flexibility. Algorithm-of-Thought [Sel et al., 2023] integrates the strengths of CoT and ToT by using the entire search path of BFS/DFS as a prompt to guide search. This approach allows LLM to dynamically adjust their paths during the CoT reasoning process, enabling more efficient solution discovery. Moreover, AoT avoids the multi-round querying required by ToT, thereby reducing reasoning overhead.

Beam Seach Beam Search, a variant of BFS, maintains k candidate sequences (referred to as the beam) during the search process, achieving an effective balance between search accuracy and computational efficiency. Its alignment with the autoregressive generation of LLMs makes it particularly suitable for guiding forward search during decoding. Based on the granularity of the search, Beam Search can be categorized into three levels: token-level, step-level, and solution-level.

Token-level Beam Search operates at the smallest unit of model generation, directly aligning with the LLM decoding process. While traditional Beam Search ranks sequences based on token log probabilities, this approach prioritizes natural language fluency over reasoning quality. To address this limitation, Lee et al. [2024c] introduce token-supervised value models, which score tokens to enhance the accuracy of mathematical reasoning. Additionally, to mitigate the issue of low diversity in generated sequences, Vijayakumar et al. [2016] propose Diverse Beam Search, which partitions the beam into multiple groups, independently optimizing within each group while applying a diversity penalty between groups to encourage varied reasoning paths.

Step-level Beam Search decomposes multi-step reasoning into sub-steps, scoring and validating each sub-step to maintain high-quality candidate paths. For instance, Wang et al. [2024i], Ma et al. [2023] employ PRM to assign rewards to sub-steps, using these scores to guide the search toward promising reasoning paths. Similarly, Chen et al. [2024b], Yu et al. [2023a] leverage learned value models to enhance search efficiency at the step level, avoiding the computational overhead of MCTS. Setlur et al. [2024] further incorporate process advantages to refine the search process. In contrast to external evaluation methods, Xie et al. [2023] utilize the model itself for self-validation, prompting it to verify step correctness while introducing diversity through temperature-adjusted randomization.

Solution-level Beam Search evaluates entire reasoning paths independently, offering faster inference by avoiding intermediate operations. Best-of-N (BoN) sampling, for example, generates multiple complete solutions and selects the highest-scoring one using a reward model. However, Wang et al. [2024i] highlight the limitations of reward models in distinguishing between similar reasoning processes, proposing a pairwise preference model for more effective ranking. Meanwhile, Wang and Zhou [2024] observe that models can spontaneously generate CoT reasoning through sampling, with CoT-derived answers exhibiting higher confidence. Leveraging this insight, they introduce CoT-decoding, a method that implicitly performs CoT reasoning by altering the decoding process, generating multiple sequences via top-k sampling, and selecting the best based on answer confidence.

Search with A* The A* algorithm optimizes search efficiency by expanding the most promising node using an evaluation function $f(n) = g(n) + h(n)$, where $g(n)$ represents the accumulated cost from the initial state to the current node, and $h(n)$ is a heuristic function estimating the **future cost** to reach the goal. This framework has been adapted to enhance multi-step reasoning in LLMs, outperforming traditional ToT methods in search efficiency.

Several studies have integrated A* principles into LLM reasoning. Zhuang et al. [2023] propose ToolChain*, which maintains a "long-term memory" of reasoning experiences for specific tasks. This memory, initially seeded with example data, dynamically expands by incorporating correct solution paths during reasoning. By matching new tasks to prior experiences using the Longest Common Subsequence, ToolChain* estimates accumulated and future costs, enabling efficient identification of optimal solutions for complex planning and reasoning tasks. In contrast, Wang et al. [2024a] introduce Q*, which employs a learned Q-value model to compute heuristic values $h(n)$ for each state, making the A* algorithm adaptable to domains like mathematics and programming.

Further advancements leverage the inherent capabilities of LLMs to refine A* search. Meng et al. [2024a] propose LLM-A*, which utilizes the global understanding of LLMs to generate waypoints, guiding the A* search direction and reducing unnecessary state exploration. Gupta and Li [2024] train an LLM to learn the residual between the true cost $h^*(n)$ and a heuristic estimate $h(n)$, accelerating search convergence by minimizing iterations. Lehnert et al. [2024] introduce Searchformer, which tokenizes A* execution traces and bootstraps a Transformer model to iteratively sample shorter paths. Similarly, Su et al. [2024] propose Dualformer, which incorporates random information dropout during A* search, enabling the model to balance fast and slow thinking for improved search strategies.

Search with MCTS MCTS is a search algorithm that balances exploration and exploitation, and has demonstrated excellent performance in tasks modeled by Markov Decision Process (MDP)[Chen et al., 2024b, Wu et al., 2024a]. Such tasks often require MCTS to leverage its advantage in balancing exploration and exploitation to find high-value action trajectories within a vast state-action space. One of its most famous applications is the MCTS-based AlphaGo Zero[Silver et al., 2017], which uses MCTS to search for high-quality move sequences in the Go task, continuously improving the performance of its policy network. Inspired by AlphaGo Zero, the idea of using MCTS to search for high-quality reasoning processes in complex action spaces also emerged. A classic MCTS typically consists of the following four steps [Browne et al., 2012]:

- **Selection** Starting from the root node, MCTS trades off exploration and exploitation to calculate the weight of each child node. The calculation of the weight can be done using different design methods. Two common schemes are Upper Confidence Bound (UCB) and Predictor Upper Confidence Tree Bound (PUCT) [Rosin, 2011]. The UCB formula is: $UCB(s,a) = Q(s,a) + c_p \cdot \pi_{prior}(a|s) \cdot \sqrt{\frac{\log N(s)}{1+N(s,a)}}$; and the PUCT formula is: $PUCT(s,a) = Q(s,a) + c_p \cdot \pi_{prior}(a|s) \cdot \frac{\sqrt{N(s)}}{1+N(s,a)}$. Where $Q(s,a)$ represents the action-state value, measuring the accumulated reward after taking action a from state s, $\pi_{prior}(a|s)$ represents the prior probability of taking action a at state s, $N(s)$ is the number of times state s has been explored in the current context, and $N(s,a)$ is the number of times action a has been explored at state s. The final weight considers both exploration and exploitation: if a child node has been explored less, its exploration weight will be increased; if past experience indicates higher expected reward for selecting that node, its exploitation weight will be increased. After selecting an action, the process moves to the corresponding child node and enters the new state. This selection process repeats until the leaf node is reached.

- **Expansion** Once a leaf node is reached, if the leaf node is not a terminal state (e.g., the final answer has not been reached), MCTS will expand the node by selecting different actions and adding several child nodes. The quality of the expansion is mainly influenced by the definition of the action space. Unlike Go, where the action is defined as placing a stone, LLM reasoning requires defining different action spaces for different tasks. Additionally, the definition of action spaces at different granularities for the same task can result in vastly different search outcomes.
- **Evaluation** After reaching the leaf node during the Selection phase, the node's state value must be evaluated. Two common evaluation methods are: 1) Using Monte Carlo sampling to estimate the value. For example, using the state-action sequence from the root to the current node as context, several complete trajectories are sampled, and the statistical metrics (such as success rate) of these rollouts are computed as the leaf node's state value. This method is simple and unbiased, but has high variance and is inefficient, making it impractical in tasks with high sampling costs. 2) Training a value model to directly estimate the leaf node's state value. However, training a value model is more difficult than training a reward model, as it estimates the expected cumulative reward in the future.
- **Backpropagation** After calculating the state value of a node, MCTS updates the state values of all nodes along the path from the leaf node to the root. The state value estimates become more accurate as the number of simulations increases. This process is repeated for multiple simulations until the maximum number of simulations is reached, resulting in a search tree that records the state values and exploration counts for each node. Due to the varying application tasks and specific innovations in methods, MCTS is designed differently for LLM reasoning scenarios.

The granularity of search action in MCTS significantly affects the search space size and the search process efficiency. At the token level, the action space is enormous, and the search process is time-consuming. Therefore, most work models the action space of MCTS at the sentence-level [Xie et al., 2024, Zhang et al., 2024c,j] rather than the token-level. Macro-o1 [Zhao et al., 2024a] further explores this issue and examines the difference between sentence-level actions and actions based on fixed-length token sequences. Futhermore, Jiang et al. [2024a] consider a logical step as an action, which better aligns with human cognitive processes. Inspired by hierarchical RL, Tian et al. [2024] propose an option-based action design.

In the *Selection* stage, the expected cumulative reward estimate $Q(s, a)$ for each action guides the search. Estimating $Q(s, a)$ requires specialized evaluation methods, which can be categorized into three types for existing studies: self-evaluation, external model-based, and consistency-based approaches.

- Self-evaluation leverages LLMs' instruction-following capabilities and prior knowledge to assess correctness. For example, MCTS-DPO [Xie et al., 2024] uses LLMs to predict the probability of correct options as rewards, while another approach uses step generation probabilities as confidence measures [Hao et al., 2023]. While these methods are easy to implement, the inherent stochasticity of LLMs and their sensitivity to sampling temperature introduce potential biases, limiting their reliability.
- On the other hand, external model-based methods employ learned verifiers to guide the search. Rest-MCTS* [Zhang et al., 2024c] learns a PRM to evaluate the quality of each step and Zhu et al. [2022] train both path-level and token-level scorers for more granular assessment. While these methods provide task-specific, fine-grained rewards, they face challenges like limited generalization, continuous updating requirements, and reward hacking disasters.
- Consistency-based methods judge answer quality by the consistency of predictions across multiple samples or different models. Inspired by Self-Consistency (SC) [Wang et al., 2023c], Zhou et al. [2023] use SC metrics for solution evaluation. However, this approach requires extensive samples, leading to inefficiencies and resource consumption. Alternatively, Feng et al. [2023] compute Jensen-Shannon divergence between expert and toy model predictions, with larger distances indicating higher rewards. Similarly, Qi et al. [2024] verify steps by measuring consistency with another model's outputs. However, these methods also face theoretical and computational limitations.
- To guide MCTS exploration, more sophisticated and powerful evaluation methods are necessary. Xin et al. [2024] introduce the intrinsic rewards to incorporate reward-free

exploration in the proof search problem. Besides, many existing studies improve evaluation accuracy by integrating multiple reward signals. For instance, Feng et al. [2023] combine JS divergence, solution generation probability, and self-evaluation for better solution evaluation, and Hao et al. [2023] use generation probability and self-evaluation for step assessment. Zhou et al. [2023] integrate self-evaluation and self-consistency for solution evaluation. These integrated approaches aim to create a robust evaluation framework, optimizing the search process.

In terms of other improvements, Wang et al. [2024k] calculate the static and dynamic gaps between preference pairs collected by MCTS, implementing preference learning on preference pair samples, which improves the learning efficiency of the policy model. Xin et al. [2024] propose three parallelized search methods based on MCTS to improve the efficiency of MCTS search. Jiang et al. [2024a] modify the *Selection* stage and introduces $MCTS_G$, which selects nodes for expansion from a global perspective. This approach enhances the search efficiency of MCTS while ensuring that the search results are closer to the global optimum.

3.2.3 Implicit Trial-and-Error Search for Long CoT

In the previous section, we have introduced the methods based on Short CoT, in which each reasoning step is correct. Any reasoning step deemed unpromising is pruned through algorithmic control, allowing the search to focus on more promising branches. This ensures that only the correct reasoning steps leading to the final solution are retained.

By contrast, Long CoT [DeepSeek-AI et al., 2025, Team et al., 2025, Hou et al., 2025, Xu et al., 2025] does not require each step to be correct but allows the model to explore through trial and error. We define Long CoT as the CoT that emulates human-like cognitive processes. Unlike Short CoT, Long CoT not only encompasses step-by-step logical reasoning but also integrates behaviors such as self-evaluation and self-correction throughout the reasoning process. Correspondingly, we term the process employed by R1 and similar works to explore Long CoT as **Implicit Trial-and-Error Search**. During this process, no external evaluator or refiner are needed; the LLM autonomously triggers its self-evaluation mechanism and utilizes self-correction and backtracking capabilities to adjust the reasoning path, which are crucial for improving reasoning performance. Although Long CoT appears as a linear structure, its correction and backtracking operations make it logically behave like a tree structure. Therefore, we refer to it as an implicit search.

To enable LLMs to generate Long CoT, the O1 journey proposed a distillation method [Huang et al., 2024]. However, simple distillation may only teach the LLM to follow the pattern of trial-and-error search, without cultivating the capacity for valid slow thinking. To address this, DeepSeek-R1 [DeepSeek-AI et al., 2025], Kimi k1.5 [Team et al., 2025], and T1 [Hou et al., 2025] independently proposed using RL to train LLMs in generating Long CoT. DeepSeek-R1 modified PPO [Schulman et al., 2017] with the improved GRPO [Shao et al., 2024] algorithm, while Kimi k1.5 introduced an online policy mirroring variant Lazic et al. [2019] for training. T1 employed the RLOO [Ahmadian et al., 2024] algorithm for optimization. These optimization techniques will be discussed in the following model evolution section(§4).

Notably, all these open-source studies have opted to replace the PRM with the ORM to guide LLMs in autonomously exploring the solution space. This strategic shift has enabled the models to achieve performance comparable to or even exceeding O1. The decision to abandon PRM primarily stems from its limited generalization capabilities and the significant issue of reward hacking, as discussed in detail in the previous section (§3.2.1). Furthermore, as RL training progressed, these studies found that the LLM-generated CoTs became progressively longer, with reasoning accuracy and generalization steadily improving. During this process, LLMs demonstrated “aha moments” [DeepSeek-AI et al., 2025], automatically conducted verification and alternative reasoning thoughts and manifested as an epiphany.

3.2.4 Difference and Unification between Explicit Tree & Implicit Trial-and-Error Searches

Difference between Explicit Tree & Implict Trial-and-Error Searches

Before delving into a detailed comparison, we briefly summarize the procedural distinctions between explicit tree search for Short CoT and trial-and-error search for Long CoT:

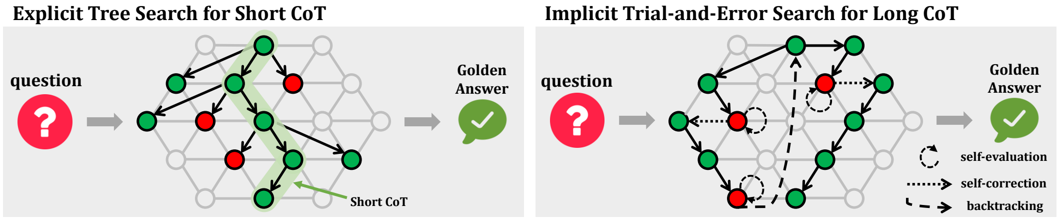


Figure 5: An illustration of two search paradigms. Explicit Tree Search improves search efficiency by simultaneously expanding multiple steps and prioritizing the most promising ones, thereby identifying logically consecutive Short CoTs. In contrast, Implicit Trial-and-Error Search mimics human thinking by step-by-step reasoning, trial-and-error, and backtracking to past states to explore alternative thoughts.

- As shown in Figure 5, explicit tree search employs heuristic search algorithms (e.g., MCTS, A*, and beam search) to explore the solution space. At each state, multiple actions are expanded for candidate states, resulting in a tree-structured search process. During this process, the reasoning system passively invokes operations such as evaluation and pruning. Each reasoning step in the generated CoT is guaranteed to be correct, and behaviors like evaluation, pruning, and error correction are not presented in the Short CoT.
- In contrast, implicit trial-and-error search does not rely on heuristic algorithms. Instead, the LLM actively invokes capabilities such as self-evaluation and self-correction during the reasoning process, expressing these operations in natural language. As a result, the Long CoT in trial-and-error search not only includes step-by-step reasoning but also incorporates self-evaluation, self-correction, backtracking, and other operations, making the reasoning process more transparent and dynamic.

In terms of performance, tree search has also seen successful implementations like rStar-Math [Guan et al., 2025], which uses MCTS and PRM along with self-evolution training to achieve performance comparable to O1 on small LLMs. However, recent open-source projects, including DeepSeek R1 [Team, 2024a] and Kimi k1.5 [Team et al., 2025], collectively choose the trial-and-error search route for remarkable generalization [Yeo et al., 2025].

The reasons why these open-source projects abandoned explicit tree search for Short CoT in favor of trial-and-error search for Long CoT can be inferred from their technical reports:

- First, tree search often relies on verifiers like reward models or value models to provide scores. While these verifiers offer fine-grained evaluation guidance, they suffer from limited generalization and severe reward hacking issues, leading to inaccurate intermediate evaluations and even training collapse due to the LLM exploiting shortcuts to maximize rewards. In contrast, R1, Kimi k1.5, and T1 leverage self-evaluation capabilities during search and employ rule-based outcome rewards during training, significantly mitigating reward hacking and improving generalization.
- Additionally, the scores from verifiers in tree search only reflect the relative quality of reasoning, failing to pinpoint errors or causes, resulting in limited evaluation quality. In contrast, R1 and similar projects generate verbal evaluation feedback via self-evaluation, offering richer and more informative feedback.
- Finally, while tree search can explore multiple paths simultaneously, these paths are independent. Thus, intermediate experience cannot be shared among them, lowering the utilization of parallel reasoning processes. This makes tree search significantly different from human reasoning, where insights from past errors guide subsequent reasoning, as seen in trial-and-error search with Long CoT.

While the above discussion highlights the weaknesses of explicit tree search compared to trial-and-error search, it does not imply that trial-and-error search is free from drawbacks.

- The application of Long Long CoT in trial-and-error search might introduce inefficiencies in two key aspects. 1) For simple tasks, Long CoT methods tend to exhibit **overthinking**. As noted by [Chen et al., 2024f], approaches such as QwQ [Team, 2024b] and R1 [DeepSeek-AI

	Explicit Tree Search for Short CoT	Implicit Trial-and-Error Search for Long CoT
Operators in CoT	Step-by-Step Reasoning	Step-by-Step Reasoning, Evaluation, Post-Processing
Evaluation Method	Score-based Verifier	Verbal-based Critic
Strengths	1. Less affected by overthinking and underthinking; 2. Resulting in concise CoT.	1. High generalization performance; 2. Mimicking human thought processes. 3. Key insights in previous trials can be utilized in subsequent reasoning.
Weaknesses	1. Error experiences from other reasoning branches cannot be utilized in subsequent reasoning; 2. Suffering from low generalization capacities.	1. Suffering from overthinking on easy tasks and underthinking on hard tasks.
Representative Works	O1-Coder [Zhang et al., 2024j], rStar-Math [Guan et al., 2025]	R1 [DeepSeek-AI et al., 2025], Kimi k1.5 [Team et al., 2025]

Table 1: A comparison of two search paradigms.

et al., 2025], often explore multiple potential solutions even when the initial solution is typically sufficient. This propensity for excessive exploration incurs substantial computational overhead. 2) For complex tasks, Wang et al. [2025a] observe that QwQ and R1 are prone to **underthinking**. These methods frequently abandon promising reasoning paths and switch strategies prematurely, resulting in unstable and inefficient search processes accompanied by unnecessarily lengthy reasoning chains. In contrast, Short CoT-based methods produce more concise reasoning paths, providing clear efficiency benefits. [Wu et al., 2025b, Xie et al., 2025a] further argue that longer CoTs do not necessarily improve reasoning performance; instead, an optimal CoT length exists for each model and task. Thus, the inefficiency of trial-and-error search not only increases token usage and computational costs but also degrades performance.

- Furthermore, implicit trial-and-error search heavily relies on the self-evaluation and self-correction capabilities of LLMs. On one hand, the background mechanisms of these abilities remain an area for further research; on the other hand, these capabilities have not been specifically optimized during the learning process of LLMs. Models such as R1 [DeepSeek-AI et al., 2025], kimi k1.5 [Team et al., 2025], and T1 [Hou et al., 2025] simultaneously learn reasoning, evaluation, reflection, and error correction within the same action space using only outcome-level rewards, but lack dedicated reward signals to guide the learning of evaluation, reflection, and correction abilities. As a result, these capabilities in LLMs are not specially optimized, and one consequence is that even when LLMs engage in low-quality reflection or error correction during early stages, they can still receive positive rewards as long as the final answer is correct. Moreover, the insufficiency of self-evaluation capabilities is one of the reasons why methods like R1 frequently fail to assess reasoning paths accurately, thus abandoning promising ones prematurely.

To address the inefficiency issue, Kimi k1.5 [Team et al., 2025] introduces a length penalty as part of the length reward for response length control. Yeo et al. [2025] designs the Consine Reward function, where the reward for a correct response increases with shorter lengths, while for incorrect responses, the reward increases with longer lengths. Luo et al. [2025] propose the Length-Harmonizing Reward to suppress excessively long responses. In addition to introducing new reward functions, Chen et al. [2024f] employ preference learning, treating the shortest response as positive and the longest response as negative, thereby encouraging LLMs to generate shorter CoTs and suppressing the generation of overly long ones. We briefly summarize the differences between these two search methods in Table 1.

Unification between Explicit Tree & Implicit Trial-and-Error Searches

These two search strategies—tree search and trial-and-error search—each offer distinct advantages, raising a critical question: what is the relationship between them, and can they be unified? We explore this question from two perspectives.

First, we analyze the correlation between these two searches from the perspective of action space, focusing on the roles of different meta-operators.

- Initially, both strategies include step-by-step reasoning, with Short CoT primarily consisting of logically consecutive reasoning steps.

- However, the two strategies diverge significantly in their evaluation mechanisms. Explicit tree search typically requires learning a PRM or Value Model to assess reasoning quality, which introduces high bias due to the poor generalization capabilities of these models. In contrast, trial-and-error search relies on the LLM's intrinsic self-evaluation capability to assess reasoning states.
- Regarding post-processing, we take "correction" as an example for analysis. Tree search generally lacks direct correction operations, although switching between branches can be viewed as a form of indirect error correction. However, such "corrections" cannot leverage the contents of previous trials, as they are merely sampled in advance during the earlier expansion stage, where different trials remain independent of one another. For instance, in the expansion phase of MCTS, multiple child candidate actions are sampled simultaneously. During the subsequent selection phase, the action chosen in the current state may differ from that in the previous simulation, which can be viewed as a form of "correction." However, the new action selected in this simulation is not generated based on the evaluation feedback of the action chosen in the previous simulation; instead, both actions are sampled independently during the expansion phase.

Therefore, the primary limitation of current tree search methods, compared to trial-and-error search, mainly lies in their constrained action space. If the action space of tree search were expanded to incorporate actions such as evaluation and correction, it could also facilitate the exploration of Long CoT in theory Lin et al. [2025].

From the perspective of reasoning capability evolution, Long Long CoT serves as an effective approach to solving novel problems, while Short CoT represents the ultimate goal achieved through continuous training on Long CoT. Specifically, when faced with challenging tasks, humans initially engage in trial-and-error search and eventually identify efficient pathways to solve such tasks. These efficient pathways can be learned to reduce unnecessary trial and error, thereby shortening the Long CoT. Thus, Long CoT can be viewed as an initial and intermediate solution for handling complex tasks. Once a task is solved, the knowledge distilled from Long CoT can be used to learn Short CoT, which in turn serves as prior knowledge to reduce the trial-and-error iterations of Long CoT when tackling even more complex tasks. In summary, a robust reasoning system should possess the ability to dynamically switch between Long CoT and Short CoT, enabling it to adaptively balance exploration and efficiency in problem-solving.

4 Model Evolution

After collecting high-quality reasoning data, the next step is to further improve the capabilities of the models in the system, thereby providing a solid model foundation for the next round of data improvement. Given the limited research on Task Creator, we will primarily focus on the training methods for the Reasoner, Evaluator, and Post-Processor modules. From the perspective of training methods, we summarize existing work from the view of Reinforcement Learning, with specific methods including Behavior Cloning, Preference Optimization, and Reinforcement Learning.

4.1 Background RL Knowledge

For direct references to certain algorithms in the main body, we would first like to introduce several representative reinforcement learning algorithms here.

4.1.1 Set Sail from RLHF

Given the tremendous success of products like ChatGPT [Ouyang et al., 2022] and Claude OpenAI [2024a], we introduce RL in LLM post-training, specifically starting from Reinforcement Learning with Human Feedback (RLHF) [Ouyang et al., 2022]. RLHF, as a preference-based reinforcement learning framework, consists of two key stages [Wang et al., 2024h]:

- **Rewarding**: In this step, preference data is collected to train the reward model r_θ. Early methods for collecting preference data involved manual annotation, where multiple responses were sampled for the same prompt, and annotators were asked to rank the responses based on quality. Based on this preference ranking, the reward model is trained to capture human preferences:

$$\max_{\theta} \mathbb{E}_{(x,y^w,y^l)\in\mathcal{D}}[\log(\sigma(r_\theta(x,y^w) - r_\theta(x,y^l)))], \tag{2}$$

where $y^w \succ y^l$ is the preference relationship labeled by the annotators.

- **Policy Optimization**: In this step, the LLM is fine-tuned as a policy model π_ϕ, with the goal of maximizing the reward it can receive. During this process, the LLM generates content, the reward model scores the content, and then PPO [Schulman et al., 2017] is used to train the LLM:

$$E_{x\sim D, y\sim\pi_\phi(.|x)}[r_\theta(x,y) \quad -\beta D_{KL}(\pi_\phi(y|x)||\pi_{ref}(y|x))], \tag{3}$$

where π_{ref} is the reference model, typically an LLM with frozen parameters after supervised fine-tuning (SFT). The KL divergence term $D_{KL}(\pi_\phi(y|x)||\pi_{ref}(y|x))$ aims to prevent the policy model from deviating too far from the reference model while also maintaining diversity in the generated outputs and preventing the model from collapsing to a single high-reward answer.

Although RLHF was initially employed to optimize LLMs for the alignment tasks, this training framework can clearly be adapted to optimize the reasoning capabilities of LLMs. By explicitly constructing preference data from the correctness of reasoning outcomes, it can encourage the model to generate correct reasoning processes while discouraging incorrect ones, thus enhancing its reasoning ability.

4.1.2 From RLHF to more fine-grained PPO

Although RLHF uses PPO for optimization, in practice, classical RLHF is often regarded as a bandit method, where the entire sentence is treated as a single action [Zhong et al., 2024]. This is because RLHF relies solely on outcome-level rewards, resulting in a lack of finer-grained optimization signals.

In general, sparse rewards increase the difficulty of the learning process compared to dense rewards [Andrychowicz et al., 2017], and this is especially evident in complex reasoning tasks. For example, in multi-step reasoning, a failed solution does not necessarily mean that all reasoning steps are incorrect; it is possible that the first few steps were correct while later steps contained errors. However, using only outcome rewards would suppress the correct reasoning steps during training. Therefore, relying solely on outcome rewards may not fully leverage RL's potential. An improvement is to use step-level or even token-level rewards to provide finer-grained optimization signals. How to integrate these finer-grained reward signals into the RL training process requires revisiting the PPO algorithm.

PPO [Schulman et al., 2017] is a classic on-policy algorithm that has proven its effectiveness and stability across various fields. In its general form, the training objective of PPO is:

$$\mathbb{E}_{q\sim P(Q), y\sim\pi_\phi(y|q)} \frac{1}{|y|} \sum_{t=1}^{|y|} \min[\frac{\pi_\phi(y_t|q, y_{<t})}{\pi_{ref}(y_t|q, y_{<t})} A_t, \text{clip}(\frac{\pi_\phi(y_t|q, y_{<t})}{\pi_{ref}(y_t|q, y_{<t})}, 1-\epsilon, 1+\epsilon) A_t], \tag{4}$$

where y represents the text generated by the policy model, and $|y|$ denotes the number of characters in y. $A_t = Q(s_t, y_t) - V(s_t)$ is the advantage function, which normalizes the action-value function $Q(s_t, y_t)$ to be around the state-value baseline $V(s_t)$, helping to reduce the variance and improve learning stability. In practice, the Generalized Advantage Estimation (GAE) form of A_t is often employed, which balances the trade-off between bias and variance in the estimation:

$$\begin{aligned} A_t^{GAE} &= \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) = Q(s_t, a_t) - V(a_t), \end{aligned} \tag{5}$$

where γ is the discount factor, and λ is also a hyperparameter within $[0,1]$. When $\gamma = 0$, $A_t^{GAE} = \delta_t = Q(s_t, a_t) - V(s_t)$.

Although PPO has proven its effectiveness through RLHF, its high training resource requirements hinder its widespread application in improving LLM reasoning. Specifically speaking, the full PPO framework includes four modules: policy model, reference model, value model, and reward model. The initialization of the latter two modules further increases the complexity of the training process and impacts the stability of the policy model. As a result, a series of improvements have emerged to simplify the PPO framework and training pipeline, such as by bypassing the direct modeling and learning of the value model [Shao et al., 2024] or reward model [Rafailov et al., 2023]. Below, we introduce some representative works that aim to simplify the PPO training process.

4.1.3 From PPO to REINFORCE

In order to reduce the overall training device burden, recent work has revisited the potential of REINFORCE [Sutton et al., 1999] for optimizing LLMs [Li et al., 2023d, Ahmadian et al., 2024]. REINFORCE is a classic Policy Gradient algorithm and traditionally optimizes the following objective:

$$\sum_{t=1}^{T} \mathbb{E}_{a_t \sim \pi_\phi(a_t|s_t)}[R(s_t, a_t) \log \pi_\phi(a_t|s_t)], \tag{6}$$

where $R(s_t, a_t) = \sum_{s=t}^{T} \gamma^{s-t} r_\theta(s_t, a_t)$ refers to the cumulative reward to control the direction and step size of the policy gradient updates.

However, REINFORCE algorithm is known to suffer from high variance in the $R(s_t, a_t)$, leading to instability during training. To mitigate this issue, methods replace the cumulative reward with the action-value function $Q(s_t, a_t)$ or the advantage function $A(s_t, a_t)$ as exemplified by the PPO algorithm. Alternatively, variance reduction can also be reduced by subtracting a baseline:

$$\mathbb{E}_{a_t \sim \pi_\phi(.|s_t)}[(R(s_t, a_t) - b) \log \pi_\phi(a_t|s_t)]. \tag{7}$$

The baseline $b(s_t)$ can be implemented in various ways. To avoid the need for an additional value model, ReMax [Li et al., 2023d] opts to use the reward of the action with the highest probability as the baseline:

$$b(s_t) = r(s_t, a_t), \quad a_t \in \arg\max \pi_\phi(.|s_t). \tag{8}$$

Ahmadian et al. [2024] proposed the REINFORCE Leave-One-Out (RLOO) estimator. Given a task q, RLOO samples multiple responses $\{r_1, r_2, ..., r_K\}$ and then uses the mean of these sampled trajectories as the baseline:

$$b(r^i) = \frac{1}{k-1} \sum_{j \neq i} R(r^j, x). \tag{9}$$

[Ahmadian et al., 2024] found that, in a Bandit setting where only outcome rewards are available, RLOO outperforms PPO. A possible reason for this is that LLM, after large-scale pretraining and instruction fine-tuning, is an extremely strong initialization policy. As a result, the variance issues at the sentence level in the sampled trajectories are not as pronounced. Furthermore, by estimating the value function through sampling, RLOO reduces variance while avoiding the bias introduced by learning a value function.

However, this advantage may only hold in bandit settings. Although using REINFORCE [Sutton et al., 1999] eliminates the value model and reduces the learning cost, RLOO [Ahmadian et al., 2024] might suffer from more significant variance in multi-hop reasoning tasks, which require denser rewards, such as step-level or token-level rewards.

Framework	Online	Remove VM	Remove RM	Remove RefM	Low Variance	Architecture
PPO Schulman et al. [2017]	✓	✗	✗	✗	✓	Actor-Critic
RLOO Ahmadian et al. [2024]	✓	✓	✗	✓	✗	Policy Gradient
GRPO Shao et al. [2024]	✓	✓	✗	✗	✓	Actor-Critic
DPO Rafailov et al. [2023]	✗	✓	✓	✗	✓	BT Model
PRIME (RLOO) Cui et al. [2025]	✓	✓	✗	✗	✓	Policy Gradient

Table 2: A comparison of 5 RL algorithms: PPO, RLOO, GRPO, DPO, and PRIME. VM means Value Model; RM is Reward Model; and RefM means Reference Model.

4.1.4 From PPO to GRPO

When step-level or token-level rewards are available, using PPO to fine-tune the policy model is an ideal choice, as PPO ensures the stability of the training process through the advantage function and the clip operation. However, as can be seen in Eq. 5, in order to compute A_t^{GAE}, both a reward model and a value model $V(s_t)$ are required. Typically, the value model often matches the reasoner model in parameter scale, and its training is challenging due to difficulties in data acquisition and learning instability. Therefore, the introduction of a value model not only increases the resource burden but also leads to training instability.

To address this challenge, Shao et al. [2024] propose the GRPO algorithm to modify PPO by replacing the value model with Monte Carlo (MC) sampling. Specifically, for a given task q, GRPO simultaneously samples G complete solutions $y_1, y_2, ..., y_G$ as a group, and reward signals can be provided for each solution based on the reward function. Depending on the granularity of reward signals, GRPO provides two versions. When the PRM is available, it assigns rewards for all steps of each solution and obtains the reward set $R = \{r_1^{index(1)}, ..., r_1^{index(k_1)}, ..., r_G^{index(1)}, ..., r_G^{index(k_G)}\}$, where k_i represents the number of steps in y_i and $r_i^{index(j)}$ denotes the index of the end token in the j-th step of y_i. Therefore, the advantage function in GRPO is calculated as follows:

$$\begin{aligned} \tilde{A}_{i,t} &= \sum_{index(j) \geq t} \tilde{r}_i^{index(j)}, \\ s.t. \quad \tilde{r}_i^{index(j)} &= \frac{r_i^{index(j)} - \text{mean}(R)}{\text{std}(R)}. \end{aligned} \tag{10}$$

When the Outcome Reward Model (ORM) is available, it assigns rewards for all G solutions, resulting in the reward set $R = \{r_1, ..., r_G\}$. The advantage function in GRPO is then computed as:

$$\tilde{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(R)}{\text{std}(R)}. \tag{11}$$

Regardless of whether PRM or ORM is used, rewards within the group are normalized, with the mean replacing the value model as the baseline. During training using the group trajectories, actions with below-average rewards are suppressed, while those with above-average rewards are reinforced. Finally, the optimization objective of GRPO is:

$$\begin{aligned} &\mathbb{E}[q \sim P(Q), \{y_i\}_{i=1}^G \sim \pi_{\phi_{old}}(y_i|q)] \frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \\ &\left\{ \min[\frac{\pi_\phi(y_{i,t}|q, y_{i,<t})}{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})} \tilde{A}_{i,t}, \text{clip}(\frac{\pi_\phi(y_{i,t}|q, y_{i,<t})}{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})}, 1-\epsilon, 1+\epsilon) \tilde{A}_{i,t}] - \beta \mathbb{D}_{KL}[\pi_\theta || \pi_{ref}] \right\}, \end{aligned} \tag{12}$$

where $\pi_{\phi_{old}}$ is the reference model from the previous training iteration. Additionally, GRPO enhances training stability by incorporating KL divergence on top of PPO. However, instead of the conventional KL implementation, it employs an alternative unbiased estimator [Schulman, 2020] to ensure positive and stable:

$$\mathbb{D}_{KL}[\pi_\phi || \pi_{\phi_{old}}] = \frac{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})}{\pi_\phi(y_{i,t}|q, y_{i,<t})} - \log \frac{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})}{\pi_\phi(y_{i,t}|q, y_{i,<t})} - 1. \tag{13}$$

In summary, GRPO estimates the advantage function through MC sampling, which allows the overall framework to include only the policy model, reference model, and reward model. This approach maintains the advantages of PPO while simplifying the training architecture. Moreover, since LLMs inherently possess strong prior knowledge, variance issues are less pronounced. The use of Monte Carlo sampling to estimate the baseline ensures unbiased estimation. Additionally, the unified optimization within groups reinforces high-quality trajectories, further stabilizing the training process. As a result, GRPO has been successfully applied in many open-source O1-like works [Shao et al., 2024, Yang et al., 2024a, Wang et al., 2024e, DeepSeek-AI et al., 2025].

4.1.5 From PPO to DPO

RLHF requires modeling and pretraining a reward model explicitly, increasing the computational resource demands and the training complexity. To address this, DPO first points out the closed-form solution in Equation 3:

$$\pi^*(y|x) = \frac{1}{Z(x)} \pi_{ref}(y|x) \exp(\frac{1}{\beta} r(x, y)). \tag{14}$$

This conclusion indicates that the optimal policy model $\pi^*(y|x)$ is strongly bound to the reward model $r(x, y)$. In other words, by setting a certain reward model, a specific optimal policy can be optimized, which maximizes the probability of the optimal trajectory implied by this reward model. DPO further transforms Equation 14, leading to a new expression:

$$r(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{ref}(y|x)} + \beta \log Z(x). \tag{15}$$

That is, the reward model $r(x, y)$ can be represented by the policy model $\pi(y|x)$. Based on this, Rafailov et al. [2023] argue that it may be more efficient to directly optimize the $\pi(y|x)$, rather than performing reward modeling first and then using the learned $r(x, y)$ to guide the optimizing $\pi(y|x)$.

RLHF uses Bradley-Terry model to learn the reward model:

$$\mathbb{E}_{(x,y_w,y_l)\in D, y_w \succ y_l}[\sigma(r_\theta(x, y_w) - r_\theta(x, y_l))]. \tag{16}$$

By bringing Equation 15 into Equation 16, the objective of reward modeling can be transformed into directly learning the policy model:

$$\mathbb{E}_{(x,y_w,y_l)\in D}[\log \sigma(\beta \log \frac{\pi_r(y_w|x)}{\pi_{ref}(y_w|x)} - \beta \log \frac{\pi_r(y_l|x)}{\pi_{ref}(y_l|x)})]. \tag{17}$$

Although DPO has simplified RLHF and lowers the threshold for optimizing LLMs, several issues have emerged in subsequent applications:

- **Insufficient Granularity of Optimization**: The vanilla DPO approach focuses on preference learning at the response level, which suffers from overly coarse granularity, particularly in tasks involving complex reasoning that require multiple steps. Such coarse granularity may result in instability during the preference learning, where even partially correct reasoning steps within erroneous solutions are incorrectly treated as negative steps uniformly. To address this challenge, subsequent works have introduced methods like step-DPO and token-DPO, which enable more fine-grained preference optimization. A comprehensive discussion of these will be provided in Section 4.2.2.
- **Data Distribution Shift**: DPO is commonly optimized using the offline setting, where a fixed dataset of preference data is first collected using π_{ref}, and subsequently, DPO is used to train the policy model π_ϕ. While this approach offers high training efficiency, training exclusively on a static offline dataset may hinder the learning process [Chen et al., 2024a]. To address this issue, subsequent work has explored the adaptation of DPO for online learning [Chen et al., 2024a]. Specifically, after collecting a batch of preference data, DPO is applied to train the policy model, which then replaces π_{ref} with the newly trained model π_ϕ before collecting the next batch of preference data.
- **Suppression of Positive Samples**: A notable issue of DPO is the unexpected suppression of positive samples during training, alongside the reduction in negative sample probabilities. This phenomenon can occur when the distinction between positive and negative samples is not sufficiently pronounced. To address this, several studies have proposed modifications to the DPO loss function, incorporating regularization terms that quantify the difference in quality between positive and negative samples [Azar et al., 2023, Le et al., 2024].
- **Data Collection and Reward Signal Utilization**: One limitation of DPO is its lack of explicit modeling of the preference degree [Wang et al., 2024b]. In scenarios where reward values are available, DPO constructs preference pairs by solely comparing the rewards, neglecting the direct use of these reward signals. This results in insufficient utilization of the reward signal. Furthermore, reliance on preference pair data increases the construction difficulty of training data. To address these limitations, OREO [Wang et al., 2024b] proposes a novel offline RL algorithm that requires only reward signals, eliminating the need for preference data.

Besides, experimental results indicate that DPO generalizes worse than PPO [Li et al., 2023c], with some tasks even benefiting more from direct SFT [Yuan et al., 2024b, Chen et al., 2024d]. fDPO [Wang et al., 2023a] extends the method by incorporating divergence constraints to manage a broader range of preference complexity and model uncertainty. cDPO [Chowdhury et al., 2024] enhances the robustness of DPO, ensuring consistent performance in environments with noisy feedback. KTO [Ethayarajh et al., 2024] combines model behavior with human decision patterns using the Kahneman-Tversky function based on psychological factors. GPO [Tang et al., 2024] defines a family of convex functions to parameterize the loss function, aiming to theoretically unify the DPO preference alignment approaches. ORPO [Hong et al., 2024] introduces a new method for optimizing preferences without the need for a reference model, simplifying the optimization process and broadening its applicability.

4.1.6 From PPO to PRIME

Rafailov et al. [2024] further analyze DPO and introduce the concept of **Implicit Reward**, which is formulated as follows:

$$r(x, y) = \beta \log \frac{\pi_\theta(x, y)}{\pi_{ref}(y|x)}. \tag{18}$$

Rafailov et al. [2024] argue that the policy model trained using DPO essentially serves as a token-level reward function, where the reward assigned to each token is exactly the implicit reward. The effectiveness of implicit rewards has been proven in several works [Zhong et al., 2024, Chen et al., 2024a].

Yuan et al. [2024c] prove that by defining the outcome reward function as $r_\theta(y) = \beta \log \frac{\pi_\theta(x,y)}{\pi_{\text{ref}}(y|x)}$, the learned ORM can directly compute token-level rewards. In other words, the ORM learned in this format inherently functions as a PRM. Specifically, PRIME [Cui et al., 2025] comprises four key components: a policy model π_ϕ, an outcome reward verifier r_o, a PRM π_θ, and its corresponding reference model π_{ref}. After generating a response y, PRIME first obtains the outcome-level reward $r_o(y)$ and trains $r_\theta(y)$ using a cross-entropy loss:

$$r_o(y) \cdot \log \sigma(r_\theta(y)) + (1 - r_o(y)) \cdot \log(1 - \sigma(r_\theta(y))), \tag{19}$$

where $r_\theta(y)$ is optimized to approximate the true outcome reward. During this process, the PRM π_θ is also optimized. The trained PRM π_θ can then express token-level rewards for each token y_t as:

$$r_\theta(y_t) = \beta \log \frac{\pi_\theta(y_t|x, y_{<t})}{\pi_{\text{ref}}(y_t|x, y_{<t})}, \tag{20}$$

which is exactly the implicit reward. By leveraging the trained π_θ, PRIME provides token-level rewards for training the policy model π_ϕ. This design enables PRIME to integrate seamlessly with various existing reinforcement learning algorithms, such as RLOO, as demonstrated in the paper.

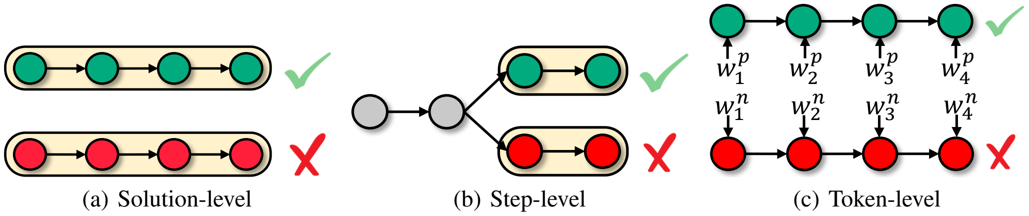
The core idea behind PRIME is to distribute the outcome reward across individual tokens, allowing the model to learn token-level rewards through extensive sampling. Tokens that contribute to higher outcome rewards are assigned higher token-level rewards. PRIME learns token-level rewards directly from outcome rewards without requiring manual annotation, thus it can simultaneously train both the policy model and the reward model. This dual training approach mitigates issues such as reward hacking and improves the generalization capability of the reward model.

4.2 Reasoner Optimization

4.2.1 Behavior Cloning

After collecting reasoning process data, the most straightforward method to optimize LLMs is through Behavior Cloning (BC), i.e., Supervised Fine-Tuning (SFT). However, BC can theoretically only be conducted on correct data. Yuan et al. [2023a], Tong et al. [2024] introduce rejection fine-tuning, where incorrect reasoning trajectories are filtered out based on the answers, followed by fine-tuning on the correct trajectories. Although this approach ensures the quality of training data, it results in considerable data wastage. To enhance the volume of correct solutions, Zelikman et al. [2022] regenerate the reasoning process by incorporating a Rationalization step for incorrect solutions. Zhang et al. [2023a] introduce HIR to leverage incorrect solutions by relabeling them. For failure solutions, HIR relabels the instruction, for example, changing “Generate a correct answer to this problem” to “Generate a wrong answer to this problem.” This hindsight approach allows the utilization of incorrect solutions without introducing additional parameters. Concurrently, Zhang et al. [2024c], Wang et al. [2024k] apply MCTS to improve the efficiency of searching for correct trajectories. Additionally, Chen et al. [2024e] expand the dataset by constructing reverse problems and performing SFT on them, thereby enabling the models to learn reverse reasoning capabilities.

While these approaches either increase the proportion of correct data or modify instructions to utilize failure data, they still face challenges in maximizing data utilization and exploring negative data.



(a) Solution-level (b) Step-level (c) Token-level

Figure 6: Three granularities for preference optimization: Solution-level, Step-level, and Token-level.

4.2.2 Preference Optimization

Preference optimization is a widely used method to enhance LLM reasoning capability. The core idea is to push up the probability of high-quality CoTs while pushing down the probability of inferior ones, during which increases the reasoning ability of the target model.

Early works, such as RRHF [Yuan et al., 2023b], implement preference learning from a ranking perspective. RRHF uses a reward model to rank responses y_{i_k} sampled from different sources, constructing preference pairs, and then directly optimizes the LLM through ranking loss:

$$\begin{aligned} p_i &= \frac{\sum_t \log \pi_\phi(y_{i,t}|x, y_{i,<t})}{||y_i||}, \\ L_{rank} &= \sum_{r_i<r_j} max(0, p_i - p_j). \end{aligned} \tag{21}$$

Other preference optimization algorithms, represented by DPO Rafailov et al. [2023], further simplify the RLHF training process while overcoming the limitation of SFT. Due to its easy implementation, DPO has seen widespread employment across various tasks.

In the following, we will focus on the researches of DPO in enhancing LLM reasoning abilities. Existing methods that utilize preference optimization to improve reasoning can be categorized based on the granularity of the preference data: solution-level, step-level, and token-level optimization.

Solution-level Preference Optimization

Solution-level preference signals are often the most accessible, leading early preference optimization efforts to focus primarily on this level. Given a set of solutions, [Pang et al., 2024, Jiang et al., 2024a] categorize them into correct and incorrect groups based on answer labels, creating preference pairs for optimization. However, in self-evolution scenarios where answer labels are typically unavailable, preference data can be constructed using methods such as LLM-as-a-Judge [Gu et al., 2024] or pre-trained reward models [Yu et al., 2024a, Ouyang et al., 2022]. For instance, Yuan et al. [2024d] utilize the LLM's self-evaluation capability to score its own generated solutions. Despite this, LLMs' self-evaluation abilities remain limited, and the generalizability of reward functions is constrained, rendering the evaluation process susceptible to noise. To address this issue, Wang et al. [2024c] propose the Uncertainty-enhanced Preference Optimization framework, which employs a Bayesian Neural Network-based estimator to quantify the uncertainty of each preference pair. By integrating this uncertainty into the DPO training process, the framework enables the LLM to evaluate the reliability of preference pairs during optimization, thereby enhancing robustness.

Step-level Preference Optimization

Although solution-level preference data is relatively easy to obtain, the coarse granularity preference often leads to insufficient optimization of the LLM policy. An intuitive explanation is that in an incorrect reasoning process, the error may only begin at a specific step, while the preceding reasoning steps are correct. Direct learning at the solution level may reduce the probability of some correct reasoning steps. To address this, researchers have explored preference optimization at the step level. Relevant works can be classified into two categories: **active construction** and **tree search**.

The core idea of the active construction approach revolves around the targeted sampling of either incorrect or correct reasoning trajectories that share the same prefix. Hwang et al. [2024] initially employ MC sampling to identify the first erroneous step in a flawed trajectory y^-. From this step onward, each step and its preceding context are concatenated, and multiple reasoning processes are sampled. If all generated reasoning processes from a particular step fail, that step is classified as incorrect. Subsequently, correct reasoning steps are generated by using the preceding steps as context, resulting in a correct trajectory y^+. The combination of y^+ and y^- forms a preference pair with an identical prefix. Preference alignment algorithms, such as DPO, are then applied to optimize the model, enabling the LLM to concentrate on refining the trajectory suffix that influences correctness. Lai et al. [2024] adopt a similar methodology, utilizing GPT-4 as a monitor to detect incorrect steps and subsequently applying DPO for step-level preference optimization, termed Step-DPO. In contrast, Lu et al. [2024c] employ a different strategy: starting from a correct trajectory, it generates incorrect subsequent steps by setting a high temperature to induce a failure suffix.

Tree search-based methods, on the other hand, directly search for preference pairs from tree search results. Zhang et al. [2024h] use the Tree-of-Thought [Yao et al., 2023] for tree search. During the search, nodes are evaluated using self-evaluation. Once the correct reasoning path is found, preference pairs are constructed using the nodes along that path. Any step filtered out during BFS search from a correct path's node can be considered as a negative step, forming a preference pair with the correct path node. Compared to ToT, many works use MCTS for tree search due to its balanced exploration and exploitation properties. Xie et al. [2024], Chen et al. [2024c] construct preference pairs after exploring the tree using MCTS. Xie et al. [2024] select nodes with the highest and lowest Q-values at the same level as the preference pair, while Chen et al. [2024c] selects child pairs with different Q-values sharing the same parent node.

Token-level Preference Optimization Recent research has explored token-level preference optimization to enable finer-grained learning. A central challenge lies in acquiring token-level preference pairs. Rafailov et al. [2024], Zhong et al. [2024] demonstrate that policy models trained via Direct Preference Optimization (DPO) can implicitly capture token-level reward signals, termed "implicit reward": $\beta \log \frac{\pi_{dpo}(y_t|x,y_{<t})}{\pi_{ref}(y_t|x,y_{<t})}$. This insight facilitates the development of token-level DPO algorithms. Yang et al. [2024b] further refine implicit rewards to enhance token-level optimization.

In a complementary approach, Lin et al. [2024] propose the cDPO algorithm, which annotates token-level importance from an alternative perspective. By fine-tuning two LLMs on correct and incorrect solutions, cDPO computes the probability difference between these models to determine the weight of each token in incorrect solutions. A lower difference score s_t indicates that the token bears greater responsibility for reasoning errors, enabling the identification of critical tokens for weighted optimization.

While DPO-based methods are widely adopted due to their simplicity, they exhibit limitations, as discussed in Section 4.1.5. Notably, as highlighted in the O1 blog [OpenAI, 2024b] and R1 report [DeepSeek-AI et al., 2025], achieving substantial improvements in complex reasoning capabilities may ultimately necessitate the integration of online RL techniques, underscoring the need for more advanced optimization frameworks.

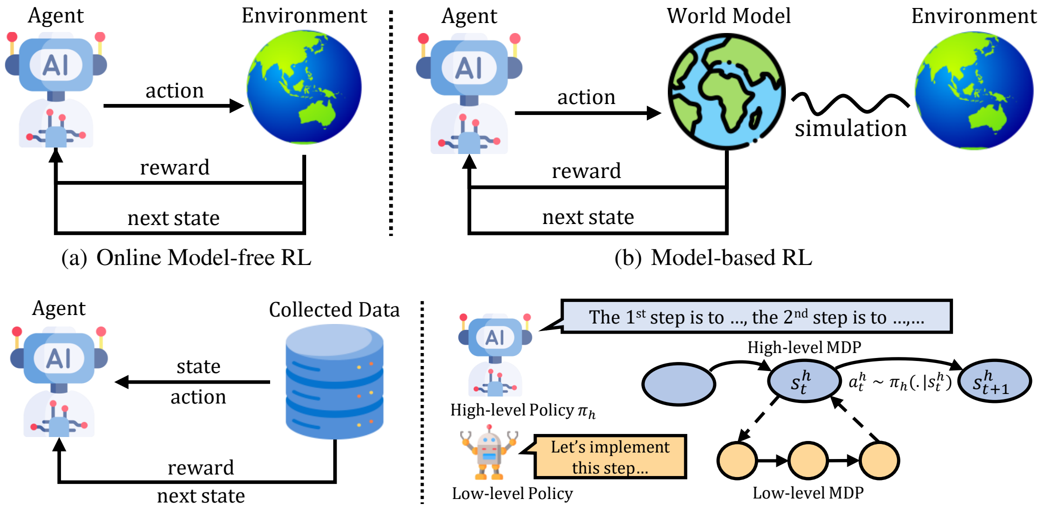
4.2.3 Reinforcement Learning

Model-free Online Reinforcement Learning

For tasks like mathematical reasoning, the environment dynamics are deterministic, as these tasks do not involve interaction with an external environment. After each action (generating a token or a reasoning step), the reasoning state is automatically updated. For instance, the context built from previous steps is augmented by the newly generated token to form the new context. Moreover, due to advancements in accelerating LLM inference, sampling reasoning solutions from LLMs has become both efficient and cost-effective. Consequently, addressing such problems generally necessitates only model-free online RL algorithms.

"Model-free" refers to the absence of explicit modeling of the environment, with direct interaction instead; "Online" indicates that training data generated by interaction between the current policy and the environment, rather than learning from a fixed corpus (i.e. "Offline" learning). The difference between Online and Offline learning affects distribution shifts in the reinforcement learning process.

Popular online RL methods for LLM training include REINFORCE [Sutton et al., 1999], PPO [Schulman et al., 2017], and GRPO [Shao et al., 2024]. Li et al. [2023d], Ahmadian et al. [2024] find that directly applying REINFORCE achieved good results without ORM and value models. ylfeng et al. [2024] improve LLM reasoning capabilities at the solution level using PPO, inspired by RLHF.



(a) Online Model-free RL

(b) Model-based RL

(c) Offline RL

(d) Hierarchical RL

Figure 7: A comparison of three RL paradigms: Online Model-based, Model-based, Offline RL, and Hierarchical RL

Zhang et al. [2024j] learn PRM and use it to guide PPO training at the step level. Zhong et al. [2024] leverages implicit rewards from DPO to further guide PPO training at the token level. Works such as deepseek-math [Shao et al., 2024], qwen-math [Yang et al., 2024a], and OpenR [Wang et al., 2024e] employ GRPO [Shao et al., 2024] for training, where the training process is guided by the PRM, significantly enhancing the LLM's multi-hop reasoning capabilities.

Despite the success of current model-free RL algorithms, as task complexity increases and reasoning tasks are extended to more real-world scenarios, relying solely on environments without interaction is insufficient. In such scenarios, more diversified RL algorithms are expected to play a crucial role in post-training LLM optimization.

Offline Reinforcement Learning

Offline RL trains the policy model using a static dataset rather than trajectory data collected through interaction between the policy model and the environment [Prudencio et al., 2022]. Due to the time and computational costs of training LLMs, batch training provides a significant advantage during large-scale fine-tuning. Therefore, many studies, particularly in academia, adopts offline training methods for LLMs.

Snell et al. [2022] improve the existing offline RL algorithm, IQL, and applies it to natural language generation tasks, proposing the ILQL algorithm. The most commonly used offline training method is DPO [Rafailov et al., 2023], which follows a process where large amounts of preference data are first collected, and then preference learning is conducted on this data. In addition to eliminating the need for reward modeling, this offline training method has facilitated the widespread application of DPO. [Wang et al., 2024b] derive a new offline RL method, OREO, based on Maximum Entropy RL [Haarnoja et al., 2017], addressing the limitation of DPO in relying solely on preference information and not incorporating actual reward values.

Despite the widespread use of offline RL, particularly DPO, in current research, the method has some notable drawbacks. The primary issue is that the data used in this approach is not sampled from the current policy model, but rather from previous, suboptimal policies. As the policy model improves during training, the divergence between the behavior policy and the target policy increases, which can severely impact the training effectiveness. By transitioning the offline training process to an online training strategy, this issue could potentially be mitigated [Chen et al., 2024a]. Another approach to utilizing offline RL is to initialize LLM [Yang et al., 2024c, He et al., 2024c], followed by further enhancement using online RL. Compared to online RL, offline RL allows for pre-prepared training signals, making it particularly advantageous for tasks with standard answers that are difficult to evaluate using reward models [Yang et al., 2024c].

Model-based Reinforcement Learning

For tasks involving interaction with external environments, such as dialogue systems and visual navigation, modeling the environment is crucial [Moerland et al., 2020]. A simulated environment, or world model [Zhu et al., 2024], can provide feedback, state transitions, and internal planning capabilities, significantly reducing interaction costs during both training and inference. Effective world models must possess sufficient task-specific knowledge to accurately simulate rewards and state transitions in response to policy model actions.

A prominent example is AlphaGo Zero [Silver et al., 2017], which models opponents and employs MCTS to simulate game states for policy learning. Similarly, Hao et al. [2023] demonstrate that LLMs can serve as world models for planning tasks and He et al. [2024c] apply LLMs to dialogue planning, simulating user interactions within an MCTS framework.

Despite these advancements, model-based RL for LLM-based complex reasoning remains underexplored, particularly in tasks like mathematical reasoning that lack external environment dynamics. However, as research progresses toward more intricate tasks, the integration of world models and model-based RL with LLMs is poised to become a focal point, offering promising avenues for enhancing reasoning capabilities.

Hierarchical Reinforcement Learning

Many reasoning tasks can be effectively modeled as Hierarchical MDPs, mirroring human cognitive processes. For instance, in mathematical reasoning, students typically do not reason word-by-word; instead, they first outline a sequence of reasoning steps and then generate specific content based on these steps. This process naturally decomposes into high-level and low-level MDPs: the high-level model generates abstract reasoning thoughts, while the low-level model produces the corresponding tokens during implementation.

Liu et al. [2024b] formalize reasoning tasks as hierarchical MDPs, where the high-level model selects a reasoning strategy (e.g., CoT [Wei et al., 2022], L2M [Zhou et al., 2022], or PoT [Chen et al., 2022]) before generating detailed reasoning steps. If the reasoning fails, the model iteratively selects a new strategy. While SMART [Liu et al., 2024b] optimizes the high-level MDP using policy gradient algorithms [Lee et al., 2024c], it does not optimize the low-level reasoning process. ReasonFlux [Yang et al., 2025a] constructs a series of thought templates first. It begins by performing high-level thought planning to generate a sequence of thoughts. Subsequently, each thought is instantiated within the context of the specific problem, forming a complete and coherent reasoning process. This structured approach decomposes complex tasks into manageable, high-level abstractions and their corresponding concrete instantiations. Similarly, Zhou et al. [2024] propose ArCHer, a hierarchical RL framework for LLMs: at the high level, it employs the value-based offline RL algorithm IQL [Kostrikov et al., 2021] to learn utterance-level Q- and V-networks, assessing response quality with outcome rewards. At the low level, ArCHer uses REINFORCE [Sutton et al., 1999] to optimize token-level MDPs, with rewards derived from the high-level advantage function.

By leveraging hierarchical learning, LLMs can establish coherence between abstract reasoning steps, moving beyond token-by-token recall to learn structured reasoning strategies. This approach enhances their ability to tackle more complex reasoning tasks effectively.

4.3 Evaluator Optimization

4.3.1 Training Data Construction

In this section, we first introduce the methodology for constructing data to optimize the evaluator, encompassing **outcome-level**, **step-level**, and **token-level** data construction.

Outcome-level Outcome-level rewards are relatively straightforward to construct. Early RLHF methods relied on manually annotated preference data to train reward models, but the high cost of manual annotation has spurred the development of automated labeling approaches.

The simplest automated method uses correct answer labels to classify solutions as correct or incorrect, forming preference pairs for training reward models based on DPO [Hosseini et al., 2024]. Alternatively, more powerful LLMs can be employed to evaluate reasoning correctness. For instance, Lee et al. [2024a] utilize a stronger LLM to score responses on a scale from 0 to 10, subsequently training a reward model on this annotated dataset. Beyond these methods, Mu et al. [2024] introduce rule-

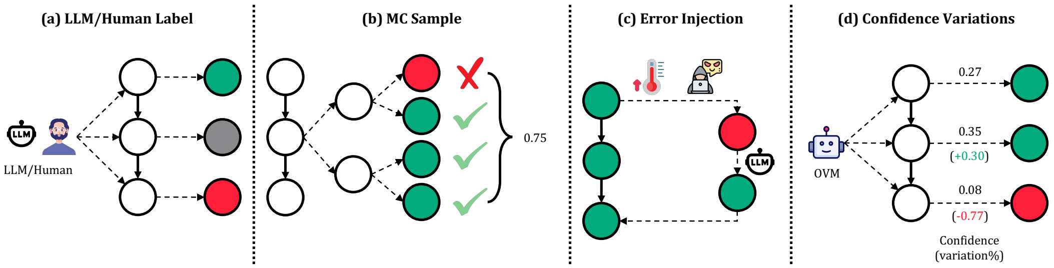


Figure 8: An illustration of four methods to construct step-level reward training signals.

based rewards, which decompose desired behaviors into specific rules and assign scores accordingly. These rule-based rewards are combined with traditional RLHF rewards and optimized through PPO to enhance model performance. Similarly, DeepSeek-AI et al. [2025] design a rule-based reward system for reasoning tasks, incorporating accuracy and format rewards to construct comprehensive reward signals for training.

These automated approaches not only reduce reliance on costly manual annotation but also provide scalable and efficient mechanisms for reward model training, advancing the practicality of RLHF in complex reasoning tasks.

Step-level To obtain step-level evaluation signals, OpenAI released the Process Reward Dataset PRM800K [Lightman et al., 2023]. However, given the continuous emergence of reasoning tasks and the high generalization requirements of evaluator, it is necessary to expand the training data for PRM. Manual annotation [Lightman et al., 2023] is fairly cost and unscalable, and LLM-as-a-Judge [Zheng et al., 2023] is easily implemented but suffers from instability and noise Ye et al. [2025]. Therefore, more automated labeling methods are effective. Based on different approaches, current automated labeling methods can be divided into three categories:

- The first type of methods estimates the correctness of each reasoning step. [Wang et al., 2024g,m, Jiao et al., 2024] use Monte Carlo (MC) sampling to estimate step rewards, where the success rate of N completions for step S_i serves as its reward. Luo et al. [2024] focus on identifying the first error using binary search combined with MC sampling, and improve the utilization of MC samples using MCTS. [Zhang et al., 2024f, Xia et al., 2024, Gao et al., 2024a] employ LLM-as-a-judge to directly evaluate step correctness. Zhang et al. [2025b] highlight that MC sampling may introduce significant noise and propose a consensus filtering mechanism, combining MC estimation with LLM validation to eliminate inconsistencies and improve data accuracy. Unlike the aforementioned methods, Chen et al. [2024g] decompose problems into subproblems and directly extracts ground-truth intermediate results from standard solutions, comparing model-generated sub-solutions with ground-truth results to evaluate step correctness.
- The second category of methods first predefines labels and then generates corresponding step content based on those labels. This primarily involves actively introducing errors into correct reasoning processes to construct datasets containing error steps. Yan et al. [2024] introduces errors by sampling at higher temperatures and then generates reflection and corresponding corrections referred to the correct solutions. Similarly, Xi et al. [2024] deliberately insert errors and prompts the model to generate reflections, creating high-quality correction data.
- The third thought evaluates step quality by measuring reliability changes during the reasoning process. The underlying assumption is that good reasoning steps increase the reasoning reliability, while poor steps decrease it. Lu et al. [2024a] label step correctness based on relative confidence changes, using an Outcome-Supervised Verifier to assess confidence differences between adjacent steps, reducing computational costs by avoiding extensive sampling.

Token-level To obtain finer-grained token-level reward signals, it is necessary to automate the evaluation of the importance of each token. Chen et al. [2024h] first train a generative reward model capable of rewriting the original solution. After training, the generated solution is input into the reward model, and the predicted probability of each token in the original solution is used as its reward. The rationale behind this approach is that incorrect token is likely to be modified, leading to reduced predicted probability, while correct token is predicted consistently, resulting in higher probability. Yoon et al. [2024] adopt a similar approach by leveraging a strong LLM to iteratively modify an incorrect solution y_r through three types of operations: addition, deletion, and replacement. The modified solution y_m is then compared to y_r , and token-level rewards are labeled based on the changes made to each token. Rafailov et al. [2024], Zhong et al. [2024] derive an implicit reward from the DPO framework, expressed as $\beta \log \frac{\pi_{dpo}(y_t|x, y_{<t})}{\pi_{ref}(y_t|x, y_{<t})}$. This implicit reward can be used for token-level reward labeling. Yang et al. [2024b] refine the process of obtaining token rewards based on implicit rewards as an initial scoring mechanism. For correct reasoning processes, the top $k\%$ of tokens are labeled with a reward of 1, while the rest are labeled with 0. For incorrect reasoning processes, the lowest $k\%$ of tokens are labeled with a reward of -1. OREA [Lyu et al., 2025] aligns the summarize of all token-level rewards with the outcome reward, enabling the learning of a token-level reward model.

4.3.2 Training Format

Point-wise When the evaluation outcome is a scalar value, the most straightforward approach is to use supervised learning to train an evaluation model. For example, Wang et al. [2024g,m] sample and complete reasoning steps, using the success probability of the completed paths as the score for each step, thereby training a step-level Process-Supervised Verifier (PSV). Lu et al. [2024a] first utilize ground truth to annotate each reasoning step and then trains an Outcome-Supervised Verifier (OSV) to estimate the probability of each step ultimately leading to a correct solution. Subsequently, by calculating the relative confidence changes between steps, step-level annotations were generated, which were then used to train the PSV.

Pair-wise Inspired by the Bradley-Terry (BT) model [Bradley and Terry, 1952], several studies have adopted preference learning to train evaluation models. These approaches collect preference pairs (x, y^+, y^-) and optimize the evaluation model $r(.,.)$ using the BT objective:

$$\max \mathbb{E}_{(x,y^+,y^-)\in D} \log(\sigma(r(x, y^+) - r(x, y^-))), \tag{22}$$

eliminating the need for precise scalar evaluations and relying solely on preference data for training. For instance, Yu et al. [2024b], Hosseini et al. [2024] utilize DPO to learn reward models from preference pairs, Liang et al. [2024] classify solutions sampled from multiple LLMs into preference pairs based on answer correctness, and train reward models using SimPO [Meng et al., 2024b].

To address the limitation of existing verifiers, which are often trained on binary-labeled reasoning paths and fail to capture relative differences between intermediate steps, He et al. [2024b] propose a tree-based method. This approach samples completions for each tree node and uses the proportion of completions leading to correct solutions as rewards. By comparing sibling node rewards, step-level preference pairs are collected, and a step-level ranking loss is employed to train the verifier.

Building on these advancements, Yuan et al. [2023b] introduce Reward-Weighted Preference Learning (RRHF), which simplifies preference learning by sampling multiple responses from diverse sources (e.g., the model itself, other LLMs, human experts) and ranking them based on human preferences or reward model scores. RRHF directly optimizes the conditional probability of responses using a ranking loss, enhancing the model's generative capabilities. These methods collectively advance the efficiency and effectiveness of preference-based evaluation and optimization.

Autoregressive Leveraging the intrinsic capabilities of large language models (LLMs) has emerged as a pivotal strategy to enhance the reliability and interpretability of evaluation frameworks. Existing approaches can be broadly categorized based on the form of feedback they employ: probability scores or verbal critiques.

For probability scores, several methods extract generation probabilities of specific tokens from natural language feedback as the basis for scoring. For instance, Zhang et al. [2024f] utilize LLMs to evaluate solutions by prompting them to answer the question, "Is the answer correct (Yes/No)?", and use the probability of generating "Yes" as the score. To achieve this, they directly optimize the LLM to predict the answer token. To further improve the interpretability and reliability of the evaluated scores, Zhang et al. [2024f], Ankner et al. [2024b], Gao et al. [2024a] integrate verbal chain-of-thought (CoT) reasoning to assist in answer token generation. They propose a two-stage training process: first generating interpretable CoT and then producing the answer token based on this reasoning, thereby offering a more transparent and robust evaluation framework. Additionally, Mahan et al. [2024]

introduce CoT-GenRM-STaR, which leverages incorrectly evaluated data and employs DPO to train a generative reward model, further advancing the field.

In contrast, verbal feedback provides richer contextual information, such as error locations and underlying causes, making it more effective for guiding subsequent self-correction and backtracking. Approaches to optimizing these capabilities can be broadly divided into BC and RL. BC represents the most straightforward method, with detailed data construction techniques discussed in Section 4.3.1. For example, Xi et al. [2024] construct training data by manually injecting noise into clean data and generating verbal critiques, followed by training the evaluator using BC. However, due to the inherent challenges of data construction, many works have turned to RL-based methods to learn self-evaluation capabilities. RL4F [Akyürek et al., 2023] and Retroformer [Yao et al., 2024b] formulate feedback generation as an RL task: the state consists of the current generated content and external environmental feedback, the action is the evaluator generating specific feedback based on the state, and the quality difference between the generated content before and after incorporating the feedback serves as the reward for the evaluator. Xie et al. [2025b] initialize the model using BC and further refine it through RL, demonstrating the complementary strengths of these approaches. State-of-the-art open-source studies like R1 [DeepSeek-AI et al., 2025] and Kimi k1.5 [Team et al., 2025] also employ verbal feedback for self-evaluation during reasoning, ultimately optimizing both step-by-step reasoning and self-evaluation capabilities under the guidance of rule-based rewards using RL.

4.4 Post-Processor Optimization

In the post-processing phase, the primary focus is on enhancing the model's ability to correct errors. Based on the optimization approach, these methods can be categorized into two types: Behavior Cloning and Reinforcement Learning.

Behavior Cloning

This category encompasses methodologies designed to enhance the model's self-correction capabilities and to train auxiliary models that facilitate the correction process. Regarding the former, studies like Zhang et al. [2024a], An et al. [2023], Yan et al. [2024], Paul et al. [2024], Gao et al. [2024c] utilize error sampling techniques, leveraging either more robust models or multiple self-generated samples to produce correction data. This data is subsequently employed for SFT to bolster the model's self-correction proficiency. Du et al. [2024] develops a progressive dataset to fine-tune the model, thereby augmenting its capacity for iterative solution refinement. Concerning the latter, approaches like Welleck et al. [2023], Zhang et al. [2024i], Wadhwa et al. [2024] involve training a specialized refiner model to rectify answers. Shridhar et al. [2024] introduce an asker model to ascertain the necessity of correction and to aid in the correction process. Additionally, Wang et al. [2024l] incorporate reflective insights into a codebook, facilitating the storage, retrieval, and application of historical insights to guide LLMs in problem-solving endeavors.

Reinforcement Learning

Kumar et al. [2024] identify two critical challenges encountered by SFT-based approaches in learning self-correction: distribution shift (where models can correct errors from a base model but struggle with self-generated errors) and behavior collapse (where models tend to optimize initial outputs while neglecting genuine correction behaviors). To mitigate these issues, they propose an on-policy multi-turn RL approach. However, Kumar et al. [2024] propose to generate only two consecutive solutions—an initial solution and a revised solution—and does not incorporate feedback from the reasoning phase. This limitation restricts its ability to fully exploit external guidance provided by feedback. To address this, Gehring et al. [2024] introduce a reinforcement learning algorithm that integrates external execution feedback, enabling the model to effectively utilize such feedback to enhance its self-improvement capabilities.

Notably, studies such as R1 [DeepSeek-AI et al., 2025], Kimi k1.5 [Team et al., 2025], and T1 [Hou et al., 2025] do not explicitly differentiate the reasoner, evaluator, and post-processor physically. Instead, guided by the same outcome reward, capacities of reasoning, self-evaluation, self-correction, and so on are optimized in the same action space simultaneously.

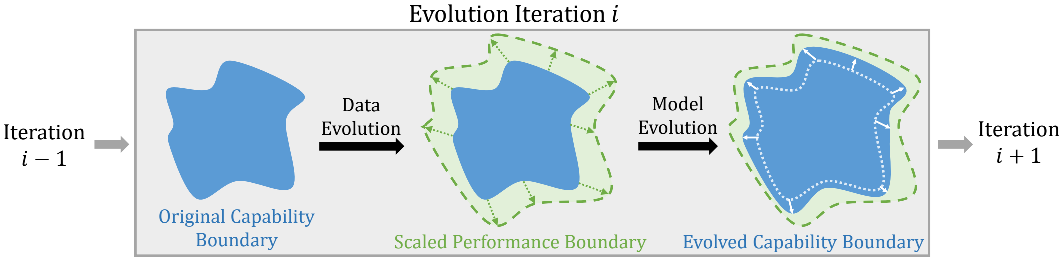

Evolution Iteration i
Iteration $i-1$
Data Evolution
Model Evolution
Iteration $i+1$
Original Capability Boundary
Scaled Performance Boundary
Evolved Capability Boundary

Figure 9: An intuitive understanding of self-evolution. Each iteration consists of data evolution and model evolution: the system first achieves a scaled performance boundary beyond its original capability boundary through data evolution (primarily via search), presented as higher-quality solutions serving as training data. Subsequently, the system expands its capability boundary by learning from these data through model evolution.

5 Self-evolution

In the Data Evolution section, we have explored how to leverage task evolution and CoT evolution to generate higher-quality training data. In the Model Evolution section, we have investigated methods to improve each module within the system. However, relying solely on either data or model evolution is insufficient to develop a satisfactory reasoning system. While data evolution enhances reasoning performance through inference-time compute without updating the model, its effectiveness is ultimately constrained by the model's inherent capabilities. Conversely, model evolution cannot achieve sustained improvement without high-quality training data.

Therefore, in this section, we focus on self-evolution, which integrates data and model evolution in a cyclical process to enable continuous system advancement. We illustrate an intuitive explanation of how self-evolution works in Figure 9. Self-evolution requires the reasoning system to autonomously generate data and enhance its capabilities without external human intervention. However, self-evolution still poses specific implementation challenges during implementation, such as ensuring continuous improvement and coordinating modules co-evolution. In the following, we first introduce the convergence properties of self-evolution and subsequently propose a scaling law of self-evolution. Next, we survey self-evolution works from evolutionary strategies and patterns. Finally, we reinterpret representative O1-like works from a self-evolution perspective.

5.1 Theory Behind Self-evolution

Self-evolution requires the system to leverage its own generated data to continuously enhance its performance without external intervention [Zelikman et al., 2022]. However, the theory behind this "self-driven" training process needs first investigation. To ensure the validity of this training approach, two key questions need to be addressed: (1) Does self-evolution of reasoning follow a Scaling Law? (2) What are the key factors enabling self-evolution to achieve continuous improvement?

[RQ1] Does self-evolution of reasoning follow a Scaling Law?

To answer this question, we first discuss whether self-evolution can converge when the task set is fixed. Singh et al. [2023] conduct an attempt from the perspective of Expectation Maximization (EM) [Moon, 1996]. Specifically, this work formulates the reasoning task as $p(z = \hat{z}|x)$, where x represents the question and $\hat{z}$ is the correct answer. Typically, the LLM generates a chain of reasoning y which assists in deriving the final answer. Therefore, y can be modeled as a latent variable. For convenience, we define the variable $O = 1$ to indicate a correct answer, i.e., $z = \hat{z}$. The final optimization objective is formulated as:

$$
\begin{aligned}
\max_{\phi} \log p_{\phi}(O=1|x) =& \max_{\phi} \log \mathbb{E}_{q(y|x)} \left[\frac{p_{\phi}(O=1,y|x)}{q(y|x)} \right] \\
\geq& \max_{\phi} \mathbb{E}_{q(y|x)} \log \left[\frac{p_{\phi}(O=1,y|x)}{q(y|x)} \right] \\
=& \max_{\phi} \mathbb{E}_{q(y|x)} \log \left[\frac{p_{\phi}(O=1,y|x)}{q(y|x)} \right] \\
=& \max_{\phi} -\mathbb{D}_{KL}[q(y|x)||p_{\phi}(O=1,y|x)] \\
=& \max_{\phi} \mathbb{E}_{q(y|x)}[p_{\phi}(O=1|x,y)] - \mathbb{D}_{KL}[q(y|x)||p_{\phi}(y|x)].
\end{aligned} \tag{23}
$$

For optimization problems involving latent variables, the EM algorithm is often employed.

In the E-Step, we fix $p(O=1,y|x)$, and maximize the objective by optimizing $\mathbb{D}_{KL}(q(y|x)||p(O=1,y|x))$. This optimization ultimately leads to $q(y|x)=p(O=1,y|x)=p(O=1|x,y)p(y|x)$. The empirical interpretation of this result is: first, generate the reasoning process y, and then estimate the correctness of the answer derived from y using $p(O=1|x,y)$. Thus, the E-Step can be understood as modeling the data generation process, including both the generation of data and its subsequent evaluation.

While in the M-Step, we fix $q(y|x)$, and maximize the objective by optimizing $\mathbb{D}_{KL}(q(y|x)||p(y|x))$, ultimately resulting in:

$$
\min_{\phi} \mathbb{D}_{KL}(q(y|x)||p_{\phi}(y|x)) = \max_{\phi} q(y|x) \log p_{\phi}(y|x). \tag{24}
$$

Therefore, the M-Step aims to train the reasoning model $p_{\phi}(y|x)$ using the generated data, leveraging data of varying quality weightedly. In other words, the M-Step learns the process of model evolution. Considering the convergence properties of the EM algorithm, we have reason to believe that this iterative training process, which continuously improves both the data and the model, will converge.

However, the conclusion that self-evolution can converge is contingent on the assumption of a fixed task set. Whether convergence remains achievable under task evolution is still an open question. Intuitively, expanding the task set can enhance data coverage, thereby further improving the generalization of the reasoning system. However, the existence of a scaling law for self-evolution hinges on whether task evolution itself follows a scaling law—that is, whether task evolution has an inherent ceiling or limit.

Of course, the discussion above is merely a hypothesis, and definitive conclusions will require further experimental validation.

[RQ2] What are the key factors of self-evolution to achieve continuous improvement?

Zeng et al. [2024a] find that in existing self-evolution methods, the performance gain diminishes obviously after 3–4 epoches of training, with even performance degradation observed. To investigate the bottleneck affecting model performance, Zeng et al. [2024a] analyze the diversity of reasoning trajectories searched by the model. The results show that, as the self-evolution training progresses, the diversity of the reasoning trajectories explored decreases significantly. This is because the trajectories that are evaluated more highly are increasingly likely to be sampled again, leading the reasoning model to converge on narrow reasoning patterns. While this reduction in exploration helps the model focus on generating higher-quality reasoning processes, it also means that the model fails to explore new knowledge, preventing further improvements in its generalization ability.

Based on these observations, Zeng et al. [2024a] propose a new self-evolution framework, B-STAR. First, they design the Balance Score metric to measure exploration ability during the training process. Then, they introduce a dynamic training strategy based on sampling temperature and reward filtering thresholds to relieve the issue of exploration reduction during the self-evolution process. The conclusion of B-STAR highlights that a key factor influencing the performance of self-evolution is the diversity of reasoning trajectories encountered by the LLM during training. When this diversity decreases, the evolution also diminishes.

From a higher-level perspective, B-STAR enhances the generalization of the reasoning system by intensifying exploration efforts for CoT evolution. Following this principle of improving reasoning generalization, additional strategies can be employed to further enhance system generalization: 1) increasing task diversity and difficulty could directly enhance reasoning trajectory diversity and improve system generalization [Li et al., 2024a]; 2) enhancing the system's self-evaluation and post-processing abilities can significantly improve the system's robustness when encountering errors; and 3) improving the consistency and generalization of the reasoner's logic can reduce the frequency of errors.

5.2 Self-Evolution Strategies

In the Preliminaries section, we have defined four key modules in the reasoning system and described their respective functions and interrelations. Thus, the reasoning system can be viewed as a multi-agent system, where any evolution of the four agents should strengthen the overall system's performance in theory. Moreover, the combination optimization of multiple agents is expected to provide more substantial performance gains. We will first summarize the three multi-agent training strategies that can be applied within the reasoning system.

5.2.1 Independent Evolution

In early self-evolution systems, thoughts for improving internal modules are relatively simple: often only one module was improved, or the interdependencies between modules are inapparent. For example, Zelikman et al. [2022], Gulcehre et al. [2023] use golden answers to evaluate the correctness of the solutions and select the correct solutions to enhance the reasoning capabilities of Reasoner. Hosseini et al. [2024] also classify solutions based on golden answers, using DPO to train the evaluator (verifier). Although the preference data for training the verifier is generated by the reasoner, the verifier fails in providing assistance to the training of the reasoner. Madaan et al. [2023b] iteratively correct solutions to continually improve their quality, thus achieving logical self-evolution. However, existing work suggests that relying solely on In-Context Learning is insufficient for LLM self-correction. Wang et al. [2023d] iteratively improve the refinement of the post-processor during inference. In this process, the reasoner is only used to provide the initial solution, offering no further assistance to the evolution of post-processor.

Independent evolution is simple in design and easy to implement, but provides limited performance gains. By individually studying the evolution mechanisms of each module, we can lay the foundation for future research that jointly improves multiple agents.

5.2.2 Collaborative Evolution

When it comes to the joint evolution of multiple modules, a common approach is to leverage cooperation between the modules to improve overall system performance.

Jiang et al. [2024a] use the reasoner to generate both correct and incorrect solutions, constructing training data for the reward model. On the other hand, the reward model is also used to filter solutions for training the reasoner. Wang et al. [2024e] adopt an RL-based thought to train the entire reasoning system. In policy iteration, they used the PRM (Evaluator) to provide reward signals to guide the optimization of the policy model (Reasoner), while in value iteration, they used the data collected by the Reasoner to train the Evaluator.

Based on the above experience, more cooperation strategies need to be further explored in reasoning systems.

5.2.3 Adversarial Evolution

Besides cooperation, another strategy for joint learning is adversarial training. The effectiveness of adversarial thought has already been demonstrated in existing works, such as GANs [Goodfellow et al., 2014], where the adversarial training between the generator and discriminator marked a milestone in the development of generative models. In the reasoning system, there is a natural adversarial relationship between the task creator and the reasoner. The task creator generates more novel and challenging tasks to test the reasoner, while the reasoner strives to solve these new tasks. Based on this adversarial strategy, Ye et al. [2024] propose a joint training approach for the reasoner and task creator. The reasoner receives new tasks from the task creator and uses ReST [Gulcehre et al., 2023] to improve itself, while the task creator first estimates the uncertainty of the reasoner's ability to solve each problem and selects tasks with higher uncertainty as seeds to generate more diverse questions based on Eval Instruct. This adversarial learning process enables the mutual evolution of the task creator and reasoner.

The adversarial relationship is harder to identify and learn compared to cooperation, but offers the likelihood to mitigate the issue of local optima during iterative training. While cooperation may exacerbate this issue, as agents tend to cater to each other. By strategically combining cooperation and adversarial training, the system may achieve more significant improvements.

5.3 Self-Evolution Patterns

After discussing the overall strategies, we further explore various self-evolution modes from the perspective of modules.

5.3.1 Only Reasoner

Most methods focus on directly improving Reasoner to enhance system performance. These approaches are relatively simple, with the main differences in the training data and the methods used to train.

For training the Reasoner, Gulcehre et al. [2023], Min et al. [2024], Zelikman et al. [2022] perform SFT on correct reasoning processes. Chen et al. [2024b], Xie et al. [2024], Wang et al. [2024j,k] use MCTS to search preference data and optimize it to train the Reasoner. Gulcehre et al. [2023] use rewards generated by a reward model to guide the Reasoner's learning via RL.

For the data acquisition, Singh et al. [2023], Min et al. [2024], Pang et al. [2024] directly use golden answers to select correct data for the next round of Reasoner training. Zelikman et al. [2022] aim to increase positive data by using correct answers as hints to re-answer originally incorrect questions. Peng et al. [2024] argue that directly providing answers as hints in STaR [Zelikman et al., 2022] could help the Reasoner to find shortcuts. To this end, they propose providing answers only during abstract reasoning steps, not the generation of specific solutions. Without answer labels, Huang et al. [2022], Li et al. [2024c] select answers based on consistency to filter data.

Additionally, Aksitov et al. [2023], Dong et al. [2023] use an extra reward model to score and rank reasoning trajectories, selecting high-quality ones for training the Reasoner. While Song et al. [2024] filter out low-quality trajectories based on rewards from the environment.

5.3.2 Reasoner + Evaluator

In the self-evolution process, the Evaluator is tasked with assessing the reasoning process, which introduces the challenge of ensuring its generalization capability. This issue becomes more pronounced as the self-evolved progresses: as training continues, the problems and reasoning processes generated by the Reasoner may gradually diverge from the data distribution used to train the Evaluator. Therefore, enhancing the Evaluator's generalization ability is a critical aspect of the self-evolution framework.

[Yuan et al., 2024d, Wang et al., 2024c] train the Reasoner using correct reasoning processes it generates, while also using both correct and incorrect reasoning processes to train the reward model, enabling the Reasoner to assist in improving the Evaluator. Building on this work, Jiang et al. [2024a] incorporate the reward model into the solution selection process, adopting an active learning approach to prioritize difficult cases. Zhang et al. [2024c], Guan et al. [2025] propose an iterative training framework for the Reasoner and Evaluator: the Reasoner uses MCTS to obtain action value estimates at the step level, which are then used to train a PPM (Evaluator); the trained PPM further assists in identifying high-quality reasoning trajectories during MCTS, which in turn trains the Reasoner. Zhang et al. [2024j], Wang et al. [2024e] adopt RL to train the Reasoner and use a process reward model to score reasoning steps, guiding the Reasoner's learning, thus enabling mutual assistance and co-evolution between the Reasoner and Evaluator. Chen et al. [2024b], Zhang et al. [2024c] do not directly use the Evaluator to assist the Reasoner during training, but instead use the reward model to guide the Reasoner's MCTS search process, improving the proportion of correct reasoning processes generated by the Reasoner. Cheng et al. [2024], Chen et al. [2024i] employ adversarial training for the Reasoner and Evaluator. The main idea is to train the Evaluator to distinguish whether responses match golden responses, while simultaneously improving the Reasoner's ability to generate responses that confuse the Evaluator. However, this approach requires further exploration in reasoning tasks, as such tasks only require correct answers, not necessarily following a specific reasoning process. In fact, the diversity of reasoning better reflects a system's reasoning capability, so aligning solely with golden responses is not the optimal goal.

5.3.3 Reasoner + Post-Processor

On the other hand, the stronger the Reasoner is, the less pressure there is on the Post-Processor, especially for the Refiner. Some works consider improving these two modules simultaneously. Dou et al. [2024] first use the Reasoner to generate the initial solution, then refine it. Wang et al. [2023d] employ multi-turn refinements until the solution is correct or the maximum number of turns is reached. During training, Dou et al. [2024] collect refined solutions and trains the Refiner's modification ability via SFT, while also using correct solutions to SFT the Reasoner for improving its reasoning skills. Wang et al. [2023d] adopt reinforcement learning, where the final correct solution is used as the target to train the Reasoner, and the refinement process is modeled as an MDP, with RL used to train the Refiner's modifications in each round. Ultimately, both the Reasoner and Refiner improve concurrently.

5.3.4 Reasoner + Task Creator

The diversity and complexity of tasks significantly impact the efficacy of self-evolution. Empirically, if learning is confined to a static set of tasks, the Reasoner tends to overfit, thereby diminishing its capability to handle out-of-distribution (OOD) tasks. Therefore, the evolution of the Task Creator during this process is equally vital. Ye et al. [2024] introduce an adversarial training method for both the Reasoner and Task Creator: the Task Creator endeavors to devise increasingly challenging tasks, while the Reasoner aims to enhance its reasoning skills to address these new challenges. This approach, discussed in Section 5.2.3, facilitates mutual evolution through adversarial learning, ensuring continuous improvement.

5.3.5 Reasoner + Evaluator + Post-Processor

Compared to the previously discussed studies that evolve only one or two modules, the simultaneous evolution of the Reasoner, Evaluator, and Post-Processor can theoretically yield greater improvements. Recent works such as R1 [DeepSeek-AI et al., 2025], T1 [Hou et al., 2025], and Kimi-k1.5 [Team et al., 2025], which focus on learning Long CoT [Xu et al., 2025], exemplify this approach. These works generate Long CoTs that incorporate self-evaluation, self-reflection, and self-correction operations, and then optimize the LLM using online RL guided by outcome rewards.

First, we would like to emphasize that learning reasoning capabilities based on online RL inherently aligns with the self-evolution framework. Unlike traditional approaches that rely on fixed training datasets, online RL methods drive the LLM to interact with the environment, generating trajectories and rewards to guide the LLM training. As training progresses, the LLM discovers more diverse and higher-quality trajectories in subsequent exploration epochs. During this iterative process of evolution, the LLM can also effectively mitigate the issue of failing to improve continuously by controlling the balance between exploration and exploitation.

Moreover, R1 and similar works do not explicitly distinguish the modules of evaluator and post-processor or explicitly optimize evaluation and correction capacities like rStar-Math [Guan et al., 2025]. However, they unify the optimization of reasoning, verification, evaluation, reflection, and error correction within the same action space under the guidance of the same outcome rewards. In other words, during the learning process in works like R1, the LLM's abilities in evaluation, reflection, and self-correction are simultaneously optimized. Therefore, we argue that R1 and similar works effectively achieve the co-evolution of reasoning, evaluation, and post-processing capacities. Based on this conclusion, we can also understand, from the perspective of self-evolution, why R1 and similar works surpass earlier approaches that focused on improving only one module or two modules.

5.4 Challenges and Future Directions

5.4.1 More Promising Self-evolution Patterns

We've discussed five common self-evolution patterns in the above content, but there are $2^4 - 1 = 15$ possible optimization combinations for these four modules theoretically. By exploring different module combinations and strategies like cooperation and adversarial learning, more effective self-evolution frameworks can be achieved. Ideally, simultaneous enhancement of all modules would lead to sustained and significant improvements.

5.4.2 System Generalization

Self-evolution enhances system performance through iterative training. The key to sustained evolution lies in preventing overfitting and ensuring generalization during this process. First, task generalization is crucial; synthesizing more diverse and complex tasks ensures broader coverage, which is fundamental to addressing generalization issues [Yu et al., 2024a]. Second, the generalization ability of the reasoner, evaluator, and post-processor is vital. B-StAR [Zeng et al., 2024a] shows that enhancing the reasoner's exploration reduces overfitting. The post-processor also plays a key role in diversifying solutions. Moreover, reward hacking highlights that current evaluators may overfit to the reasoner and exploit reward shortcuts. In summary, the generalization of the reasoning system is crucial for continuous enhancement in the self-evolution framework.

6 Reinterpretation of Representative O1-like Studies

Building on the discussion of the self-evolution technical framework, this section seeks to reinterpret existing representative O1-like works from the lens of self-evolution.

Marco-O1

Marco-o1 [Zhao et al., 2024a] generates a dataset using MCTS search and performs SFT on this dataset. While Marco-o1 does not incorporate iterative training, the use of MCTS for data collection aligns with the concept of data evolution, and its SFT process represents model evolution. However, the absence of further iterations restricts the overall improvement in reasoning performance.

O1 Journey

O1 Journey [Qin et al., 2024, Huang et al., 2024] introduces the concept of Journey Learning, which explores reasoning processes involving self-reflection, self-correction, and backtracking, corresponding to the Long CoT discussed in Section 3.2.3. The generated Long CoTs are classified into positive and negative samples based on answer correctness, with DPO used for optimization, reflecting model evolution. Although it does not incorporate self-evolution, its strong performance is driven by the deep learning of implicit trial-and-error search capabilities.

Slow Thinking with LLMs

***Part 1*:** Part 1 of Slow Thinking [Jiang et al., 2024a] with LLMs involves iterative training across two stages. Initially, the reasoner and evaluator generate a series of solutions and corresponding scores via MCTS-based inference-time compute. Subsequently, DPO is applied to optimize both the reasoner and evaluator. The MCTS-based inference-time compute represents explicit tree search in data evolution, while DPO optimization aligns with model evolution. With the joint evolution of both the reasoner and evaluator, this work can be classified into the "reasoner + evaluator" self-evolution pattern.

***Part 2*:** Building on long-form thought capabilities distilled from QwQ [Team, 2024b] and DeepSeek [DeepSeek-AI et al., 2025], Part 2 of Slow Thinking [Min et al., 2024] with LLMs performs self-evolution through a cycle of exploration and learning. The generation of Long CoTs via Long-form Thought reflects implicit trial-and-error search in data evolution, followed by SFT or DPO to evolve the reasoner. This iterative process facilitates the self-evolution of the whole system.

rStar-Math

rStar-Math [Guan et al., 2025] is a representative example of self-evolution for reasoning capabilities, consisting of three training rounds: 1) The first round employs terminal-guided MCTS to collect accurate data for SFT on the reasoner. 2) The second round trains the PRM (evaluator) using the high-quality data gathered through MCTS. 3) The third round utilizes PRM-guided MCTS to collect additional high-quality data and retrains both the reasoner and the PRM. Each round integrates both data evolution and model evolution. Unlike prior works, rStar-Math focuses on evolving specific capabilities on each iteration, leading to overall performance improvement across multiple cycles.

OpenR/O1-Coder

Both OpenR [Wang et al., 2024e] and O1-Coder [Zhang et al., 2024j] utilize traditional RL approaches to enhance LLM reasoning capabilities. They simultaneously train the policy model (reasoner) and the PRM (evaluator). Specifically, the policy model explores new solutions via tree search, e.g., beam search and MCTS. And the PRM guides the training process, aligning with data evolution and model evolution, respectively. Through continuous RL-based exploration and learning, both modules achieve self-evolution.

DeepSeek R1/Kimi k1.5

R1 [DeepSeek-AI et al., 2025], and Kimi k1.5 [Team et al., 2025] represent the current state-of-the-art open-source reasoning models, achieving performance that closely rivals or even surpasses that of O1 [OpenAI, 2024b]. Not only do these models demonstrate exceptional results, but they also conduct a fundamentally similar core algorithm. These works leverage online RL for training and rely solely on ORM to guide optimization, which encourages exploration and facilitates the emergence of Long CoT generation capabilities.

Furthermore, the RL-based training paradigm employed by these models aligns with the principles of the self-evolution paradigm. Specifically, the RL agent explores new trajectories, corresponding to the data evolution phase; and is trained under the guidance of rewards, corresponding to the model evolution phase. Through iterative cycles of self-exploration and training, the system achieves self-evolution. Notably, works like R1 advance not only step-by-step reasoning but also self-evaluation and self-correction capabilities during self-evolution, which aligns with the Reasoner + Evaluator + Post-Processor pattern defined in Section 5.3.5. This holistic evolution of three components explains why R1 and Kimi k1.5 significantly outperform previous systems that evolved only one or two modules.

7 Future Chanllenges and Directions

How can we further enhance the complex reasoning capabilities of LLMs within the self-evolution framework?

Although models like O1 and R1 exhibit impressive reasoning capabilities, there is still significant room for improvement, including enhancing reasoning abilities and increasing token efficiency. Continuous training is essential, but it should focus on addressing key challenges. In future research, several critical issues remain to be resolved, which we summarize as follows:

- **How can we further enhance task diversity?** More challenging tasks are the most effective way to improve system generalization. For instance, Min et al. [2024] observe that models tend to converge after just a few turns of iterative training due to the sparsity of the task pool. To sustain self-evolution, increasing task diversity is crucial. While methods like R1 jointly evolve step-by-step reasoning, self-evaluation, and self-correction abilities, they do not incorporate task evolution. Effective task evolution could potentially lead to more significant and sustained improvements in R1. Current methods remain relatively simplistic, and further research is needed to generate more diverse, complex, and effective tasks.
- **How can we develop more fine-grained reward modeling?** Works like R1 have shown that satisfactory reasoning capabilities can be achieved using ORM alone, with R1 detailing its unsuccessful attempts with MCTS+PRM. This has fueled skepticism about the practicality of PRM. Compared to learnable PRMs, the rule-based ORM employed by R1 offers clear advantages in generalization and mitigating reward hacking. However, this kind of ORM fails to provide fine-grained rewards during optimization. Analysis has revealed that models like R1 tend to overthink simple problems while underthinking more complex ones [Chen et al., 2024f, Wang et al., 2025a], which could motivate research into PRM, which could offer process signals to guide efficient step-by-step reasoning. However, challenges such as the weak generalizability of PRM, continuous updates, and reward hacking remain significant obstacles to its further development. R1 implements PRM through self-evaluation and uses an ORM to optimize step-by-step reasoning, self-evaluation, and self-correction capacities simultaneously. However, they do not specifically optimize self-evaluation. The sustained and effective enhancement of self-evaluation remains further research.
- **How can we balance efficiency and effectiveness to identify the optimal CoT evolution?** Explicit tree search for Short CoT offers advantages in efficiency but lacks generalization

ability of Long CoT. While trial-and-error search mimics human reasoning, it suffers from inefficiencies such as overthinking and underthinking. A key research challenge is how to leverage the strengths of both search types during the inference-time computation phase. One approach could be to enhance LLMs' self-evaluation and self-correction capabilities to alleviate overthinking and underthinking. Another potential direction is to integrate explicit tree search principles with trial-and-error during inference, thereby expanding the performance of sequence reasoning in R1.

How can self-evolved reasoning be applied to embodied intelligence scenarios?

This survey focuses on self-evolution for complex reasoning tasks in the text modality. However, future AI systems will inevitably need to interact with the real world [Wang et al., 2024d], where numerous scenarios require reasoning across multimodal data [Xiang et al., 2024, Yao et al., 2024a, Wu et al., 2025a]. To achieve this goal, several challenges must be addressed. First, a comprehensive understanding of multimodal data is essential as the foundation for multimodal reasoning. Second, the format of CoT must be redefined, such as considering whether CoT should include tokens composed of multimodal data [Li et al., 2025]. Additionally, many reasoning tasks in multimodal scenarios (e.g., embodied intelligence) face challenges such as high costs of environmental interaction and limited data resources during training [He et al., 2024a].

8 Conclusion

This survey provides a systematic review of existing research on complex reasoning with LLMs from the perspective of self-evolution. We first examine relevant techniques through the lenses of data evolution and model evolution, laying the foundation for self-evolution. We then shift focus to self-evolution itself, analyzing how existing self-evolved studies by exploring the evolutionary relationships among system modules. Additionally, we further analyze and summarize existing O1-like open-source studies, finding that they can all be explained using our self-evolution framework. Ultimately, we hope this survey inspires further research to advance LLM-based complex reasoning.

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL `https://aclanthology.org/2024.acl-long.662/`.

Renat Aksitov, Sobhan Miryoosefi, Zong xiao Li, Daliang Li, Sheila Babayan, Kavya Kopparapu, Zachary Fisher, Ruiqi Guo, Sushant Prakash, Pranesh Srinivasan, Manzil Zaheer, Felix X. Yu, and Sanjiv Kumar. Rest meets react: Self-improvement for multi-step reasoning llm agent. *ArXiv*, abs/2312.10003, 2023. URL `https://arxiv.org/pdf/2312.10003`.

Afra Feyza Akyürek, Ekin Akyürek, Aman Madaan, A. Kalyan, Peter Clark, Derry Tanti Wijaya, and Niket Tandon. Rl4f: Generating natural language feedback with reinforcement learning for repairing model outputs. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL `https://aclanthology.org/2023.acl-long.427.pdf`.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. Learning from mistakes makes LLM better reasoner. *CoRR*, abs/2310.20689, 2023. doi: 10.48550/ARXIV.2310.20689. URL `https://doi.org/10.48550/arXiv.2310.20689`.

Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Joshua Tobin, P. Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Neural Information Processing Systems*, 2017. URL `https://arxiv.org/pdf/1707.01495`.

Zachary Ankner, Cody Blakeney, Kartik K. Sreenivasan, Max Marion, Matthew L. Leavitt, and Mansheej Paul. Perplexed by perplexity: Perplexity-based data pruning with small reference models. *ArXiv*, abs/2405.20541, 2024a. URL `https://arxiv.org/pdf/2405.20541`.

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D. Chang, and Prithviraj Ammanabrolu. Critique-out-loud reward models. *CoRR*, abs/2408.11791, 2024b. doi: 10.48550/ARXIV.2408.11791. URL https://doi.org/10.48550/arXiv.2408.11791.

Mohammad Gheshlaghi Azar, Mark Rowland, Bilal Piot, Daniel Guo, Daniele Calandriello, Michal Valko, and Rémi Munos. A general theoretical paradigm to understand learning from human preferences. *ArXiv*, abs/2310.12036, 2023. URL https://arxiv.org/pdf/2310.12036.

Ralph Allan Bradley and Milton E. Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39:324, 1952. URL https://api.semanticscholar.org/CorpusID:125209808.

Cameron Browne, Edward Jack Powley, Daniel Whitehouse, Simon M. M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez Liebana, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43, 2012. URL https://ieeexplore.ieee.org/document/6145622.

Changyu Chen, Zi-Yan Liu, Chao Du, Tianyu Pang, Qian Liu, Arunesh Sinha, Pradeep Varakantham, and Min Lin. Bootstrapping language models with dpo implicit rewards. *ArXiv*, abs/2406.09760, 2024a. URL https://arxiv.org/pdf/2406.09760.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Alphamath almost zero: process supervision without process. *ArXiv*, abs/2405.03553, 2024b. URL https://arxiv.org/pdf/2405.03553.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. Step-level value preference optimization for mathematical reasoning. In *Conference on Empirical Methods in Natural Language Processing*, 2024c. URL https://arxiv.org/pdf/2406.10858.

Huayu Chen, Guande He, Lifan Yuan, Hang Su, and Jun Zhu. Noise contrastive alignment of language models with explicit rewards. *ArXiv*, abs/2402.05369, 2024d. URL https://arxiv.org/pdf/2402.05369.

Justin Chih-Yao Chen, Zifeng Wang, Hamid Palangi, Rujun Han, Sayna Ebrahimi, Long T. Le, Vincent Perot, Swaroop Mishra, Mohit Bansal, Chen-Yu Lee, and Tomas Pfister. Reverse thinking makes llms stronger reasoners. *ArXiv*, abs/2411.19865, 2024e. URL https://arxiv.org/pdf/2411.19865.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 2022. URL https://arxiv.org/pdf/2211.12588.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do not think that much for 2+3=? on the overthinking of o1-like llms. *ArXiv*, abs/2412.21187, 2024f. URL https://arxiv.org/pdf/2412.21187.

Xinyun Chen, Renat Aksitov, Uri Alon, Jie Ren, Kefan Xiao, Pengcheng Yin, Sushant Prakash, Charles Sutton, Xuezhi Wang, and Denny Zhou. Universal self-consistency for large language model generation. *CoRR*, abs/2311.17311, 2023a. doi: 10.48550/ARXIV.2311.17311. URL https://doi.org/10.48550/arXiv.2311.17311.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *ArXiv*, abs/2304.05128, 2023b. URL https://doi.org/10.48550/arXiv.2304.05128.

Zhaorun Chen, Zhaorun Chen, Zhuokai Zhao, Zhihong Zhu, Ruiqi Zhang, Xiang Li, Bhiksha Raj, and Huaxiu Yao. Autoprm: Automating procedural supervision for multi-step reasoning via controllable question decomposition. *ArXiv*, abs/2402.11452, 2024g. URL https://aclanthology.org/2024.naacl-long.73/.

Zhipeng Chen, Kun Zhou, Wayne Xin Zhao, Junchen Wan, Fuzheng Zhang, Di Zhang, and Ji-Rong Wen. Improving large language models via fine-grained reinforcement learning with minimum editing constraint. In *Annual Meeting of the Association for Computational Linguistics*, 2024h. URL https://arxiv.org/pdf/2401.06081.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning converts weak language models to strong language models. *ArXiv*, abs/2401.01335, 2024i. URL https://arxiv.org/pdf/2401.01335.

Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, and Nan Du. Self-playing adversarial language game enhances llm reasoning. *ArXiv*, abs/2404.10642, 2024. URL https://arxiv.org/pdf/2404.10642.

Sayak Ray Chowdhury, Anush Kini, and Nagarajan Natarajan. Provably robust dpo: Aligning language models with noisy feedback. *ArXiv*, abs/2403.00409, 2024. URL https://arxiv.org/pdf/2403.00409.

Zheng Chu, Jingchang Chen, Qianglong Chen, Weijiang Yu, Tao He, Haotian Wang, Weihua Peng, Ming Liu, Bing Qin, and Ting Liu. Navigate through enigmatic labyrinth a survey of chain of thought reasoning: Advances, frontiers and future. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:263153015.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168, 2021. URL https://arxiv.org/abs/2110.14168.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, Hao Peng, Yu Cheng, Zhiyuan Liu, Maosong Sun, Bowen Zhou, and Ning Ding. Process reinforcement through implicit rewards. 2025. URL https://arxiv.org/pdf/2502.01456.

DeepSeek-AI, Daya Guo, Dejian Yang, Haowei Zhang, Jun-Mei Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiaoling Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bing-Li Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, Damai Dai, Deli Chen, Dong-Li Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Han Bao, Hanwei Xu, Haocheng Wang, Honghui Ding, Huajian Xin, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jiawei Wang, Jingchang Chen, Jingyang Yuan, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shao-Kang Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wanjia Zhao, Wen Liu, Wenfeng Liang, Wenjun Gao, Wen-Xia Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyu Jin, Xi-Cheng Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yi Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yu-Jing Zou, Yujia He, Yunfan Xiong, Yu-Wei Luo, Yu mei You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yao Li, Yi Zheng, Yuchen Zhu, Yunxiang Ma, Ying Tang, Yukun Zha, Yuting Yan, Zehui Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhen guo Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zi-An Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. 2025. URL https://arxiv.org/pdf/2501.12948.

Hanze Dong, Wei Xiong, Deepanshu Goyal, Rui Pan, Shizhe Diao, Jipeng Zhang, Kashun Shum, and T. Zhang. Raft: Reward ranked finetuning for generative foundation model alignment. *ArXiv*, abs/2304.06767, 2023. URL https://arxiv.org/pdf/2304.06767.

Zi-Yi Dou, Cheng-Fu Yang, Xueqing Wu, Kai-Wei Chang, and Nanyun Peng. Re-rest: Reflection-reinforced self-training for language agents. In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL https://aclanthology.org/2024.emnlp-main.861.pdf.

Chengyu Du, Jinyi Han, Yizhou Ying, Aili Chen, Qianyu He, Haokun Zhao, Sirui Xia, Haoran Guo, Jiaqing Liang, Zulong Chen, Liangyue Li, and Yanghua Xiao. Think thrice before you act: Progressive thought refinement in large language models. *CoRR*, abs/2410.13413, 2024. doi: 10.48550/ARXIV.2410.13413. URL https://doi.org/10.48550/arXiv.2410.13413.

Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. Successive prompting for decomposing complex questions. In *Conference on Empirical Methods in Natural Language Processing*, 2022. URL https://aclanthology.org/2022.emnlp-main.81.pdf.

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

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. *ArXiv*, abs/2309.17179, 2023. URL https://arxiv.org/pdf/2309.17179.

Thomas Palmeira Ferraz, Kartik Mehta, Yu-Hsiang Lin, Haw-Shiuan Chang, Shereen Oraby, Sijia Liu, Vivek Subramanian, Tagyoung Chung, Mohit Bansal, and Nanyun Peng. LLM self-correction with decrim: Decompose, critique, and refine for enhanced following of instructions with multiple constraints. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 7773–7812. Association for Computational Linguistics, 2024. URL https://aclanthology.org/2024.findings-emnlp.458.

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. Stream of search (sos): Learning to search in language. *ArXiv*, abs/2404.03683, 2024. URL https://arxiv.org/pdf/2404.03683.

Bofei Gao, Zefan Cai, Runxin Xu, Peiyi Wang, Ce Zheng, Runji Lin, Keming Lu, Junyang Lin, Chang Zhou, Tianyu Liu, and Baobao Chang. Llm critics help catch bugs in mathematics: Towards a better mathematical verifier with natural language feedback. *ArXiv*, abs/2406.14024, 2024a. URL https://doi.org/10.48550/arXiv.2406.14024.

Jiaxuan Gao, Shusheng Xu, Wenjie Ye, Weiling Liu, Chuyi He, Wei Fu, Zhiyu Mei, Guangju Wang, and Yi Wu. On designing effective rl reward at training time for llm reasoning. *ArXiv*, abs/2410.15115, 2024b. URL https://arxiv.org/pdf/2410.15115.

Kuofeng Gao, Huanqia Cai, Qingyao Shuai, Dihong Gong, and Zhifeng Li. Embedding self-correction as an inherent ability in large language models for enhanced mathematical reasoning. *CoRR*, abs/2410.10735, 2024c. doi: 10.48550/ARXIV.2410.10735. URL https://doi.org/10.48550/arXiv.2410.10735.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. RLEF: grounding code llms in execution feedback with reinforcement learning. *CoRR*, abs/2410.02089, 2024. doi: 10.48550/ARXIV.2410.02089. URL https://doi.org/10.48550/arXiv.2410.02089.

Robert Geirhos, Jörn-Henrik Jacobsen, Claudio Michaelis, Richard S. Zemel, Wieland Brendel, Matthias Bethge, and Felix Wichmann. Shortcut learning in deep neural networks. *Nature Machine Intelligence*, 2:665 – 673, 2020. URL https://www.nature.com/articles/s42256-020-00257-z.

Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Neural Information Processing Systems*, 2014. URL https://arxiv.org/pdf/1406.2661.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. CRITIC: large language models can self-correct with tool-interactive critiquing. 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=Sx038qxjek.

Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, Yuanzhuo Wang, and Jian Guo. A survey on llm-as-a-judge. *ArXiv*, abs/2411.15594, 2024. URL https://arxiv.org/pdf/2411.15594.

Xinyu Guan, Li Lyna Zhang, Yifei Liu, Ning Shang, Youran Sun, Yi Zhu, Fan Yang, and Mao Yang. rstar-math: Small llms can master math reasoning with self-evolved deep thinking. 2025. URL https://arxiv.org/pdf/2501.04519.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alexa Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, A. Doucet, Orhan Firat, and Nando de Freitas. Reinforced self-training (rest) for language modeling. *ArXiv*, abs/2308.08998, 2023. URL https://arxiv.org/pdf/2308.08998.

Devaansh Gupta and Boyang Li. A training data recipe to accelerate a* search with language models. *ArXiv*, abs/2407.09985, 2024. URL https://arxiv.org/pdf/2407.09985.

Tuomas Haarnoja, Haoran Tang, P. Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning*, 2017. URL https://arxiv.org/pdf/1702.08165.

Patrick M. Haluptzok, Matthew Bowers, and Adam Tauman Kalai. Language models can teach themselves to program better. *ArXiv*, abs/2207.14502, 2022. URL https://arxiv.org/abs/2207.14502.

Shibo Hao, Yi Gu, Haodi Ma, Joshua Jiahua Hong, Zhen Wang, Daisy Zhe Wang, and Zhiting Hu. Reasoning with language model is planning with world model. *ArXiv*, abs/2305.14992, 2023. URL https://arxiv.org/pdf/2305.14992.

Haoran He, Chenjia Bai, Ling Pan, Weinan Zhang, Bin Zhao, and Xuelong Li. Learning an actionable discrete diffusion policy via large-scale actionless video pre-training. In *Neural Information Processing Systems*, 2024a. URL https://arxiv.org/pdf/2402.14407.

Mingqian He, Yongliang Shen, Wenqi Zhang, Zeqi Tan, and Weiming Lu. Advancing process verification for large language models via tree-based preference learning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 2086–2099. Association for Computational Linguistics, 2024b. URL https://aclanthology.org/2024.emnlp-main.125.

Tao He, Lizi Liao, Yixin Cao, Yuanxing Liu, Ming Liu, Zerui Chen, and Bing Qin. Planning like human: A dual-process framework for dialogue planning. *ArXiv*, abs/2406.05374, 2024c. URL https://arxiv.org/pdf/2406.05374.

Jiwoo Hong, Noah Lee, and James Thorne. Orpo: Monolithic preference optimization without reference model. *ArXiv*, abs/2403.07691, 2024. URL https://aclanthology.org/2024.emnlp-main.626.pdf.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron C. Courville, Alessandro Sordoni, and Rishabh Agarwal. V-star: Training verifiers for self-taught reasoners. *ArXiv*, abs/2402.06457, 2024. URL https://arxiv.org/pdf/2402.06457.

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. Advancing language model reasoning through reinforcement learning and inference scaling. 2025. URL https://arxiv.org/pdf/2501.11651.

Chi Hu, Yimin Hu, Hang Cao, Tong Xiao, and Jingbo Zhu. Teaching language models to self-improve by learning from language feedback. In *Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/pdf/2406.07168.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. *ArXiv*, abs/2210.11610, 2022. URL `https://arxiv.org/pdf/2210.11610`.

Zhen Huang, Haoyang Zou, Xuefeng Li, Yixiu Liu, Yuxiang Zheng, Ethan Chern, Shijie Xia, Yiwei Qin, Weizhe Yuan, and Pengfei Liu. O1 replication journey - part 2: Surpassing o1-preview through simple distillation, big progress or bitter lesson? *ArXiv*, abs/2411.16489, 2024. URL `https://arxiv.org/pdf/2411.16489`.

Wenyang Hui, Chengyue Jiang, Yan Wang, and Kewei Tu. Rot: Enhancing large language models with reflection on search trees. *CoRR*, abs/2404.05449, 2024. doi: 10.48550/ARXIV.2404.05449. URL `https://doi.org/10.48550/arXiv.2404.05449`.

Hyeonbin Hwang, Doyoung Kim, Seungone Kim, Seonghyeon Ye, and Minjoon Seo. Self-explore: Enhancing mathematical reasoning in language models with fine-grained rewards. In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL `https://aclanthology.org/2024.findings-emnlp.78.pdf`.

Jinhao Jiang, Zhipeng Chen, Yingqian Min, Jie Chen, Xiaoxue Cheng, Jiapeng Wang, Yiru Tang, Haoxiang Sun, Jia Deng, Wayne Xin Zhao, Zheng Liu, Dong Yan, Jian Xie, Zhongyuan Wang, and Ji-Rong Wen. Enhancing llm reasoning with reward-guided tree search. 2024a. URL `https://arxiv.org/pdf/2411.11694`.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T. Kwok. Forward-backward reasoning in large language models for mathematical verification. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 6647–6661. Association for Computational Linguistics, 2024b. doi: 10.18653/V1/2024.FINDINGS-ACL.397. URL `https://doi.org/10.18653/v1/2024.findings-acl.397`.

Fangkai Jiao, Chengwei Qin, Zhengyuan Liu, Nancy F. Chen, and Shafiq R. Joty. Learning planning-based reasoning by trajectories collection and process reward synthesizing. In *Conference on Empirical Methods in Natural Language Processing*, 2024. URL `https://aclanthology.org/2024.emnlp-main.20/`.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. Offline reinforcement learning with implicit q-learning. *ArXiv*, abs/2110.06169, 2021. URL `https://arxiv.org/pdf/2110.06169`.

Jens Kreber and Christopher Hahn. Generating symbolic reasoning problems with transformer gans. *ArXiv*, abs/2110.10054, 2021. URL `https://arxiv.org/abs/2110.10054`.

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. *CoRR*, abs/2409.12917, 2024. doi: 10.48550/ARXIV.2409.12917. URL `https://doi.org/10.48550/arXiv.2409.12917`.

Xin Lai, Zhuotao Tian, Yukang Chen, Senqiao Yang, Xiangru Peng, and Jiaya Jia. Step-dpo: Step-wise preference optimization for long-chain reasoning of llms. *ArXiv*, abs/2406.18629, 2024. URL `https://arxiv.org/pdf/2406.18629`.

Nevena Lazic, Yasin Abbasi-Yadkori, Kush Bhatia, G. Weisz, Peter L. Bartlett, and Csaba Szepesvari. Politex: Regret bounds for policy iteration using expert prediction. In *International Conference on Machine Learning*, 2019. URL `https://proceedings.mlr.press/v97/lazic19a/lazic19a-supp.pdf`.

Long Tan Le, Han Shu, Tung-Anh Nguyen, Choong Seon Hong, and Nguyen H. Tran. irepo: implicit reward pairwise difference based empirical preference optimization. *ArXiv*, abs/2405.15230, 2024. URL `https://arxiv.org/pdf/2405.15230`.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Thomas Mesnard, Johan Ferret, Kellie Lu, Colton Bishop, Ethan Hall, Victor Carbune, Abhinav Rastogi, and Sushant Prakash. Rlaif vs. rlhf: Scaling reinforcement learning from human feedback with 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=uydQ2W41KO.

Jaehyeok Lee, Keisuke Sakaguchi, and Jinyeong Bak. Self-training meets consistency: Improving llms' reasoning with consistency-driven rationale evaluation. *ArXiv*, abs/2411.06387, 2024b. URL https://doi.org/10.48550/arXiv.2411.06387.

Jung Hyun Lee, June Yong Yang, Byeongho Heo, Dongyoon Han, and Kang Min Yoo. Token-supervised value models for enhancing mathematical reasoning capabilities of large language models. *ArXiv*, abs/2407.12863, 2024c. URL https://arxiv.org/pdf/2407.12863.

Jung Hyun Lee, June Yong Yang, Byeongho Heo, Dongyoon Han, and Kang Min Yoo. Token-supervised value models for enhancing mathematical reasoning capabilities of large language models. *CoRR*, abs/2407.12863, 2024d. doi: 10.48550/ARXIV.2407.12863. URL https://doi.org/10.48550/arXiv.2407.12863.

Lucas Lehnert, Sainbayar Sukhbaatar, Paul Mcvay, Michael Rabbat, and Yuandong Tian. Beyond a*: Better planning with transformers via search dynamics bootstrapping. *ArXiv*, abs/2402.14083, 2024. URL https://arxiv.org/pdf/2402.14083.

Chen Li, Weiqi Wang, Jingcheng Hu, Yixuan Wei, Nanning Zheng, Han Hu, Zheng Zhang, and Houwen Peng. Common 7b language models already possess strong math capabilities. *ArXiv*, abs/2403.04706, 2024a. URL https://arxiv.org/pdf/2403.04706.

Chengpeng Li, Guanting Dong, Mingfeng Xue, Ru Peng, Xiang Wang, and Dayiheng Liu. Dotamath: Decomposition of thought with code assistance and self-correction for mathematical reasoning. *CoRR*, abs/2407.04078, 2024b. doi: 10.48550/ARXIV.2407.04078. URL https://doi.org/10.48550/arXiv.2407.04078.

Chengzu Li, Wenshan Wu, Huanyu Zhang, Yan Xia, Shaoguang Mao, Li Dong, Ivan Vuli'c, and Furu Wei. Imagine while reasoning in space: Multimodal visualization-of-thought. 2025. URL https://arxiv.org/pdf/2501.07542.

Siheng Li, Cheng Yang, Zesen Cheng, Lemao Liu, Mo Yu, Yujiu Yang, and Wai Lam. Large language models can self-improve in long-context reasoning. *ArXiv*, abs/2411.08147, 2024c. URL https://arxiv.org/pdf/2411.08147.

Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Luke Zettlemoyer, Omer Levy, Jason Weston, and Mike Lewis. Self-alignment with instruction backtranslation. *ArXiv*, abs/2308.06259, 2023a. URL https://arxiv.org/abs/2308.062599.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making language models better reasoners with step-aware verifier. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.291. URL https://aclanthology.org/2023.acl-long.291/.

Ziniu Li, Tian Xu, and Yang Yu. Policy optimization in rlhf: The impact of out-of-preference data. *ArXiv*, abs/2312.10584, 2023c. URL https://arxiv.org/pdf/2312.10584.

Ziniu Li, Tian Xu, Yushun Zhang, Yang Yu, Ruoyu Sun, and Zhimin Luo. Remax: A simple, effective, and efficient reinforcement learning method for aligning large language models. *ArXiv*, abs/2310.10505, 2023d. URL https://arxiv.org/pdf/2310.10505.

Zhenwen Liang, Ye Liu, Tong Niu, Xiangliang Zhang, Yingbo Zhou, and Semih Yavuz. Improving llm reasoning through scaling inference computation with collaborative verification. *CoRR*, abs/2410.05318, 2024. doi: 10.48550/ARXIV.2410.05318. URL https://doi.org/10.48550/arXiv.2410.05318.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *ArXiv*, abs/2305.20050, 2023. URL https://arxiv.org/pdf/2305.20050.

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=v8L0pN6EOi`.

Qingwen Lin, Boyan Xu, Zijian Li, Zhifeng Hao, Keli Zhang, and Ruichu Cai. Leveraging constrained monte carlo tree search to generate reliable long chain-of-thought for mathematical reasoning. 2025. URL `https://arxiv.org/pdf/2502.11169`.

Zicheng Lin, Tian Liang, Jiahao Xu, Xing Wang, Ruilin Luo, Chufan Shi, Siheng Li, Yujiu Yang, and Zhaopeng Tu. Critical tokens matter: Token-level contrastive estimation enhances llm's reasoning capability. *ArXiv*, abs/2411.19943, 2024. URL `https://arxiv.org/pdf/2411.19943`.

Bingbin Liu, Sébastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. Tinygsm: achieving >80% on gsm8k with small language models. *ArXiv*, abs/2312.09241, 2023. URL `https://arxiv.org/abs/2312.09241`.

Haoxiong Liu, Yifan Zhang, Yifan Luo, and Andrew Chi-Chih Yao. Augmenting math word problems via iterative question composing. *ArXiv*, abs/2401.09003, 2024a. URL `https://arxiv.org/abs/2401.09003`.

Rongxing Liu, Kumar Shridhar, Manish Prajapat, Patrick Xia, and Mrinmaya Sachan. Smart: Self-learning meta-strategy agent for reasoning tasks. *ArXiv*, abs/2410.16128, 2024b. URL `https://arxiv.org/pdf/2410.16128`.

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. *CoRR*, abs/2410.18528, 2024c. doi: 10.48550/ARXIV.2410.18528. URL `https://doi.org/10.48550/arXiv.2410.18528`.

Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yingjia Wan, Yinya Huang, and Zhijiang Guo. Autopsv: Automated process-supervised verifier. 2024a. URL `https://doi.org/10.48550/arXiv.2405.16802`.

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. Mathgenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of llms. *ArXiv*, abs/2402.16352, 2024b. URL `https://arxiv.org/abs/2402.16352`.

Zimu Lu, Aojun Zhou, Ke Wang, Houxing Ren, Weikang Shi, Junting Pan, and Mingjie Zhan. Step-controlled dpo: Leveraging stepwise error for enhanced mathematical reasoning. *ArXiv*, abs/2407.00782, 2024c. URL `https://arxiv.org/pdf/2407.00782`.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *ArXiv*, abs/2308.09583, 2023. URL `https://arxiv.org/abs/2308.09583`.

Haotian Luo, Li Shen, Haiying He, Yibo Wang, Shiwei Liu, Wei Li, Naiqiang Tan, Xiaochun Cao, and Dacheng Tao. O1-pruner: Length-harmonizing fine-tuning for o1-like reasoning pruning. 2025. URL `https://arxiv.org/pdf/2501.12570`.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, and Abhinav Rastogi. Improve mathematical reasoning in language models by automated process supervision. *ArXiv*, abs/2406.06592, 2024. URL `https://api.semanticscholar.org/CorpusID:270379625`.

Chengqi Lyu, Songyang Gao, Yuzhe Gu, Wenwei Zhang, Jianfei Gao, Kuikun Liu, Ziyi Wang, Shuaibin Li, Qian Zhao, Haian Huang, Weihan Cao, Jiangning Liu, Hongwei Liu, Junnan Liu, Songyang Zhang, Dahua Lin, and Kai Chen. Exploring the limit of outcome reward for learning mathematical reasoning. 2025. URL `https://arxiv.org/pdf/2502.06781`.

Qianli Ma, Haotian Zhou, Tingkai Liu, Jianbo Yuan, Pengfei Liu, Yang You, and Hongxia Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *ArXiv*, abs/2310.10080, 2023. URL `https://arxiv.org/pdf/2310.10080`.

Aman Madaan, Alex Shypula, Uri Alon, Milad Hashemi, Parthasarathy Ranganathan, Yiming Yang, Graham Neubig, and Amir Yazdanbakhsh. Learning performance-improving code edits. *ArXiv*, abs/2302.07867, 2023a. URL `https://arxiv.org/abs/2302.07867`.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html`.

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

Silin Meng, Yiwei Wang, Cheng-Fu Yang, Nanyun Peng, and Kai-Wei Chang. Llm-a*: Large language model enhanced incremental heuristic search on path planning. *ArXiv*, abs/2407.02511, 2024a. URL `https://arxiv.org/pdf/2407.02511`.

Yu Meng, Mengzhou Xia, and Danqi Chen. Simpo: Simple preference optimization with a reference-free reward. *ArXiv*, abs/2405.14734, 2024b. URL `https://arxiv.org/pdf/2405.14734`.

Yingqian Min, Zhipeng Chen, Jinhao Jiang, Jie Chen, Jia Deng, Yiwen Hu, Yiru Tang, Jiapeng Wang, Xiaoxue Cheng, Huatong Song, Wayne Xin Zhao, Zheng Liu, Zhongyuan Wang, and Jiahui Wen. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems. 2024. URL `https://arxiv.org/pdf/2412.09413`.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. Orca-math: Unlocking the potential of slms in grade school math. *ArXiv*, abs/2402.14830, 2024. URL `https://arxiv.org/abs/2402.14830`.

Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. Model-based reinforcement learning: A survey. *Found. Trends Mach. Learn.*, 16:1–118, 2020. URL `https://arxiv.org/pdf/2006.16712`.

Todd K. Moon. The expectation-maximization algorithm. *IEEE Signal Process. Mag.*, 13:47–60, 1996. URL `https://ieeexplore.ieee.org/document/543975/`.

Tong Mu, Alec Helyar, Jo hannes Heidecke, Joshua Achiam, Andrea Vallone, Ian D. Kivlichan, Molly Lin, Alex Beutel, John Schulman, and Lilian Weng. Rule based rewards for language model safety. *ArXiv*, abs/2411.01111, 2024. URL `https://doi.org/10.48550/arXiv.2411.01111`.

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

OpenAI. Openai o1 system card, 2024b. URL `https://cdn.openai.com/o1-system-card-20241205.pdf`.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke E. Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Francis Christiano, Jan Leike, and Ryan J. Lowe. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155, 2022. URL `https://arxiv.org/pdf/2203.02155`.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. Iterative reasoning preference optimization. *ArXiv*, abs/2404.19733, 2024. URL `https://arxiv.org/pdf/2404.19733`.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. REFINER: reasoning feedback on intermediate representations. In Yvette Graham and Matthew Purver, editors, *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2024 - Volume 1: Long Papers, St. Julian's, Malta, March 17-22, 2024*, pages 1100–1126. Association for Computational Linguistics, 2024. URL `https://aclanthology.org/2024.eacl-long.67`.

Baolin Peng, Michel Galley, Pengcheng He, Hao Cheng, Yujia Xie, Yu Hu, Qiuyuan Huang, Lars Liden, Zhou Yu, Weizhu Chen, and Jianfeng Gao. Check your facts and try again: Improving large language models with external knowledge and automated feedback. *CoRR*, abs/2302.12813, 2023. doi: 10.48550/ARXIV.2302.12813. URL `https://doi.org/10.48550/arXiv.2302.12813`.

Xiangyu Peng, Congying Xia, Xinyi Yang, Caiming Xiong, Chien-Sheng Wu, and Chen Xing. Regenesis: Llms can grow into reasoning generalists via self-improvement. *ArXiv*, abs/2410.02108, 2024. URL `https://arxiv.org/pdf/2410.02108`.

Stanislas Polu, Jesse Michael Han, Kunhao Zheng, Mantas Baksys, Igor Babuschkin, and Ilya Sutskever. Formal mathematics statement curriculum learning. *ArXiv*, abs/2202.01344, 2022. URL `https://arxiv.org/pdf/2202.01344`.

Rafael Figueiredo Prudencio, Marcos R.O.A. Maximo, and Esther Luna Colombini. A survey on offline reinforcement learning: Taxonomy, review, and open problems. *IEEE Transactions on Neural Networks and Learning Systems*, 35:10237–10257, 2022. URL `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10078377`.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. Mutual reasoning makes smaller llms stronger problem-solvers. *ArXiv*, abs/2408.06195, 2024. URL `https://arxiv.org/pdf/2408.06195`.

Yiwei Qin, Xuefeng Li, Haoyang Zou, Yixiu Liu, Shijie Xia, Zhen Huang, Yixin Ye, Weizhe Yuan, Hector Liu, Yuanzhi Li, and Pengfei Liu. O1 replication journey: A strategic progress report - part 1. *ArXiv*, abs/2410.18982, 2024. URL `https://arxiv.org/pdf/2410.18982`.

Yujia Qin, Shi Liang, Yining Ye, Kunlun Zhu, Lan Yan, Ya-Ting Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Marc H. Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789, 2023. URL `https://arxiv.org/pdf/2307.16789`.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *ArXiv*, abs/2305.18290, 2023. URL `https://arxiv.org/pdf/2305.18290`.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. From r to q^*: Your language model is secretly a q-function. 2024. URL `https://arxiv.org/pdf/2404.12358`.

Keshav Ramji, Young-Suk Lee, Ramón Fernandez Astudillo, Md. Arafat Sultan, Tahira Naseem, Asim Munawar, Radu Florian, and Salim Roukos. Self-refinement of language models from external proxy metrics feedback. *CoRR*, abs/2403.00827, 2024. doi: 10.48550/ARXIV.2403.00827. URL `https://doi.org/10.48550/arXiv.2403.00827`.

Christopher D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61:203–230, 2011. URL `https://link.springer.com/article/10.1007/s10472-011-9258-6`.

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

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

Bilgehan Sel, Ahmad S. Al-Tawaha, Vanshaj Khattar, Lucy Wang, R. Jia, and Ming Jin. Algorithm of thoughts: Enhancing exploration of ideas in large language models. *ArXiv*, abs/2308.10379, 2023. URL `https://arxiv.org/pdf/2308.10379`.

Amrith Rajagopal 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. *ArXiv*, abs/2410.08146, 2024. URL `https://arxiv.org/pdf/2410.08146`.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Jun-Mei Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300, 2024. URL `https://arxiv.org/pdf/2402.03300`.

Freda Shi, Xinyun Chen, Kanishka Misra, Nathan Scales, David Dohan, Ed H. Chi, Nathanael Scharli, and Denny Zhou. Large language models can be easily distracted by irrelevant context. In *International Conference on Machine Learning*, 2023. URL `https://arxiv.org/abs/2302.00093`.

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, editors, *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`.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ramakanth Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. The ART of LLM refinement: Ask, refine, and trust. In Kevin Duh, Helena Gómez-Adorno, and Steven Bethard, editors, *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, pages 5872–5883. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.NAACL-LONG.327. URL `https://doi.org/10.18653/v1/2024.naacl-long.327`.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy P. Lillicrap, Fan Hui, L. Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017. URL `https://www.nature.com/articles/nature24270`.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Brian Warkentin, Yundi Qian, Ethan Dyer, Behnam Neyshabur, Jascha Narain Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Trans. Mach. Learn. Res.*, 2024, 2023. URL `https://arxiv.org/pdf/2312.06585`.

Charles Burton Snell, Ilya Kostrikov, Yi Su, Mengjiao Yang, and Sergey Levine. Offline rl for natural language generation with implicit language q learning. *ArXiv*, abs/2206.11871, 2022. URL `https://api.semanticscholar.org/CorpusID:249954054`.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *ArXiv*, abs/2408.03314, 2024. URL `https://arxiv.org/pdf/2408.03314`.

Yifan Song, Da Yin, Xiang Yue, Jie Huang, Sujian Li, and Bill Yuchen Lin. Trial and error: Exploration-based trajectory optimization for llm agents. *ArXiv*, abs/2403.02502, 2024. URL `https://arxiv.org/pdf/2403.02502`.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. Learning to summarize with human feedback. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *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/1f89885d556929e98d3ef9b86448f951-Abstract.html`.

DiJia Su, Sainbayar Sukhbaatar, Michael Rabbat, Yuandong Tian, and Qinqing Zheng. Dualformer: Controllable fast and slow thinking by learning with randomized reasoning traces. *ArXiv*, abs/2410.09918, 2024. URL `https://arxiv.org/pdf/2410.09918`.

Ilya Sutskever. Sequence to sequence learning with neural networks: what a decade, 2024. URL `https://www.youtube.com/watch?v=1yvBqasHLZs`.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems*, 1999. URL `https://www.cis.upenn.edu/~mkearns/finread/Sutton.pdf`.

Yunhao Tang, Zhaohan Daniel Guo, Zeyu Zheng, Daniele Calandriello, Rémi Munos, Mark Rowland, Pierre H. Richemond, Michal Valko, Bernardo Ávila Pires, and Bilal Piot. Generalized preference optimization: A unified approach to offline alignment. *ArXiv*, abs/2402.05749, 2024. URL `https://arxiv.org/pdf/2402.05749`.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *ArXiv*, abs/2404.14387, 2024. URL `https://arxiv.org/pdf/2404.14387`.

DeepSeek-AI Team. Deepseek-v3 technical report, 2024a. URL `https://github.com/deepseek-ai/DeepSeek-V3/blob/main/DeepSeek_V3.pdf`.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Feng Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Haochen Ding, Hao-Xing Hu, Haoming Yang, Hao Zhang, Haotian Yao, Hao-Dong Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jia-Xing Guo, Jianling Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Li-Na Shi, Li-Tao Ye, Long Yu, Meng-Xiao Dong, Neo Zhang, Ningchen Ma, Qi Pan, Qucheng Gong, Shaowei Liu, Shen Ma, Shu-Yan Wei, Sihan Cao, Si-Da Huang, Tao Jiang, Wei-Wei Gao, Weiming Xiong, Weiran He, Weixiao Huang, Wenhao Wu, Wen He, Xian sen Wei, Xian-Xian Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xue biao Pan, Y. Charles, Yang Li, Yan-Ling Hu, Yangyang Liu, Yanru Chen, Ye-Jia Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Yingbo Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhengxin Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhi-Sheng Huang, Zihao Huang, Ziya Xu, and Zonghan Yang. Kimi k1.5: Scaling reinforcement learning with llms. 2025. URL `https://arxiv.org/pdf/2501.12599`.

Qwen Team. Qwq: Reflect deeply on the boundaries of the unknown, 2024b. URL `https://qwenlm.github.io/blog/qwq-32b-preview/`.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *CoRR*, abs/2404.12253, 2024. doi: 10.48550/ARXIV.2404.12253. URL `https://doi.org/10.48550/arXiv.2404.12253`.

Yuxuan Tong, Xiwen Zhang, Rui Wang, Rui Min Wu, and Junxian He. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. *ArXiv*, abs/2407.13690, 2024. URL `https://arxiv.org/pdf/2407.13690`.

Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. Diverse beam search: Decoding diverse solutions from neural sequence models. *ArXiv*, abs/1610.02424, 2016. URL `https://arxiv.org/pdf/1610.02424`.

Manya Wadhwa, Xinyu Zhao, Junyi Jessy Li, and Greg Durrett. Learning to refine with fine-grained natural language feedback. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 12281–12308. Association for Computational Linguistics, 2024. URL `https://aclanthology.org/2024.findings-emnlp.716`.

Chaojie Wang, Yanchen Deng, Zhiyi Lv, Zeng Liang, Jujie He, Shuicheng Yan, and Bo An. Q*: Improving multi-step reasoning for llms with deliberative planning. *ArXiv*, abs/2406.14283, 2024a. URL `https://arxiv.org/pdf/2406.14283`.

Chaoqi Wang, Yibo Jiang, Chenghao Yang, Han Liu, and Yuxin Chen. Beyond reverse kl: Generalizing direct preference optimization with diverse divergence constraints. *ArXiv*, abs/2309.16240, 2023a. URL `https://arxiv.org/pdf/2309.16240`.

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. Offline reinforcement learning for llm multi-step reasoning. 2024b. URL `https://arxiv.org/pdf/2412.16145`.

Jianing Wang, Yang Zhou, Xiaocheng Zhang, Mengjiao Bao, and Peng Yan. Self-evolutionary large language models through uncertainty-enhanced preference optimization. *ArXiv*, abs/2409.11212, 2024c. URL `https://arxiv.org/pdf/2409.11212`.

Jiaqi Wang, Zihao Wu, Yiwei Li, Hanqi Jiang, Peng Shu, Enze Shi, Huawen Hu, Chong-Yi Ma, Yi-Hsueh Liu, Xuhui Wang, Yincheng Yao, Xuan Liu, Huaqin Zhao, Zheng Liu, Haixing Dai, Lin Zhao, Bao Ge, Xiang Li, Tianming Liu, and Shu Zhang. Large language models for robotics: Opportunities, challenges, and perspectives. *ArXiv*, abs/2401.04334, 2024d. URL `https://arxiv.org/pdf/2401.04334`.

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M. Ni, Linyi Yang, Ying Wen, and Weinan Zhang. Openr: An open source framework for advanced reasoning with large language models. *ArXiv*, abs/2410.09671, 2024e. URL `https://arxiv.org/pdf/2410.09671`.

Ke Wang, Jiahui Zhu, Minjie Ren, Zeming Liu, Shiwei Li, Zongye Zhang, Chenkai Zhang, Xiaoyu Wu, Qiqi Zhan, Qingjie Liu, and Yunhong Wang. A survey on data synthesis and augmentation for large language models. *ArXiv*, abs/2410.12896, 2024f. URL `https://arxiv.org/pdf/2410.12896`.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Annual Meeting of the Association for Computational Linguistics*, 2023b. URL `https://aclanthology.org/2023.acl-long.147.pdf`.

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, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9426–9439. Association for Computational Linguistics, 2024g. doi: 10.18653/V1/2024.ACL-LONG.510. URL `https://doi.org/10.18653/v1/2024.acl-long.510`.

Shuhe Wang, Shengyu Zhang, Jie Zhang, Runyi Hu, Xiaoya Li, Tianwei Zhang, Jiwei Li, Fei Wu, Guoyin Wang, and Eduard H. Hovy. Reinforcement learning enhanced llms: A survey. 2024h. URL `https://arxiv.org/pdf/2412.10400`.

Teng Wang, Wing-Yin Yu, Zhenqi He, Zehua Liu, Xiongwei Han, Hailei Gong, Han Wu, Wei Shi, Ruifeng She, Fangzhou Zhu, and Tao Zhong. Bpp-search: Enhancing tree of thought reasoning for mathematical modeling problem solving. *ArXiv*, abs/2411.17404, 2024i. URL `https://arxiv.org/pdf/2411.17404`.

Tianduo Wang, Shichen Li, and Wei Lu. Self-training with direct preference optimization improves chain-of-thought reasoning. *ArXiv*, abs/2407.18248, 2024j. URL `https://arxiv.org/pdf/2407.18248`.

Xiyao Wang, Linfeng Song, Ye Tian, Dian Yu, Baolin Peng, Haitao Mi, Furong Huang, and Dong Yu. Towards self-improvement of llms via mcts: Leveraging stepwise knowledge with curriculum preference learning. *ArXiv*, abs/2410.06508, 2024k. URL `https://arxiv.org/pdf/2410.06508`.

Xuezhi Wang and Denny Zhou. Chain-of-thought reasoning without prompting. *ArXiv*, abs/2402.10200, 2024. URL https://arxiv.org/pdf/2402.10200.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023c. URL https://openreview.net/forum?id=1PL1NIMMrw.

Yaoke Wang, Yun Zhu, Xintong Bao, Wenqiao Zhang, Suyang Dai, Kehan Chen, Wenqiang Li, Gang Huang, Siliang Tang, and Yueting Zhuang. Meta-reflection: A feedback-free reflection learning framework. 2024l. URL https://doi.org/10.48550/arXiv.2412.13781.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Annual Meeting of the Association for Computational Linguistics*, 2022. URL https://arxiv.org/abs/2212.10560.

Yue Wang, Qiuzhi Liu, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Linfeng Song, Dian Yu, Juntao Li, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Thoughts are all over the place: On the underthinking of o1-like llms. 2025a. URL https://arxiv.org/pdf/2501.18585.

Yutong Wang, Pengliang Ji, Chaoqun Yang, Kaixin Li, Ming Hu, Jiaoyang Li, and Guillaume Sartoretti. Mcts-judge: Test-time scaling in llm-as-a-judge for code correctness evaluation. 2025b. URL https://arxiv.org/pdf/2502.12468.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 7309–7319. Association for Computational Linguistics, 2024m. URL https://aclanthology.org/2024.findings-emnlp.429.

Ziqi Wang, Le Hou, Tianjian Lu, Yuexin Wu, Yunxuan Li, Hongkun Yu, and Heng Ji. Enabling language models to implicitly learn self-improvement. 2023d. URL https://arxiv.org/pdf/2310.00898.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, F. Xia, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022. URL https://arxiv.org/pdf/2201.11903.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering code generation with oss-instruct. In *International Conference on Machine Learning*, 2023. URL https://arxiv.org/abs/2312.02120.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net, 2023. URL https://openreview.net/forum?id=hH36JeQZDaO.

Lilian Weng. Reward hacking in reinforcement learning. *lilianweng.github.io*, Nov 2024. URL https://lilianweng.github.io/posts/2024-11-28-reward-hacking/.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 2550–2575. Association for Computational Linguistics, 2023. doi: 10.18653/V1/2023.FINDINGS-EMNLP.167. URL https://doi.org/10.18653/v1/2023.findings-emnlp.167.

Jinyang Wu, Mingkuan Feng, Shuai Zhang, Feihu Che, Zengqi Wen, and Jianhua Tao. Beyond examples: High-level automated reasoning paradigm in in-context learning via mcts. *ArXiv*, abs/2411.18478, 2024a. URL https://arxiv.org/pdf/2411.18478.

Jinyang Wu, Mingkuan Feng, Shuai Zhang, Ruihan Jin, Feihu Che, Zengqi Wen, and Jianhua Tao. Boosting multimodal reasoning with mcts-automated structured thinking. 2025a. URL https://arxiv.org/pdf/2502.02339.

Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in llms. 2025b. URL https://arxiv.org/pdf/2502.07266.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. Large language models can self-correct with key condition verification. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 12846–12867. Association for Computational Linguistics, 2024b. URL https://aclanthology.org/2024.emnlp-main.714.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. Enhancing mathematical reasoning in llms by stepwise correction. *CoRR*, abs/2410.12934, 2024c. doi: 10.48550/ARXIV.2410.12934. URL https://doi.org/10.48550/arXiv.2410.12934.

Zhiheng Xi, Dingwen Yang, Jixuan Huang, Jiafu Tang, Guanyu Li, Yiwen Ding, Wei He, Boyang Hong, Shihan Do, Wenyu Zhan, Xiao Wang, Rui Zheng, Tao Ji, Xiaowei Shi, Yitao Zhai, Rongxiang Weng, Jingang Wang, Xun-Ye Cai, Tao Gui, Zuxuan Wu, Qi Zhang, Xipeng Qiu, Xuanjing Huang, and Yuxin Jiang. Enhancing llm reasoning via critique models with test-time and training-time supervision. 2024. URL https://doi.org/10.48550/arXiv.2411.16579.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. Evaluating mathematical reasoning beyond accuracy. *ArXiv*, abs/2404.05692, 2024. URL https://doi.org/10.48550/arXiv.2404.05692.

Kun Xiang, Zhili Liu, Zihao Jiang, Yunshuang Nie, Runhu Huang, Haoxiang Fan, Hanhui Li, Weiran Huang, Yihan Zeng, Jianhua Han, Lanqing Hong, Hang Xu, and Xiaodan Liang. Atomthink: A slow thinking framework for multimodal mathematical reasoning. *ArXiv*, abs/2411.11930, 2024. URL https://arxiv.org/pdf/2411.11930.

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. 2025a. URL https://arxiv.org/pdf/2502.14768.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Qizhe Xie. Self-evaluation guided beam search for reasoning. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *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/81fde95c4dc79188a69ce5b24d63010b-Abstract-Conference.html.

Yuxi Xie, Anirudh Goyal, Wenyue Zheng, Min-Yen Kan, Timothy P. Lillicrap, Kenji Kawaguchi, and Michael Shieh. Monte carlo tree search boosts reasoning via iterative preference learning. *ArXiv*, abs/2405.00451, 2024. URL https://arxiv.org/pdf/2405.00451.

Zhihui Xie, Jie chen, Liyu Chen, Weichao Mao, Jingjing Xu, and Lingpeng Kong. Teaching language models to critique via reinforcement learning. 2025b. URL https://arxiv.org/pdf/2502.03492.

Huajian Xin, Zhizhou Ren, Jun-Mei Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu (Benjamin Liu), Liyue Zhang, Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z. F. Wu, Fuli Luo, and Chong Ruan. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *ArXiv*, abs/2408.08152, 2024. URL https://arxiv.org/pdf/2408.08152.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. Wizardlm: Empowering large language models to follow complex instructions. *ArXiv*, abs/2304.12244, 2023. URL https://arxiv.org/abs/2304.12244.

Haotian Xu, Xing Wu, Weinong Wang, Zhongzhi Li, Da Zheng, Boyuan Chen, Yi Hu, Shijia Kang, Jiaming Ji, Yingying Zhang, Zhijiang Guo, Yaodong Yang, Muhan Zhang, and Debing Zhang. Redstar: Does scaling long-cot data unlock better slow-reasoning systems? 2025. URL https://arxiv.org/pdf/2501.11284.

Yuchen Yan, Jin Jiang, Yang Liu, Yixin Cao, Xin Xu, Mengdi Zhang, Xunliang Cai, and Jian Shao. S^3c-math: Spontaneous step-level self-correction makes large language models better mathematical reasoners. *CoRR*, abs/2409.01524, 2024. doi: 10.48550/ARXIV.2409.01524. URL https://doi.org/10.48550/arXiv.2409.01524.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *ArXiv*, abs/2409.12122, 2024a. URL https://arxiv.org/pdf/2409.12122.

Kailai Yang, Zhiwei Liu, Qianqian Xie, Jimin Huang, Erxue Min, and Sophia Ananiadou. Selective preference optimization via token-level reward function estimation. *ArXiv*, abs/2408.13518, 2024b. URL https://arxiv.org/pdf/2408.13518.

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. Reasonflux: Hierarchical llm reasoning via scaling thought templates. 2025a. URL https://arxiv.org/pdf/2502.06772.

Qwen An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxin Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yi-Chao Zhang, Yunyang Wan, Yuqi Liu, Zeyu Cui, Zhenru Zhang, Zihan Qiu, and Shanghaoran Quan. Qwen2.5 technical report. *ArXiv*, abs/2412.15115, 2024c. URL https://arxiv.org/pdf/2412.15115.

Xiaowen Yang, Xuan-Yi Zhu, Wenda Wei, Ding-Chu Zhang, Jiejing Shao, Zhi Zhou, Lan-Zhe Guo, and Yu-Feng Li. Step back to leap forward: Self-backtracking for boosting reasoning of language models. 2025b. URL https://arxiv.org/pdf/2502.04404.

Huanjin Yao, Jiaxing Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, and Dacheng Tao. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search. *ArXiv*, abs/2412.18319, 2024a. URL https://arxiv.org/pdf/2412.18319.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. *ArXiv*, abs/2210.03629, 2022. URL https://arxiv.org/pdf/2210.03629.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *ArXiv*, abs/2305.10601, 2023. URL https://arxiv.org/pdf/2305.10601.

Weiran Yao, Shelby Heinecke, Juan Carlos Niebles, Zhiwei Liu, Yihao Feng, Le Xue, Rithesh R. N., Zeyuan Chen, Jianguo Zhang, Devansh Arpit, Ran Xu, Phil Mui, Huan Wang, Caiming Xiong, and Silvio Savarese. Retroformer: Retrospective large language agents with policy gradient optimization. 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=KOZu91CzbK.

Zihuiwen Ye, Luckeciano Carvalho Melo, Younesse Kaddar, Phil Blunsom, Sam Staton, and Yarin Gal. Uncertainty-aware step-wise verification with generative reward models. 2025. URL https://arxiv.org/pdf/2502.11250.

Ziyu Ye, Rishabh Agarwal, Tianqi Liu, Rishabh Joshi, Sarmishta Velury, Quoc Le, Qijun Tan, and Yuan Liu. Evolving alignment via asymmetric self-play. *ArXiv*, abs/2411.00062, 2024. URL https://arxiv.org/pdf/2411.00062.

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. Demystifying long chain-of-thought reasoning in llms. 2025. URL https://arxiv.org/pdf/2502.03373.

ylfeng, Yang Xu, Libo Qin, Yasheng Wang, and Wanxiang Che. Improving language model reasoning with self-motivated learning. *ArXiv*, abs/2404.07017, 2024. URL https://aclanthology.org/2024.lrec-main.774.pdf.

Eunseop Yoon, Hee Suk Yoon, SooHwan Eom, Gunsoo Han, Daniel Wontae Nam, DaeJin Jo, Kyoung-Woon On, Mark A. Hasegawa-Johnson, Sungwoong Kim, and Chang Dong Yoo. Tlcr: Token-level continuous reward for fine-grained reinforcement learning from human feedback. *ArXiv*, abs/2407.16574, 2024. URL https://arxiv.org/pdf/2407.16574.

Dian Yu, Baolin Peng, Ye Tian, Linfeng Song, Haitao Mi, and Dong Yu. Siam: Self-improving code-assisted mathematical reasoning of large language models. *ArXiv*, abs/2408.15565, 2024a. URL https://arxiv.org/pdf/2408.15565.

Fei Yu, Anningzhe Gao, and Benyou Wang. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *NAACL-HLT*, 2023a. URL https://arxiv.org/pdf/2311.09724.

Long Long Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zheng Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *ArXiv*, abs/2309.12284, 2023b. URL https://arxiv.org/abs/2309.12284.

Yue Yu, Zhengxing Chen, Aston Zhang, Liang Tan, Chenguang Zhu, Richard Yuanzhe Pang, Yundi Qian, Xuewei Wang, Suchin Gururangan, Chao Zhang, Melissa Hall Melanie Kambadur, Dhruv Mahajan, and Rui Hou. Self-generated critiques boost reward modeling for language models. *ArXiv*, abs/2411.16646, 2024b. URL https://arxiv.org/pdf/2411.16646.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. Advancing LLM reasoning generalists with preference trees. *CoRR*, abs/2404.02078, 2024a. doi: 10.48550/ARXIV.2404.02078. URL https://doi.org/10.48550/arXiv.2404.02078.

Lifan Yuan, Ganqu Cui, Hanbin Wang, Ning Ding, Xingyao Wang, Jia Deng, Boji Shan, Huimin Chen, Ruobing Xie, Yankai Lin, Zhenghao Liu, Bowen Zhou, Hao Peng, Zhiyuan Liu, and Maosong Sun. Advancing llm reasoning generalists with preference trees. *ArXiv*, abs/2404.02078, 2024b. URL https://arxiv.org/pdf/2404.02078.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. Free process rewards without process labels. 2024c. URL https://arxiv.org/pdf/2412.01981.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *ArXiv*, abs/2401.10020, 2024d. URL https://arxiv.org/pdf/2401.10020.

Zheng Yuan, Hongyi Yuan, Cheng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. Scaling relationship on learning mathematical reasoning with large language models. *ArXiv*, abs/2308.01825, 2023a. URL https://arxiv.org/pdf/2308.01825.

Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, Songfang Huang, and Feiran Huang. Rrhf: Rank responses to align language models with human feedback without tears. *ArXiv*, abs/2304.05302, 2023b. URL https://arxiv.org/pdf/2304.05302.

E. Zelikman, Yuhuai Wu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning. 2022. URL https://arxiv.org/pdf/2203.14465.

Weihao Zeng, Yuzhen Huang, Lulu Zhao, Yijun Wang, Zifei Shan, and Junxian He. B-star: Monitoring and balancing exploration and exploitation in self-taught reasoners. 2024a. URL https://arxiv.org/pdf/2412.17256.

Zhiyuan Zeng, Qinyuan Cheng, Zhangyue Yin, Bo Wang, Shimin Li, Yunhua Zhou, Qipeng Guo, Xuanjing Huang, and Xipeng Qiu. Scaling of search and learning: A roadmap to reproduce o1 from reinforcement learning perspective. *ArXiv*, abs/2412.14135, 2024b. URL `https://arxiv.org/pdf/2412.14135`.

Bohan Zhang, Xiaokang Zhang, Jing Zhang, Jifan Yu, Sijia Luo, and Jie Tang. Cot-based synthesizer: Enhancing llm performance through answer synthesis. 2025a. URL `https://doi.org/10.48550/arXiv.2501.01668`.

Che Zhang, Zhenyang Xiao, Chengcheng Han, Yixin Lian, and Yuejian Fang. Learning to check: Unleashing potentials for self-correction in large language models. 2024a. URL `https://doi.org/10.48550/arXiv.2402.13035`.

Dan Zhang, Ziniu Hu, Sining Zhoubian, Zhengxiao Du, Kaiyu Yang, Zihan Wang, Yisong Yue, Yuxiao Dong, and Jie Tang. Sciinstruct: a self-reflective instruction annotated dataset for training scientific language models. 2024b. URL `https://doi.org/10.48550/arXiv.2401.07950`.

Dan Zhang, Sining Zhoubian, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *ArXiv*, abs/2406.03816, 2024c. URL `https://arxiv.org/pdf/2406.03816`.

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *ArXiv*, abs/2406.07394, 2024d. URL `https://doi.org/10.48550/arXiv.2406.07394`.

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, Wanli Ouyang, and Dongzhan Zhou. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *CoRR*, abs/2410.02884, 2024e. doi: 10.48550/ARXIV.2410.02884. URL `https://doi.org/10.48550/arXiv.2410.02884`.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. Generative verifiers: Reward modeling as next-token prediction. *ArXiv*, abs/2408.15240, 2024f. URL `https://arxiv.org/pdf/2408.15240`.

Tianjun Zhang, Fangchen Liu, Justin Wong, P. Abbeel, and Joseph E. Gonzalez. The wisdom of hindsight makes language models better instruction followers. In *International Conference on Machine Learning*, 2023a. URL `https://arxiv.org/pdf/2302.05206`.

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. Self-contrast: Better reflection through inconsistent solving perspectives. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 3602–3622. Association for Computational Linguistics, 2024g. doi: 10.18653/V1/2024.ACL-LONG.197. URL `https://doi.org/10.18653/v1/2024.acl-long.197`.

Xuan Zhang, Chao Du, Tianyu Pang, Qian Liu, Wei Gao, and Min Lin. Chain of preference optimization: Improving chain-of-thought reasoning in llms. *ArXiv*, abs/2406.09136, 2024h. URL `https://arxiv.org/pdf/2406.09136`.

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. Small language models need strong verifiers to self-correct reasoning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar, editors, *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 15637–15653. Association for Computational Linguistics, 2024i. doi: 10.18653/V1/2024.FINDINGS-ACL.924. URL `https://doi.org/10.18653/v1/2024.findings-acl.924`.

Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong, and Jitao Sang. o1-coder: an o1 replication for coding. 2024j. URL `https://arxiv.org/pdf/2412.00154`.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. The lessons of developing process reward models in mathematical reasoning. 2025b. URL `https://doi.org/10.48550/arXiv.2501.07301`.

Zheyu Zhang, Zhuorui Ye, Yikang Shen, and Chuang Gan. Autonomous tree-search ability of large language models. *ArXiv*, abs/2310.10686, 2023b. URL https://arxiv.org/pdf/2310.10686.

Zhihan Zhang, Tao Ge, Zhenwen Liang, Wenhao Yu, Dian Yu, Mengzhao Jia, Dong Yu, and Meng Jiang. Learn beyond the answer: Training language models with reflection for mathematical reasoning. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 14720–14738. Association for Computational Linguistics, 2024k. URL https://aclanthology.org/2024.emnlp-main.817.

Yu Zhao, Huifeng Yin, Bo Zeng, Hao Wang, Tianqi Shi, Chenyang Lyu, Longyue Wang, Weihua Luo, and Kaifu Zhang. Marco-o1: Towards open reasoning models for open-ended solutions. *ArXiv*, abs/2411.14405, 2024a. URL https://arxiv.org/pdf/2411.14405.

Zirui Zhao, Hanze Dong, Amrita Saha, Caiming Xiong, and Doyen Sahoo. Automatic curriculum expert iteration for reliable llm reasoning. *ArXiv*, abs/2410.07627, 2024b. URL https://arxiv.org/pdf/2410.07627.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Haotong Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. *ArXiv*, abs/2306.05685, 2023. URL https://arxiv.org/pdf/2306.05685.

Xin Zheng, Jie Lou, Boxi Cao, Xueru Wen, Yuqiu Ji, Hongyu Lin, Yaojie Lu, Xianpei Han, Debing Zhang, and Le Sun. Critic-cot: Boosting the reasoning abilities of large language model via chain-of-thoughts critic. *CoRR*, abs/2408.16326, 2024. doi: 10.48550/ARXIV.2408.16326. URL https://doi.org/10.48550/arXiv.2408.16326.

Han Zhong, Guhao Feng, Wei Xiong, Li Zhao, Di He, Jiang Bian, and Liwei Wang. Dpo meets ppo: Reinforced token optimization for rlhf. *ArXiv*, abs/2404.18922, 2024. URL https://arxiv.org/pdf/2404.18922.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning acting and planning in language models. *ArXiv*, abs/2310.04406, 2023. URL https://arxiv.org/pdf/2310.04406.

Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed H. Chi. Least-to-most prompting enables complex reasoning in large language models. *ArXiv*, abs/2205.10625, 2022. URL https://arxiv.org/pdf/2205.10625.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. Archer: Training language model agents via hierarchical multi-turn rl. *ArXiv*, abs/2402.19446, 2024. URL https://arxiv.org/pdf/2402.19446.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. Solving math word problems via cooperative reasoning induced language models. In *Annual Meeting of the Association for Computational Linguistics*, 2022. URL https://aclanthology.org/2023.acl-long.245.pdf.

Zheng Zhu, Xiaofeng Wang, Wangbo Zhao, Chen Min, Nianchen Deng, Min Dou, Yuqi Wang, Botian Shi, Kai Wang, Chi Zhang, Yang You, Zhaoxiang Zhang, Dawei Zhao, Liang Xiao, Jian Zhao, Jiwen Lu, and Guan Huang. Is sora a world simulator? a comprehensive survey on general world models and beyond. *ArXiv*, abs/2405.03520, 2024. URL https://arxiv.org/pdf/2405.03520.

Yuchen Zhuang, Xiang Chen, Tong Yu, Saayan Mitra, Victor S. Bursztyn, Ryan A. Rossi, Somdeb Sarkhel, and Chao Zhang. Toolchain*: Efficient action space navigation in large language models with a* search. *ArXiv*, abs/2310.13227, 2023. URL https://arxiv.org/pdf/2310.13227.