Breaking the Reasoning Barrier
A Survey on LLM Complex Reasoning through the Lens of Self-Evolution

Tao He[1*], **Hao Li**[1*], **Jingchang Chen**[1] **Runxuan Liu**[1] **Yixin Cao**[3], **Lizi Liao**[4], **Zihao Zheng**[1], **Zheng Chu**[1], **Jiafeng Liang**[1], **Ming Liu**[1,2†], **Bing Qin**[1,2]
[1]Harbin Institute of Technology, Harbin, China
[2]Pengcheng Laboratory, Shenzhen, China
[3]Institute of Trustworthy Embodied AI, Fudan University, Shanghai, China
[4]Singapore Management University, Singapore
{the, haoli, mliu, qinb}@ir.hit.edu.cn

Abstract

The release of OpenAI's O1 and subsequent projects like DeepSeek R1 has significantly advanced research on complex reasoning in LLMs. This paper systematically analyzes existing reasoning studies from the perspective of self-evolution, structured into three components: data evolution, model evolution, and self-evolution. Data evolution explores methods to generate higher-quality reasoning training data. Model evolution focuses on training strategies to boost reasoning capabilities. Self-evolution research autonomous system evolution via iterating cycles of data and model evolution. We further discuss the scaling law of self-evolution and analyze representative O1-like works through this lens. By summarizing advanced methods and outlining future directions, this paper aims to drive advancements in LLMs' reasoning abilities.

1 Introduction

Reasoning serves as the cornerstone in the domain of human cognition. The advent of OpenAI's O1 [OpenAI, 2024a] represents a transformative milestone in the study of complex reasoning for Large Language Models (LLMs), demonstrating the ability to generate intricate, human-like reasoning pathways [OpenAI, 2024b]. This breakthrough has ignited substantial interest across both industry and academia [Qin et al., 2024, Guan et al., 2025], driving efforts to replicate and extend its achievements. Industry projects, like DeepSeek R1 [DeepSeek-AI et al., 2025](shorted as R1) and Kimi-k1.5 [Team et al., 2025], release open-source models and pioneer advanced methodologies like Scaling Reinforcement Learning (RL). Concurrently, academic research, exemplified by OpenR [Wang et al., 2024e] and O1-Coder [Zhang et al., 2024j], delve into RL frameworks. Studies

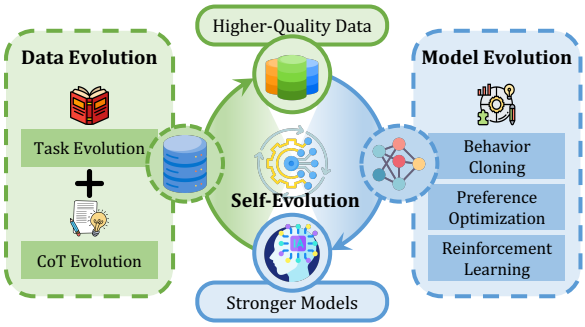


Figure 1: A conceptual framework for self-evolving complex reasoning capacities in LLMs. We identify three components in a complete self-evolution framework: data evolution, model evolution, and evolutionary strategies and patterns.

like Slow Thinking [Jiang et al., 2024b, Min et al., 2024] emphasize inference-time computing to enhance data quality. Furthermore, rStar-Math [Guan et al., 2025] achieves notable performance parity with O1 by leveraging a self-evolution framework, underscoring the potential of iterative optimization in advancing reasoning capabilities.

In this survey, we hope to systematically review existing studies to propel the advancement of complex reasoning in LLMs. The O1 blog and system card [OpenAI, 2024a,b] allude to the employment of RL and inference-time computing, drawing a compelling parallel to AlphaGo Zero [Silver et al., 2017], which achieved self-evolution through self-play and iterative improvement via Monte Carlo Tree Search (MCTS) [Browne et al., 2012] and policy networks. Inspiredly, super reasoning models may also be realized through self-evolution. Moreover, the scarcity of high-quality reasoning data highlights the urgent necessity for automated data synthesis frameworks [Sutskever, 2024, Wang et al., 2024f]. Self-evolution not only leverages synthetic data to elevate system performance but also enables improved systems to generate even higher-quality data, establishing a self-reinforcing cycle of continuous advancement.

[*] Equal contribution
[†] Corresponding Author.

This paper presents a systematic and comprehensive review of self-evolution for complex reasoning in LLMs. Self-evolution, or self-improvement, empowers LLMs to independently synthesize training data and progressively enhance their reasoning capabilities within a closed-loop system. For instance, methods based on online RL exemplify self-evolution, where the RL agent explores onlinely to collect high-quality trajectories and then improve itself through reward-guided optimization. This survey is organized into three core sections: **data evolution**, **model evolution**, and **self-evolution**. Data evolution is dedicated to the synthesis of high-quality data, encompassing two critical phases: (1) task evolution generates challenging tasks, and (2) CoT evolution pushes the boundaries of LLM performance via inference-time computing [Snell et al., 2024] and Chain-of-Thought (CoT) reasoning [Wei et al., 2022]. Model evolution optimizes modules of the reasoning system based on synthetic data. This involves training models on tasks that previously posed difficulties and selectively learning from curated datasets. Building upon iterative cycles of data evolution and model evolution, we analyze the theory behind self-evolution and focus on diverse evolutionary strategies and patterns.

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 5. (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 in 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. Therefore, the reasoning process can be formalized as: given a task q, the LLM generates a CoT y before predicting the final answer z.

2.2 Framework Elements

We first define four key components within a closed-loop self-evolution reasoning system:

Task Creator generates diverse tasks to prevent performance convergence and ensure continuous learning. Fixed task sets limit generalization, making task diversity critical for self-evolution.

Reasoner is the core module, implemented using an LLM, and generates step-by-step CoTs.

Evaluator assesses the Reasoner's CoTs. It provides feedback signals during training and guides reasoning and post-processor during inference.

Post-Processor post-process the CoTs based on feedback from Evaluator. It may filter or correct errors or apply other operations during generation.

These modules, though logically distinct, can be implemented using a single LLM due to its instruction-following capabilities. We explore their key roles in data evolution, model evolution, and self-evolution, respectively.

3 Data Evolution

As illustrated in Figure 2, this section describes how the reasoning system autonomously synthesizes higher-quality data. This process comprises two phases: task generation and CoT generation, corresponding to task evolution and CoT evolution.

3.1 Task Evolution

Task evolution focuses on generating more diverse and harder tasks, which is crucial yet underexplored in current self-evolution research. Analogous to the human learning process, we highlight three directions of task evolution for improving the models' reasoning and generalization capabilities.

Task Diversity Existing research on generating diverse tasks can be divided into two parts. The first focuses on prompting LLMs to directly generate tasks [Xu et al., 2023]. For example, Liu et al. [2023] use temperature sampling and diversity-focused prompts to enrich question generation. Wei et al. [2023] provide LLMs with diverse code snippets to inspire the generation of high-quality coding tasks. However, these approaches often produce low-quality outputs with a high proportion of invalid tasks. Other methods achieve diversity by rewriting existing reference tasks. Yu et al. [2023b] rephrase reference questions to generate new variants. [Haluptzok et al., 2022, Madaan et al., 2023a] further modify data types and logical operations to create structurally similar but logically distinct tasks. In summary, direct generation is simple but less reliable, whereas rewriting-based methods improve validity at the expense of diversity. Balancing effectiveness and diversity remains a key chal-

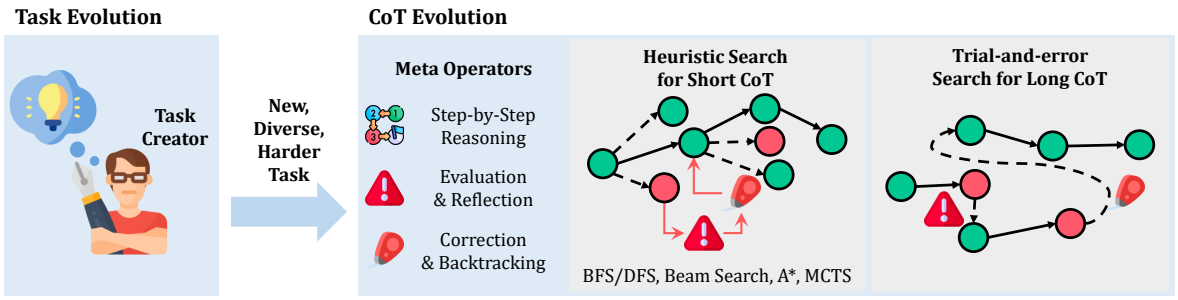


Figure 2: The pipeline of Data Evolution consists of Task Evolution and CoT Evolution. In CoT Evolution, we define three meta operators, which enable two search paradigms to generate higher-quality CoTs.

lenge in task generation research.

Task Complexity To generate more complex tasks, the most direct approach is to introduce additional conditions to the reference questions. For example, Shi et al. [2023] increase the difficulty of comprehension by incorporating irrelevant conditions, while Xu et al. [2023] adde extra constraints to heighten the complexity. In addition, increasing the number of reasoning steps has also emerged as a common strategy. Liu et al. [2024a] employ prompts to guide LLMs in generating new problems that include the original question as a subproblem or intermediate step. Beyond adding conditions and extending reasoning steps, Xu et al. [2023] further propose three distinct methods for rewriting questions. Mitra et al. [2024] relax restrictions on usable strategies by proposing a dual-agent collaborative framework: the Suggester agent analyzes a given problem and proposes ways to enhance its complexity, while the Editor agent modifies the problem based on the Suggester's recommendations.

Task Reliability Automatically generated tasks can sometimes be unsolvable or produce incorrect reference answers. To address this, Li et al. [2023a] fine-tune LLMs to rank and filter high-quality tasks, ensuring reliability. Meanwhile, Kreber and Hahn [2021] leverage GANs [Goodfellow et al., 2014] to synthesize tasks, assessing their similarity to real-world data for validity. [Haluptzok et al., 2022, Liu et al., 2023] ensure programming task quality by verifying correctness with Python interpreters and predefined rules. To ensure the accuracy, Lu et al. [2024b] propose two validation strategies: (1) using the reference answer as a condition to solve for the original problem's variables and (2) employing multiple reasoning approaches to solve the same problem.

3.2 CoT Evolution

Before starting the reasoning process, it's important to conceptualize what an effective reasoning CoT should look like and what main operations it should consist of. We further review inference-time computing methods, typically implemented via search, to generate higher-quality CoTs. These methods are categorized into **heuristic search** and **trial-and-error search**. Heuristic search uses tree search (mainly tree search) techniques to explore reasoning paths that each step is correct. Trial-and-error search leverages LLMs' self-evaluation and self-correction capabilities, enabling trial-and-error processes and backtracking upon detecting errors.

With the advent of R1 [DeepSeek-AI et al., 2025] and Kimi-k1.5 [Team et al., 2025], focus has shifted toward trial-and-error search. Analysis of O1's CoT examples reveals its ability to self-correct or backtrack upon detecting errors, mirroring human reasoning. This reasoning process has been further explored by Kimi-k1.5 [Team et al., 2025] and Redstar [Xu et al., 2025], who describe its outcome as Long CoT. In contrast, we refer to results of heuristic search as Short CoT. A detailed comparison between these two search methods and CoTs is provided in Appendix A.5.1.

3.2.1 Meta Operators

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] have sparked discussions on meta operators such as decomposition, verification, backtracking, and so on. Building on previous work [Qin et al., 2024, DeepSeek-AI et al., 2025], we summarize and categorize the following key meta operators that could exist in CoTs:

Step-by-Step Reasoning Step-by-step reasoning decomposes problems into sequential steps, solving them incrementally through chain-based reasoning.

CoT [Wei et al., 2022] reveals the capabilities of step-by-step thinking via in-context learning. Wang et al. [2023b] uses zero-shot prompting to generate a plan, followed by chain-based reasoning. Zhou et al. [2022] explicitly decomposes problems into sub-problems, solving them sequentially while incorporating previous outcomes. In contrast, Dua et al. [2022] iteratively decomposes and solves subproblems until completion. And ReACT [Yao et al., 2022] integrates iterative reasoning with actions, enabling interaction with external tools or environments to enhance reasoning.

Evaluation & Verification This meta-operator evaluates and verifies reasoning processes, guiding search, correcting errors, and optimizing outcomes. We systematically review existing research at three granularities: **outcome-level**, **step-level**, and **token-level**. Outcome-level methods evaluate the entire CoT but are often too coarse-grained. Token-level methods assess each token directly but face challenges in accuracy. Step-level methods, evaluating at step-level granularity, provide a balanced alternative. Due to space constraints, we focus on outcome-level evaluation here; step-level and token-level approaches are detailed in Appendix A.1.

Outcome-level Evaluation Early works primarily employed outcome-level evaluation, assessing complete solutions [Cobbe et al., 2021, Wang et al., 2023c, Lee et al., 2024a]. During training, some methods evaluate solutions using ground truth [Cobbe et al., 2021, Hosseini et al., 2024]. Besides answer correctness, R1 [DeepSeek-AI et al., 2025] and T1 [Hou et al., 2025] incorporate format-based outcome rewards to guide reasoning format learning. During inference, [Cobbe et al., 2021, Hosseini et al., 2024] use trained verifiers to score and rank solutions. Other methods [Madaan et al., 2023b, Zhang et al., 2024b] generate verbal feedback, playing the role of critic. [Peng et al., 2023, Shinn et al., 2023, Gou et al., 2024] incorporate both internal and external environmental information to implement critic. Furthermore, some combine verbal with score feedback for reliable evaluation [Ankner et al., 2024b, Yu et al., 2024b]. Consistency-based methods, such as Wang et al. [2023c], use voting or forward and backward reasoning [Jiang et al., 2024c, Weng et al., 2023] to assess answer quality.

Post-Processing Evaluated reasoning solutions often require further processing. For example, key knowledge can be extracted to aid subsequent reasoning, while low-quality solutions are typically handled through filtering or correction. Filtering removes unreliable solutions, whereas correction rectifies errors or reverts to a correct state. Due to space limitations, we focus on correction operations; for more post-processing strategies, please refer to Appendix A.3.

Correction Early research relies on models' intrinsic capabilities to refine solutions. For instance, Madaan et al. [2023b] iteratively improves outputs using self-generated feedback and Zhang et al. [2024g] aggregates differences across solutions into checklists to enhance self-reflection stability. Ramji et al. [2024] employs predefined metrics for iterative correction, and Wu et al. [2024c] trains a PSV model to identify and correct erroneous steps. Shridhar et al. [2024] uses an Asker model to generate sub-questions, guiding further corrections. To enhance critique capabilities, Zheng et al. [2024], Xi et al. [2024], Yan et al. [2024], Zhang et al. [2024i] train models to autonomously generate critical reviews, improving correction processes. The above iterative methods treat corrections as an MDP, where solutions are states and corrections are actions. Based on this, Zhang et al. [2024e,d] combine MCTS with self-correction. These works initialize a root node, select promising correction actions until leaf nodes, and conduct different corrections for expansion.

3.2.2 Heuristic Search for Short CoT

Heuristic Search for Short CoT employs heuristic approaches (usually tree-structured search) to explore multiple reasoning paths simultaneously, aiming to efficiently identify concise CoTs where each step is correct. Heuristic search algorithms, such as BFS/DFS, Beam Search, A*, and MCTS, underpin these methods. Due to space limitations, we focus on MCTS-based methods here. For additional works based on BFS/DFS, Beam Search, and A*, please see Appendix A.4.

Search with MCTS MCTS is a search algorithm that balances exploration and exploitation, showing strong performance in tasks modeled by MDPs [Wu et al., 2024a]. A notable application is AlphaGo Zero [Silver et al., 2017], which uses MCTS to search for optimal move actions, iteratively improving its policy network. Inspired by this, MCTS has been adapted for complex reasoning tasks. Please refer to Appendix A.5 for introduction of MCTS.

In the *Selection* phase of MCTS, the search is guided by the expected cumulative reward Q(s,a),

which can be assessed using the evaluation meta-operator discussed in Section 3.2.1 with the following specific means:

Self-evaluation Early research uses strong prior knowledge in LLM to evaluate reasoning states. For instance, MCTS-DPO [Xie et al., 2024] predicts correct option probabilities as rewards, and others use step generation probabilities as confidence measures [Hao et al., 2023].

Consistency Consistency-based methods evaluate answer quality through prediction consistency across samples or models. Inspired by Self-Consistency (SC) [Wang et al., 2023c], Zhou et al. [2023] apply SC metrics for evaluation. Alternatively, Feng et al. [2023] uses Jensen-Shannon divergence, and Qi et al. [2024] measures consistency between different models.

External Evaluator External model-based methods employ learned verifiers for evaluation. ReST-MCTS* [Zhang et al., 2024c] uses a learnable Process Reward Model (PRM) to assess step quality. Zhu et al. [2022] train path- and token-level scorers simultaneously.

Others Besides, more sophisticated evaluation methods are developed for MCTS. Xin et al. [2024] introduce intrinsic rewards for reward-free exploration in proof search. Many studies integrate multiple reward signals for improved accuracy: Feng et al. [2023] combines JS divergence, generation probability, and self-evaluation; Hao et al. [2023] integrates generation probability and self-evaluation; and Zhou et al. [2023] merges self-evaluation with self-consistency.

Other improvement works include enhancing policy learning efficiency through static and dynamic gap calculations on preference pairs collected by MCTS [Wang et al., 2024k]. Additionally, Xin et al. [2024] propose parallelized MCTS methods to improve search efficiency.

3.2.3 Trial-and-error Search for Long CoT

We have explored methods for searching Short CoT, where unpromising reasoning steps are passively pruned to prioritize more viable branches, ensuring only correct reasoning paths are retained. In contrast, Long CoT [Xu et al., 2025] mirrors human cognitive processes by linearizing complex search and leveraging LLMs' self-evaluation, self-correction, and backtracking mechanisms to dynamically adjust reasoning. This enables LLMs to engage in trial-and-error, guided by internal feedback. It manifests as simultaneously generating

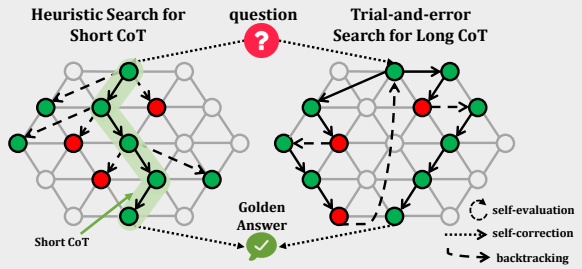


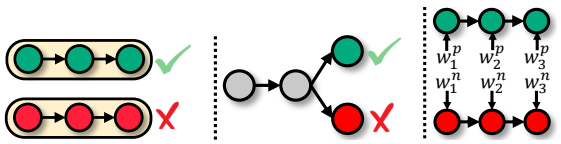
Figure 3: Comparison of Two Search Paradigms. Heuristic search follows hand-crafted rules—e.g., always expanding the branch with the highest score—and backtracks at the rule level when the current path proves unproductive. This approach aims to identify a short CoT that is correct at every step. In contrast, trial-and-error more closely resembles human problem solving: it explores the solution space through incremental attempts, where earlier mistakes serve as experience to guide future reasoning. This process ultimately yields a long CoT that captures the full trajectory of thought, including both errors and their corrections.

reasoning paths while subconsciously reflecting, verifying, and correcting errors, with past failed attempts serving as experiential guidance for subsequent reasoning processes, ultimately leading to the correct solution. We try to illustrate the differences between these two search paradigms in Figure 3. Unlike Short CoT, Long CoT does not demand perfection at every step, as LLMs can realign with the correct trajectory through internal self-reflection and self-correction mechanisms. Consequently, Long CoT demonstrates superior accuracy and robustness compared to Short CoT, enhancing LLMs' generalization when faced with novel and complex tasks. However, it imposes higher demands on LLMs, requiring not only step-by-step reasoning, verification, and correction but also the ability to actively adapt suitable reasoning strategies. For further detailed comparison of these two search paradigms, see Appendix A.5.1.

To acquire think in Long CoT, the O1 journey proposed a distillation method [Huang et al., 2024c]. However, simple distillation may fall short in cultivating efficient "slow thinking." To address this, DeepSeek-R1 [DeepSeek-AI et al., 2025] and Kimi-k1.5 [Team et al., 2025] independently proposed reinforcement learning-based training strategies. These approaches will be detailed in Section 5.

4 Model Evolution

After collecting high-quality reasoning data, the next step is to evolve modules in the system, lay-



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

Figure 4: Three granularities for preference optimization: Solution-level, Step-level, and Token-level.

ing a strong foundation for further data evolution. We focus on training the Reasoner, Evaluator, and Post-Processor modules. Before delving into specific model training work, we first introduce several representative RL algorithms in Appendix B.1 as foundational knowledge.

4.1 Reasoner Optimization

From the view of RL, we categorize existing works into Behavior Cloning, Preference Optimization, and Reinforcement Learning.

4.1.1 Behavior Cloning

After collecting reasoning process data, a straightforward approach is to optimize LLMs via Behavior Cloning (BC), i.e., Supervised Fine-Tuning (SFT). However, BC is typically limited to correct data. [Yuan et al., 2023a, Tong et al., 2024] employ rejection fine-tuning via filtering incorrect trajectories and fine-tuning on correct ones, while resulting in significant data wastage. To address this, Zelikman et al. [2022] regenerates reasoning processes for incorrect solutions, and Zhang et al. [2024c], Wang et al. [2024k] use MCTS to efficiently search for correct trajectories. Chen et al. [2024e] expands data by constructing reverse problems for SFT. Zhang et al. [2023a] introduces HIR, relabeling incorrect solutions (changing instructions for incorrect solutions) to utilize failure data without additional parameters. Despite these efforts, challenges remain in maximizing data utilization and effectively leveraging negative data.

4.1.2 Preference Optimization

Preference optimization is a widely used method to enhance LLM reasoning capabilities by promoting high-quality reasoning paths while demoting inferior ones. Early approaches, such as RRHF [Yuan et al., 2023b], employed ranking-based preference learning, using a reward model to rank responses and optimize the LLM via ranking loss. Other methods, notably DPO [Rafailov et al., 2023], simplify the RLHF process while addressing SFT limitations, gaining popularity due to their ease of implementation. Please refer to Appendix B.1.5 for details of DPO.

We classify preference optimization works into solution-level, step-level, and token-level optimizations and illustrate their differences in Figure 4. Due to space constraints, we focus on step-level methods here; for solution-level and token-level approaches, please refer to Appendix B.2.

Step-level Optimization Solution-level preference data is easy to obtain but suffers from coarse granularity. This limitation arises because errors in reasoning typically occur near the end, while preceding steps may remain correct. To address this, researchers develop step-level preference optimization methods. Based on constructing step-level training data, existing works can be classified into: **active construction** and **heuristic search**.

The active construction approach targets specific incorrect or correct reasoning steps with the same prefix. Hwang et al. [2024] uses MC sampling to identify the first error step in a incorrect CoT, then generates correct trajectories from the preceding context. The incorrect and correct pairs are used for preference optimization. Lai et al. [2024] employs GPT-4 to detect incorrect steps and uses DPO for step-level optimization. Lu et al. [2024c] generates incorrect steps by setting a high temperature to produce failure suffixes.

Heuristic tree search methods extract preference pairs from the searching tree. Zhang et al. [2024h] leverages ToT [Yao et al., 2023] with self-evaluation to identify correct reasoning paths, and filtered steps during search are viewed as negetive ones. MCTS-based approaches, like those by Xie et al. [2024] and Chen et al. [2024c], select preference pairs based on Q-value differences.

4.1.3 Reinforcement Learning

Model-free Online RL For deterministic tasks like mathematical reasoning, where environment dynamics are fixed and no external interaction is required, model-free online RL is highly effective. "Model-free" implies direct interaction with the environment, while "online" refers to training on data generated by the current policy interacting with the environment, avoiding distribution shifts common in offline learning. 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] show that

REINFORCE [Sutton et al., 1999] can achieve strong results without ORM or value models. ylfeng et al. [2024] and Zhang et al. [2024j] use PPO [Schulman et al., 2017] for solution- and step-level reasoning improvements, respectively. Zhong et al. [2024] integrate implicit rewards from DPO to guide PPO at the token level. Works like DeepSeek-Math [Shao et al., 2024], Qwen2.5-Math [Yang et al., 2024a], and OpenR [Wang et al., 2024e] employ PRM-based GRPO [Shao et al., 2024] to enhance multi-hop reasoning. Recently, R1 [DeepSeek-AI et al., 2025] and Kimi-k1.5 [Team et al., 2025] have achieved remarkable performance with ORM-based online RL, further demonstrating the immense potential of RL in LLM post-training.

We would like to emphasize significant works including R1 [DeepSeek-AI et al., 2025] and Kimi-k1.5 [Team et al., 2025]. Taking R1 as an example, this study abandons traditional PRM, instead relying solely on ORM-based online reinforcement learning, achieving remarkable breakthroughs in the process. This further underscores the immense potential of reinforcement learning in the post-training phase of LLMs. Specifically, R1 generates reasoning processes through self-exploration, which, as outlined in Section 3.2.3, encompass not only step-by-step logical derivations but also advanced behaviors such as self-reflection and self-correction. During training, R1 employs answer correctness and format as core reward metrics, encouraging LLM to learn correct reasoning pathways while discouraging those resulting in errors. If the reasoning process includes self-reflection or self-correction, these capabilities are also dynamically strengthened or weakened based on the final outcome rewards. Through this iterative mechanism of continuous exploration and learning, models like R1 progressively acquire the ability to generate Long CoT and demonstrate enhanced reasoning performance.

While model-free online RL has shown success, its limitations emerge in highly complex tasks, such as those with long reasoning chains or external environment interactions. In such scenarios, more diversified RL algorithms are expected to play a crucial role in post-training LLM optimization. Therefore, we also introduce two additional RL paradigms (model-based RL and hierarchical RL) for optimizing the reasoner under more complex scenarios in Appendix B.3.

4.2 Evaluator Optimization

In this section, we will focus on the methodology for constructing **token-level** data for evaluator optimization. Due to space limitations, the more important construction methods for **outcome-level** and **step-level** data will be detailed in Appendix B.4. Additionally, we discuss different training methods for the evaluator in Appendix B.5.

Token-level To obtain finer-grained token-level reward signals, automated methods for evaluating token importance are crucial. Chen et al. [2024i] trains a generative reward model to rewrite solutions. By inputting the solution into this reward model, the predicted probability of each original token serves as its reward. This approach leverages the intuition that incorrect tokens are likely to be modified, resulting in lower probabilities, while correct tokens remain consistent, yielding higher probabilities. Similarly, Yoon et al. [2024] employ a strong LLM to iteratively modify incorrect solutions through addition, deletion, and replacement operations. Token-level rewards are then assigned based on the differences between the modified and original solutions. In a different vein, [Rafailov et al., 2024, Zhong et al., 2024] derive implicit token-level rewards from the DPO framework, expressed as: $\beta \log \frac{\pi_{dpo}(y_t|x,y_{<t})}{\pi_{ref}(y_t|x,y_{<t})}$. Yang et al. [2024b] refines this approach, labeling the top $k\%$ of tokens in correct solutions with +1 and the lowest $k\%$ in incorrect solutions with -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 Post-Processor Optimization

Behavior Cloning Some methods [Zhang et al., 2024a, An et al., 2023, Yan et al., 2024, Paul et al., 2024, Gao et al., 2024c] use stronger models or multiple self-samples to generate correct solutions, then apply SFT to improve self-correction. Du et al. [2024] fine-tune models with a progressive dataset for iterative refinement. Others train auxiliary models, such as refiners [Welleck et al., 2023, Zhang et al., 2024i, Wadhwa et al., 2024] or an asker model [Shridhar et al., 2024] to assess and assist corrections. Wang et al. [2024l] introduce a codebook to store, retrieve, and apply reflective insights for guiding problem-solving.

Reinforcement Learning Kumar et al. [2024] identifies two challenges in SFT-based self-correction: distribution shift (struggling with self-

generated errors) and behavior collapse (over-optimizing initial outputs). To address this, Kumar et al. [2024], Gehring et al. [2024] employ the RL paradigm to learn this capability. Notably, recent studies like R1 [DeepSeek-AI 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.

5 Self-Evolution

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 8. However, self-evolution still poses specific implementation challenges during implementation, such as ensuring continuous improvement and coordinating modules co-evolution. We first introduce the convergence properties of self-evolution and propose a **Scaling Law** in Appendix C. Next, we survey self-evolution works from evolutionary strategies and patterns. Finally, we reinterpret representative O1-like works from a self-evolution perspective in Appendix D.

5.1 Self-Evolution Strategies

In the Preliminaries section, we defined four key modules and their interactions, framing the system as a multi-agent setup. Optimizing multiple agents together can enhance overall performance. Here, we summarize three multi-agent training strategies for reasoning systems.

Independent Evolution Early self-evolution systems often targeted single modules or ignored module interdependencies. For example, [Zelikman et al., 2022, Gulcehre et al., 2023] use golden answers to enhance the Reasoner, and [Hosseini et al., 2024] trains the Reasoner and Evaluator independently. Furthermore, [Madaan et al., 2023b, Wang et al., 2023d] correct solutions iteratively to improve quality without further Reasoner support. While independent evolution is easily implemented, its performance gains are limited.

Collaborative Evolution Joint evolution of multiple modules often involves leveraging cooperation to enhance overall system. Jiang et al. [2024b] uses the Reasoner to generate correct and incorrect solutions for training the reward model, which in turn filters solutions to train the Reasoner. Wang et al. [2024e] employs RL to train the entire system: the Evaluator provides reward signals to optimize the Reasoner, while the Reasoner collects data to train the Evaluator. More cooperation evolution could be further explored in reasoning systems.

Adversarial Evolution Another joint learning strategy is adversarial training [Goodfellow et al., 2014]. The competitive relationship between the Reasoner and Task Creator is obvious. Inspired by this, Ye et al. [2024] improve Reasoner via ReST [Gulcehre et al., 2023] under more challenging tasks, while the Task Creator generates harder tasks according to uncertainty feedback from Reasoner. Although adversarial relationships are harder to identify and optimize, it can mitigate the issue of local optima due to mutual catering among modules during collaborative evolution.

5.2 Self-Evolution Patterns

Only Reasoner Most methods focus on improving the Reasoner to enhance system performance, differing mainly in constructing training data. [Gulcehre et al., 2023, Min et al., 2024, Zelikman et al., 2022] use SFT on correct reasoning processes, while [Singh et al., 2023, Min et al., 2024, Pang et al., 2024] rely on golden answers for data selection. Without ground answers, [Huang et al., 2022, Li et al., 2024b] filter data based on consistency, and Song et al. [2024] use environmental rewards to achieve it. [Aksitov et al., 2023, Dong et al., 2023] employ reward models to rank trajectories for training. To increase positive data, Zelikman et al. [2022] re-answer failed questions given golden answers while Peng et al. [2024] provide answers only during abstract reasoning to avoid shortcuts. Additionally, [Chen et al., 2024b, Xie et al., 2024, Wang et al., 2024j,k] employ MCTS to search preference data for optimization.

Reasoner + Evaluator During self-evolution, the fixed Evaluator suffers from the weak generalization as the Reasoner's outputs deviate. Therefore, training the Reasoner and Evaluator simultaneously is essential.

Yuan et al. [2024c], Wang et al. [2024c] train the Reasoner using its generated correct solutions and use all solutions to train the reward model. Based on this, Jiang et al. [2024b] integrate the reward model into solution selection, adopting active learning to prioritize difficult cases. [Zhang et al., 2024c, Guan et al., 2025] employs MCTS for step-level action value estimates to train the

Evaluator, which then identifies high-quality reasoning trajectories for further Reasoner training. [Zhang et al., 2024j, Wang et al., 2024e] use RL to train the Reasoner, with a process reward model scoring reasoning steps to guide learning, fostering co-evolution between the Reasoner and Evaluator. Cheng et al. [2024], Chen et al. [2024j] employ adversarial training, where the Evaluator distinguishes responses from golden ones, while the Reasoner aims to confuse the Evaluator.

Reasoner + Post-Processor A stronger Reasoner decreases the pressure on the Post-Processor. Some works aim to improve both modules simultaneously. Dou et al. [2024] use the Reasoner to generate the initial solution and refine it. Then they SFT the Reasoner and Post-Processor using refined collect solutions. Wang et al. [2023d] employs multi-turn refinements until correctness, further using RL to train both two modules. Ultimately, both the Reasoner and Post-Processor improve concurrently.

Reasoner + Task Creator Finally, restricting evolution on fixed tasks may result in overfitting of Reasoner, reducing its ability to handle out-of-distribution tasks. Therefore, evolving the Task Creator is necessary. Just as we mentioned in self-evolution strategies, the adversarial relationship between the Reasoner and Task Creator is obvious. Ye et al. [2024] jointly trains the Reasoner and Task Creator based on adversarial principles, which have been discussed in the Adversarial Evolution of Section 5.1.

Reasoner + Evaluator + Post-Processor Evolving the Reasoner, Evaluator, and Post-Processor together can theoretically yield greater improvements than optimizing only one or two modules. Works like DeepSeek R1 [DeepSeek-AI et al., 2025] and Kimi-k1.5 [Team et al., 2025] exemplify this by optimizing reasoning, self-evaluation, reflection, and correction simultaneously via online RL, leading to co-evolution across all capacities and surpassing earlier methods focused on isolated module evolution.

6 Challenges and Future Directions

Despite the remarkable performance of open-source works like R1 on complex reasoning tasks, there are still some challenges from the self-evolution perspective that require further investigation. We discuss existing challenges in Appendix E and future directions in Appendix F.

7 Conclusion

This survey systematically reviews research on complex reasoning with LLMs from a higher view of self-evolution. We first explore data evolution and model evolution for complex reasoning, establishing the groundwork for self-evolution. Next, we have investigated self-evolution from the evolutionary strategies and patterns. Furthermore, we summarize O1-like open-source studies, demonstrating their alignment with our self-evolution framework. We hope this work inspires further advancements in LLM-based complex reasoning.

Limitations

This survey provides the first comprehensive study of self-evolution in complex reasoning for LLMs. Despite our best efforts, there may still be some limitations. On one hand, due to space constraints, we can only offer brief summaries of many methods rather than exhaustive technical details. Additionally, given the complexity of the technologies involved and our supplementary analyses, some content has been relegated to the appendix to preserve these insights, for which we apologize for any inconvenience caused during reading. On the other hand, the breadth of research areas covered means we cannot guarantee complete coverage of every subfield. While we have made every effort to collect relevant works, some important studies may have been overlooked. We will remain engaged with discussions in the research community and plan to update our perspectives and incorporate overlooked works in future revisions.

Acknowledgements

The research in this article is supported by the National Key Research and Development Project (2022YFF0903301), the National Science Foundation of China (U22B2059, 62276083).

References

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. 2024. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12248–12267, Bangkok, Thailand. Association for Computational Linguistics.

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. 2023. Rest meets react: Self-improvement for multi-step reasoning llm agent. *ArXiv*, abs/2312.10003.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2023. Learning from mistakes makes LLM better reasoner. *CoRR*, abs/2310.20689.

Marcin Andrychowicz, Dwight Crow, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Joshua Tobin, P. Abbeel, and Wojciech Zaremba. 2017. Hindsight experience replay. In *Neural Information Processing Systems*.

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

Zachary Ankner, Mansheej Paul, Brandon Cui, Jonathan D. Chang, and Prithviraj Ammanabrolu. 2024b. Critique-out-loud reward models. *CoRR*, abs/2408.11791.

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

Ralph Allan Bradley and Milton E. Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39:324.

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. 2012. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4:1–43.

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

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

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

Huayu Chen, Guande He, Lifan Yuan, Hang Su, and Jun Zhu. 2024d. Noise contrastive alignment of language models with explicit rewards. *ArXiv*, abs/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. 2024e. Reverse thinking makes llms stronger reasoners. *ArXiv*, abs/2411.19865.

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

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. 2024f. Do not think that much for 2+3=? on the overthinking of o1-like llms. *ArXiv*, abs/2412.21187.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, et al. 2024g. Do not think that much for 2+ 3=? on the overthinking of o1-like llms. *arXiv preprint arXiv:2412.21187*.

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

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

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

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024j. Self-play fine-tuning converts weak language models to strong language models. *ArXiv*, abs/2401.01335.

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

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

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. 2021. Training verifiers to solve math word problems. *CoRR*, 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. 2025. Process reinforcement through implicit rewards.

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, Shaokang 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. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning.

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

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

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. 2024. Think thrice before you act: Progressive thought refinement in large language models. *CoRR*, abs/2410.13413.

Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. 2022. Successive prompting for decomposing complex questions. In *Conference on Empirical Methods in Natural Language Processing*.

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

Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. 2025. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*.

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

Kanishk Gandhi, Denise Lee, Gabriel Grand, Muxin Liu, Winson Cheng, Archit Sharma, and Noah D. Goodman. 2024. Stream of search (sos): Learning to search in language. *ArXiv*, abs/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. 2024a. Llm critics help catch bugs in mathematics: Towards a better mathematical verifier with natural language feedback. *ArXiv*, abs/2406.14024.

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

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

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

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

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2024. 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.

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. 2024. A survey on llm-as-a-judge. *ArXiv*, abs/2411.15594.

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

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. 2023. Reinforced self-training (rest) for language modeling. *ArXiv*, abs/2308.08998.

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

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

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

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

Mingqian He, Yongliang Shen, Wenqi Zhang, Zeqi Tan, and Weiming Lu. 2024b. Advancing process verification for large language models via tree-based preference learning. In *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.

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

Jacob Hilton, Jie Tang, and John Schulman. 2023. Scaling laws for single-agent reinforcement learning. *ArXiv*, abs/2301.13442.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. *ArXiv*, abs/2403.07691.

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

Zhenyu Hou, Xin Lv, Rui Lu, Jiajie Zhang, Yujiang Li, Zijun Yao, Juanzi Li, Jie Tang, and Yuxiao Dong. 2025. Advancing language model reasoning through reinforcement learning and inference scaling.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024a. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024b. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Lei Huang, Weijiang Yu, Weitao Ma, Weihong Zhong, Zhangyin Feng, Haotian Wang, Qianglong Chen, Weihua Peng, Xiaocheng Feng, Bing Qin, and Ting Liu. 2023. A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. *ArXiv*, abs/2311.05232.

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

Wenyang Hui, Chengyue Jiang, Yan Wang, and Kewei Tu. 2024. Rot: Enhancing large language models with reflection on search trees. *CoRR*, abs/2404.05449.

Hyeonbin Hwang, Doyoung Kim, Seungone Kim, Seonghyeon Ye, and Minjoon Seo. 2024. Self-explore: Enhancing mathematical reasoning in language models with fine-grained rewards. In *Conference on Empirical Methods in Natural Language Processing*.

Dongwei Jiang, Jingyu Zhang, Orion Weller, Nathaniel Weir, Benjamin Van Durme, and Daniel Khashabi. 2024a. Self-[in]correct: Llms struggle with discriminating self-generated responses.

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. 2024b. Enhancing llm reasoning with reward-guided tree search.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T. Kwok. 2024c. Forward-backward reasoning in large language models for mathematical verification. In *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.

Fangkai Jiao, Chengwei Qin, Zhengyuan Liu, Nancy F. Chen, and Shafiq R. Joty. 2024. Learning planning-based reasoning by trajectories collection and process reward synthesizing. In *Conference on Empirical Methods in Natural Language Processing*.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*.

Ilya Kostrikov, Ashvin Nair, and Sergey Levine. 2021. Offline reinforcement learning with implicit q-learning. *ArXiv*, abs/2110.06169.

Jens Kreber and Christopher Hahn. 2021. Generating symbolic reasoning problems with transformer gans. *ArXiv*, 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. 2024. Training language models to self-correct via reinforcement learning. *CoRR*, abs/2409.12917.

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

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

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

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

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

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

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

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

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

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

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2023b. Making language models better reasoners with step-aware verifier. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5315–5333, Toronto, Canada. Association for Computational Linguistics.

Yunxin Li, Zhenyu Liu, Zitao Li, Xuanyu Zhang, Zhenran Xu, Xinyu Chen, Haoyuan Shi, Shenyuan Jiang, Xintong Wang, Jifang Wang, Shouzheng Huang, Xinping Zhao, Borui Jiang, Lanqing Hong, Longyue Wang, Zhuotao Tian, Baoxing Huai, Wenhan Luo, Weihua Luo, Zheng Zhang, Baotian Hu, and Min Zhang. 2025b. Perception, reason, think, and plan: A survey on large multimodal reasoning models. *arXiv preprint arXiv:2505.04921*.

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

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

Zhenwen Liang, Ye Liu, Tong Niu, Xiangliang Zhang, Yingbo Zhou, and Semih Yavuz. 2024. Improving llm reasoning through scaling inference computation with collaborative verification. *CoRR*, abs/2410.05318.

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

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

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

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

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

Rongxing Liu, Kumar Shridhar, Manish Prajapat, Patrick Xia, and Mrinmaya Sachan. 2024b. Smart: Self-learning meta-strategy agent for reasoning tasks. *ArXiv*, abs/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. 2024c. PRACT: optimizing principled reasoning and acting of LLM agent. *CoRR*, abs/2410.18528.

Zijun Liu, Peiyi Wang, Runxin Xu, Shirong Ma, Chong Ruan, Peng Li, Yang Liu, and Yu Wu. 2025. Inference-time scaling for generalist reward modeling. *arXiv preprint arXiv:2504.02495*.

Jianqiao Lu, Zhiyang Dou, Hongru Wang, Zeyu Cao, Jianbo Dai, Yingjia Wan, Yinya Huang, and Zhijiang Guo. 2024a. Autopsv: Automated process-supervised verifier.

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

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

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

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

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

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. 2025. Exploring the limit of outcome reward for learning mathematical reasoning.

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

Aman Madaan, Alex Shypula, Uri Alon, Milad Hashemi, Parthasarathy Ranganathan, Yiming Yang, Graham Neubig, and Amir Yazdanbakhsh. 2023a. Learning performance-improving code edits. *ArXiv*, 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. 2023b. Self-refine: Iterative refinement with self-feedback. In *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.

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

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

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. 2024. Imitate, explore, and self-improve: A reproduction report on slow-thinking reasoning systems.

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

Thomas M. Moerland, Joost Broekens, and Catholijn M. Jonker. 2020. Model-based reinforcement learning: A survey. *Found. Trends Mach. Learn.*, 16:1–118.

Todd K. Moon. 1996. The expectation-maximization algorithm. *IEEE Signal Process. Mag.*, 13:47–60.

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

OpenAI. 2024a. Learning to reason with llms.

OpenAI. 2024b. Openai o1 system card.

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. 2022. Training language models to follow instructions with human feedback. *ArXiv*, abs/2203.02155.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. *ArXiv*, abs/2404.19733.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2024. REFINER: reasoning feedback on intermediate representations. In *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.

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

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

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. *ArXiv*, abs/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. 2024. O1 replication journey: A strategic progress report - part 1. *ArXiv*, abs/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. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *ArXiv*, abs/2307.16789.

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. 2024. From r to q^*: Your language model is secretly a q-function.

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

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

Christopher D. Rosin. 2011. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61:203–230.

Schulman. 2020. Approximating kl divergence.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347.

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

Amrith Rajagopal Setlur, Chirag Nagpal, Adam Fisch, Xinyang Geng, Jacob Eisenstein, Rishabh Agarwal, Alekh Agarwal, Jonathan Berant, and Aviral Kumar. 2024. Rewarding progress: Scaling automated process verifiers for llm reasoning. *ArXiv*, abs/2410.08146.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Yu Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *ArXiv*, abs/2402.03300.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *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*.

Kumar Shridhar, Koustuv Sinha, Andrew Cohen, Tianlu Wang, Ping Yu, Ramakanth Pasunuru, Mrinmaya Sachan, Jason Weston, and Asli Celikyilmaz. 2024. The ART of LLM refinement: Ask, refine, and trust. In *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.

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. 2017. Mastering the game of go without human knowledge. *Nature*, 550:354–359.

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. 2023. Beyond human data: Scaling self-training for problem-solving with language models. *Trans. Mach. Learn. Res.*, 2024.

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

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

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel M. Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. 2020. Learning to summarize with human feedback. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

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

Ilya Sutskever. 2024. Sequence to sequence learning with neural networks: what a decade.

Richard S. Sutton, David A. McAllester, Satinder Singh, and Y. Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Neural Information Processing Systems*.

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

DeepSeek-AI Team. 2024a. Deepseek-v3 technical report.

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. 2025. Kimi k1.5: Scaling reinforcement learning with llms.

Qwen Team. 2024b. Qwq: Reflect deeply on the boundaries of the unknown.

Ye Tian, Baolin Peng, Linfeng Song, Lifeng Jin, Dian Yu, Haitao Mi, and Dong Yu. 2024. Toward self-

improvement of llms via imagination, searching, and criticizing. *CoRR*, abs/2404.12253.

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

Gladys Tyen, Hassan Mansoor, Victor Carbune, Peter Chen, and Tony Mak. 2024. Llms cannot find reasoning errors, but can correct them given the error location. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 13894–13908. Association for Computational Linguistics.

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

Manya Wadhwa, Xinyu Zhao, Junyi Jessy Li, and Greg Durrett. 2024. Learning to refine with fine-grained natural language feedback. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 12281–12308. Association for Computational Linguistics.

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

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

Huaijie Wang, Shibo Hao, Hanze Dong, Shenao Zhang, Yilin Bao, Ziran Yang, and Yi Wu. 2024b. Offline reinforcement learning for llm multi-step reasoning.

Jianing Wang, Yang Zhou, Xiaocheng Zhang, Mengjiao Bao, and Peng Yan. 2024c. Self-evolutionary large language models through uncertainty-enhanced preference optimization. *ArXiv*, abs/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. 2024d. Large language models for robotics: Opportunities, challenges, and perspectives. *ArXiv*, abs/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. 2024e. Openr: An open source framework for advanced reasoning with large language models. *ArXiv*, abs/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. 2024f. A survey on data synthesis and augmentation for large language models. *ArXiv*, abs/2410.12896.

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

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024g. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *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.

Shuhe Wang, Shengyu Zhang, Jie Zhang, Runyi Hu, Xiaoya Li, Tianwei Zhang, Jiwei Li, Fei Wu, Guoyin Wang, and Eduard H. Hovy. 2024h. Reinforcement learning enhanced llms: A survey.

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

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

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

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023c. 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.

Xuezhi Wang and Denny Zhou. 2024. Chain-of-thought reasoning without prompting. *ArXiv*, abs/2402.10200.

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

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language

models with self-generated instructions. In *Annual Meeting of the Association for Computational Linguistics*.

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. 2025. Thoughts are all over the place: On the underthinking of o1-like llms.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. 2024m. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 7309–7319. Association for Computational Linguistics.

Ziqi Wang, Le Hou, Tianjian Lu, Yuexin Wu, Yunxuan Li, Hongkun Yu, and Heng Ji. 2023d. Enabling language models to implicitly learn self-improvement.

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

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

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. 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.

Lilian Weng. 2024. Reward hacking in reinforcement learning. *lilianweng.github.io*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 2550–2575. Association for Computational Linguistics.

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

Jinyang Wu, Mingkuan Feng, Shuai Zhang, Ruihan Jin, Feihu Che, Zengqi Wen, and Jianhua Tao. 2025a. Boosting multimodal reasoning with mcts-automated structured thinking.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024b. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models.

Yuyang Wu, Yifei Wang, Tianqi Du, Stefanie Jegelka, and Yisen Wang. 2025b. When more is less: Understanding chain-of-thought length in llms.

Zhenyu Wu, Qingkai Zeng, Zhihan Zhang, Zhaoxuan Tan, Chao Shen, and Meng Jiang. 2024c. Enhancing mathematical reasoning in llms by stepwise correction. *CoRR*, abs/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. 2024. Enhancing llm reasoning via critique models with test-time and training-time supervision.

Shijie Xia, Xuefeng Li, Yixin Liu, Tongshuang Wu, and Pengfei Liu. 2024. Evaluating mathematical reasoning beyond accuracy. *ArXiv*, abs/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. 2024. Atomthink: A slow thinking framework for multimodal mathematical reasoning. *ArXiv*, abs/2411.11930.

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

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, James Xu Zhao, Min-Yen Kan, Junxian He, and Michael Qizhe Xie. 2023. Self-evaluation guided beam search for reasoning. In *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*.

Zhihui Xie, Jie chen, Liyu Chen, Weichao Mao, Jingjing Xu, and Lingpeng Kong. 2025. Teaching language models to critique via reinforcement learning.

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. 2024. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. *ArXiv*, abs/2408.08152.

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. *ArXiv*, 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. 2025. Redstar: Does scaling long-cot data unlock better slow-reasoning systems?

Yuchen Yan, Jin Jiang, Yang Liu, Yixin Cao, Xin Xu, Mengdi Zhang, Xunliang Cai, and Jian Shao. 2024. S^{3}c-math: Spontaneous step-level self-correction makes large language models better mathematical reasoners. *CoRR*, abs/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. 2024a. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement. *ArXiv*, abs/2409.12122.

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

Ling Yang, Zhaochen Yu, Bin Cui, and Mengdi Wang. 2025a. Reasonflux: Hierarchical llm reasoning via scaling thought templates.

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

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

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

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

Edward Yeo, Yuxuan Tong, Morry Niu, Graham Neubig, and Xiang Yue. 2025. Demystifying long chain-of-thought reasoning in llms.

Xunjian Yin, Xu Zhang, Jie Ruan, and Xiaojun Wan. 2024. Benchmarking knowledge boundary for large language models: A different perspective on model evaluation. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2270–2286, Bangkok, Thailand. Association for Computational Linguistics.

Zhangyue Yin, Qiushi Sun, Qipeng Guo, Jiawen Wu, Xipeng Qiu, and Xuanjing Huang. 2023. Do large language models know what they don't know? In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8653–8665, Toronto, Canada. Association for Computational Linguistics.

ylfeng, Yang Xu, Libo Qin, Yasheng Wang, and Wanxiang Che. 2024. Improving language model reasoning with self-motivated learning. *ArXiv*, abs/2404.07017.

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. 2024. Tlcr: Token-level continuous reward for fine-grained reinforcement learning from human feedback. *ArXiv*, abs/2407.16574.

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

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023a. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *NAACL-HLT*.

Long Long Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zheng Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models. *ArXiv*, 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. 2024b. Self-generated critiques boost reward modeling for language models. *ArXiv*, abs/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. 2024a. Advancing llm reasoning generalists with preference trees. *ArXiv*, abs/2404.02078.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. 2024b. Free process rewards without process labels.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. 2024c. Self-rewarding language models. *ArXiv*, abs/2401.10020.

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

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

E. Zelikman, Yuhuai Wu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning.

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

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

Bohan Zhang, Xiaokang Zhang, Jing Zhang, Jifan Yu, Sijia Luo, and Jie Tang. 2025a. Cot-based synthesizer: Enhancing llm performance through answer synthesis.

Che Zhang, Zhenyang Xiao, Chengcheng Han, Yixin Lian, and Yuejian Fang. 2024a. Learning to check: Unleashing potentials for self-correction in large language models.

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

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

Di Zhang, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024d. Accessing gpt-4 level mathematical olympiad solutions via monte carlo tree self-refine with llama-3 8b. *ArXiv*, abs/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. 2024e. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. *CoRR*, abs/2410.02884.

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

Tianjun Zhang, Fangchen Liu, Justin Wong, P. Abbeel, and Joseph E. Gonzalez. 2023a. The wisdom of hindsight makes language models better instruction followers. In *International Conference on Machine Learning*.

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. 2024g. Self-contrast: Better reflection through inconsistent solving perspectives. In *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.

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

Yunxiang Zhang, Muhammad Khalifa, Lajanugen Logeswaran, Jaekyeom Kim, Moontae Lee, Honglak Lee, and Lu Wang. 2024i. Small language models need strong verifiers to self-correct reasoning. In *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.

Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong, and Jitao Sang. 2024j. o1-coder: an o1 replication for coding.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025b. The lessons of developing process reward models in mathematical reasoning.

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

Zhihan Zhang, Tao Ge, Zhenwen Liang, Wenhao Yu, Dian Yu, Mengzhao Jia, Dong Yu, and Meng Jiang. 2024k. Learn beyond the answer: Training language models with reflection for mathematical reasoning. In *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.

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

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

Yuxiang Zheng, Dayuan Fu, Xiangkun Hu, Xiaojie Cai, Lyumanshan Ye, Pengrui Lu, and Pengfei Liu. 2025. Deepresearcher: Scaling deep research via reinforcement learning in real-world environments. *arXiv preprint arXiv:2504.03160*.

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

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

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

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

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. 2022. Solving math word problems via cooperative reasoning induced language models. In *Annual Meeting of the Association for Computational Linguistics*.

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. 2024. Is sora a world simulator? a comprehensive survey on general world models and beyond. *ArXiv*, abs/2405.03520.

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

A Data-Evolution Appendix

A.1 More Evaluation Granularity

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] employs state scoring within MCTS to guide the search process, whereas Xie et al. [2023] implements 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 remains insufficient for comprehensive reasoning assessment [Yoon et al., 2024, Chen et al., 2024i]. This has spurred the development of token-level evaluation frameworks, which offer finer-grained analysis. Yoon et al. [2024] introduces 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. [2024i] proposes 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. [2024c] proposes a token-supervised value model, which supervises individual tokens to provide a more accurate assessment of solution correctness. Meanwhile, Yang et al. [2024b] derives 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.

A.2 Evaluation Categories

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] employs a verifier to estimate the probability of a solution being correct, while Hosseini et al. [2024] leverages 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., 2024i, Lee et al., 2024c, 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 correction and enhance the interpretability of scoring mechanisms. For instance, Madaan et al. [2023b] exploits 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

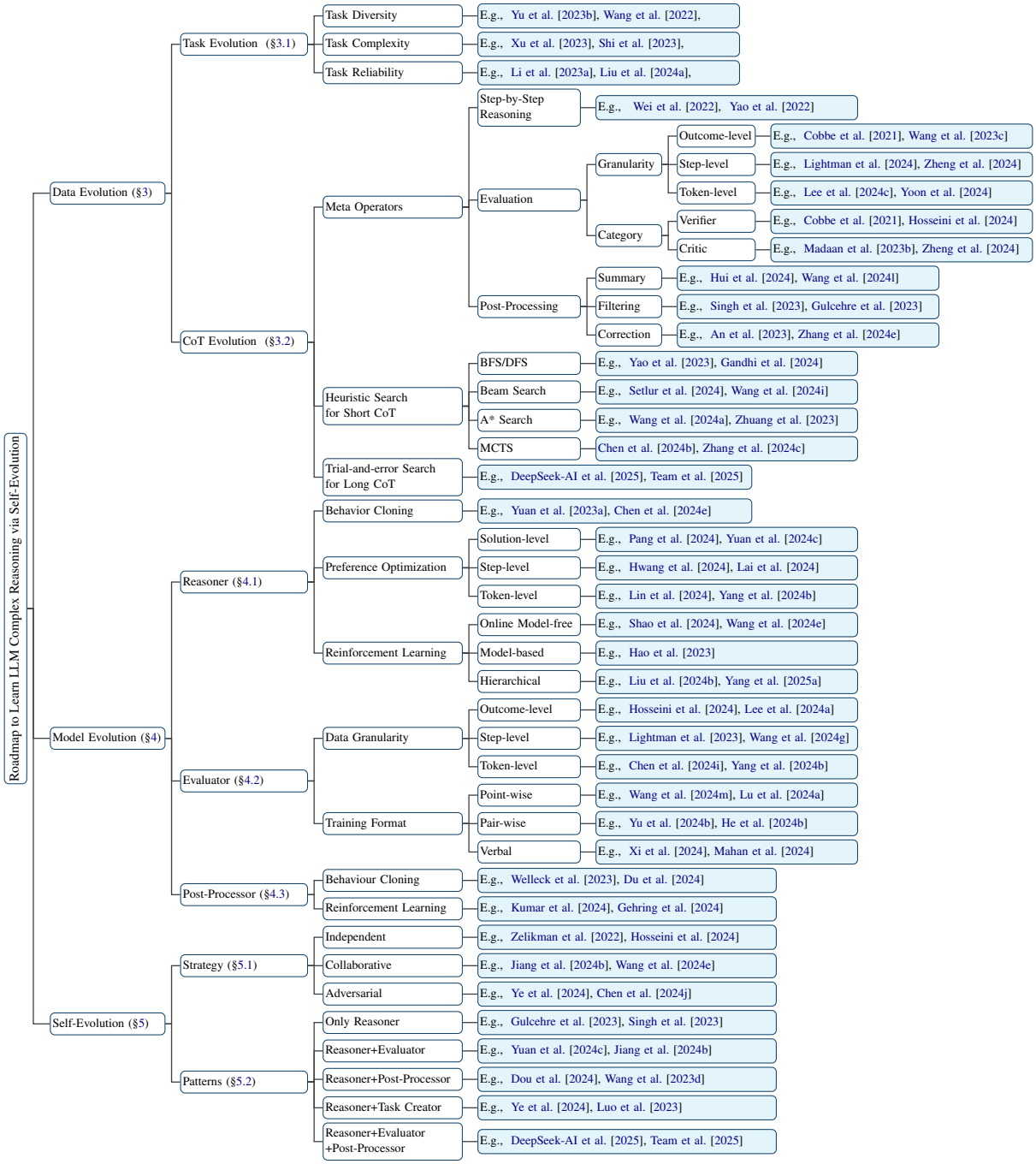


Figure 5: Taxonomy of Advanced Methods, including data evolution, model evolution, and self-evolution.

robust framework for assessing solution quality.

A.3 More Post-Processing Methods

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] integrates 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] identifies 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] introduces textual principles for action selection, continually refining these principles through iterative reflec-

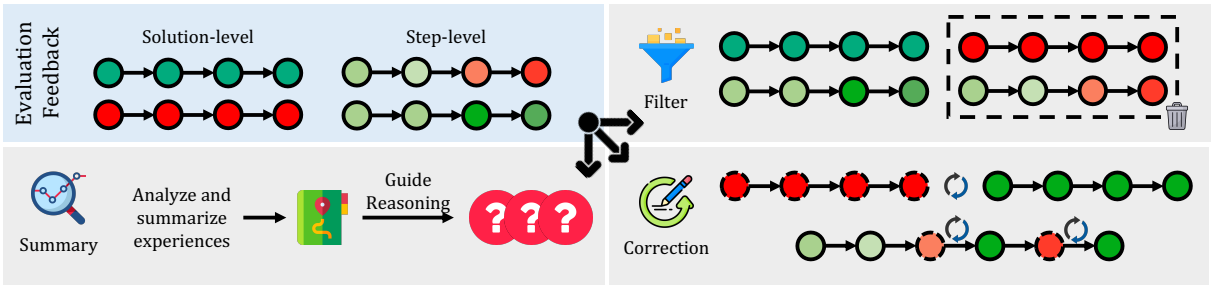


Figure 6: Three post-processing methods following evaluation: Filter, Summary, and Correction.

tion to flexibly guide action execution. Additionally, Zhang et al. [2025a] proposes 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., 2023], forward-backward consistency [Jiang et al., 2024c, 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.

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 performed: when an error is detected, the system can backtrack to a previous state and explore alternative reasoning paths.

A.4 More Tree Search Methods

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] combines 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. [2024d] introduces 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] proposes 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 incorporates process advantages to refine the search process. In contrast to external evaluation methods, Xie et al. [2023] utilizes 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] highlights 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] observes 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] proposes 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] introduces 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. [2024] proposes 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] trains 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] introduces Searchformer, which tokenizes A* execution traces and bootstraps a Transformer model to iteratively sample shorter paths. Similarly, Su et al. [2024] proposes Dualformer, which incorporates random information dropout during A* search, enabling the model to balance fast and slow thinking for improved search strategies.

A.5 Brief Introduction of MCTS

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 ac-

cumulated 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.

A.5.1 Comparison and Correlation between Heuristic & Trial-and-error Searches

Comparison of Heuristic & Trial-and-error Searches

Before delving into a detailed comparison, we briefly summarize the procedural distinctions between heuristic search for Short CoT and trial-and-error search for Long CoT. As shown in Figure 3, heuristic search (mainly tree search, 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 operations like evaluation, pruning, and error correction are not presented in the CoT. In contrast, 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 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, heuristic 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 performance [Yeo et al., 2025].

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

- First, heuristic 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 heuristic search only reflect the relative quality of reasoning, failing to pinpoint errors or their 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 heuristic tree search can explore multiple paths simultaneously, these paths are independent. Thus, intermediate experience cannot be shared across them, lowering the utilization of parallel reasoning processes. This makes heuristic 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 heuristic 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 implicit 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., 2024g], approaches such as QwQ [Team, 2024b] and R1 [DeepSeek-AI 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. [2025] observes 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] 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 implicit search not only increases token usage and computational costs but also degrades performance.
- Furthermore, implicit 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, kimi-k1.5, and T1 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] proposes the Length-Harmonizing Reward to suppress excessively long responses. In addition to introducing new reward functions, Chen et al. [2024f] employed 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. However, these methods have not specifically focused on directly enhanc-

ing self-evaluation and self-correction capabilities, leaving their overall effectiveness to be further evaluated.

A.5.2 Correlation and Unification between Heuristic & Trial-and-error Searches

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

First, from the perspective of the search action space, heuristic tree search focuses on exploring action space defined by individual reasoning steps. In contrast, trial-and-error search expands the action space of heuristic tree search with other meta operators, such as evaluation, verification, correction, and backtracking. In other words, with this extension, heuristic tree search can also explore Long CoT. The reduced dimensionality in heuristic tree search might be one of the reasons why existing heuristic search methods are less generalizable than trial-and-error search ones.

From the perspective of reasoning capability evolution, 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 more 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. Therefore, Long CoT can be viewed as an initial and intermediate solution for handling complex tasks. Once a task is solved, the knowledge extracted 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 dual capabilities of Long CoT and Short CoT, enabling it to adaptively balance exploration and efficiency in problem-solving.

B Model-Evolution Appendix

B.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.

B.1.1 Set Sail from RLHF

Given the tremendous success of products like ChatGPT and Claude, 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:

$$\max_\theta \mathbb{E}_{(x,y^w,y^l)\in\mathcal{D}}[\log(\sigma(r_\theta(x,y^w) - r_\theta(x,y^l)))], \tag{1}$$

  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:

$$E_{x\sim D, y\sim\pi_\phi(.|x)}[r_\theta(x,y) - \beta D_{KL}(\pi_\phi(y|x)||\pi_{ref}(y|x))], \tag{2}$$

  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.

B.1.2 From RLHF to more fine-grained PPO

Although RLHF uses PPO for optimization, in practice, classic RLHF is often considered a bandit approach because it relies only on outcome-level rewards, lacking finer-grained optimization signals. Therefore, classic RLHF can be understood as treating the entire sentence as a single action [Zhong et al., 2024].

As is well known, 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 does 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:

$$\mathbb{E}_{q\sim P(Q),y\sim\pi_\phi(y|q)}\frac{1}{|y|}\sum_{t=1}^{|y|}\min[\frac{\pi_\phi(y_t|q,y_{<t})}{\pi_{ref}(y_t|q,y_{<t})}A_t,$$
$$\text{clip}(\frac{\pi_\phi(y_t|q,y_{<t})}{\pi_{ref}(y_t|q,y_{<t})},1-\epsilon,1+\epsilon)A_t], \tag{3}$$

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 used, which balances the trade-off between bias and variance in the estimation:

$$\begin{aligned} A_t^{GAE} &= \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1} \\ \delta_t &= r_t + \gamma V(s_{t+1}) - V(s_t) \\ &= Q(s_t, a_t) - V(a_t), \end{aligned} \tag{4}$$

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 and low training efficiency 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 from the RL perspective, 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.

B.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:

$$\sum_{t=1}^{T}\mathbb{E}_{a_t\sim\pi_\phi(a_t|s_t)}[R(s_t, a_t)\log\pi_\phi(a_t|s_t)], \tag{5}$$

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:

$$\mathbb{E}_{a_t\sim\pi_\phi(.|s_t)}[(R(s_t, a_t) - b)\log\pi_\phi(a_t|s_t)]. \tag{6}$$

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:

$$b(s_t) = r(s_t, a_t), \quad a_t \in \arg\max \pi_\phi(.|s_t). \quad (7)$$

[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:

$$b(r^i) = \frac{1}{k-1} \sum_{j \neq i} R(r^j, x). \quad (8)$$

[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.

B.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. 4, 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:

$$\begin{aligned} \tilde{A}_{i,t} &= \sum_{index(j) \geq t} \tilde{r}_i^{index(j)}, \\ s.t. \quad \tilde{r}_i^{index(j)} &= \frac{r_i^{index(j)} - \text{mean}(R)}{\text{std}(R)}. \end{aligned} \quad (9)$$

When the 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:

$$\tilde{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(R)}{\text{std}(R)}. \quad (10)$$

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:

$$\begin{aligned} &\mathbb{E}[q \sim P(Q), \{y_i\}_{i=1}^G \sim \pi_{\phi_{old}}(y_i|q)] \\ &\frac{1}{G} \sum_{i=1}^{G} \frac{1}{|y_i|} \sum_{t=1}^{|y_i|} \Bigg\{ \min[\frac{\pi_\phi(y_{i,t}|q, y_{i,<t})}{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})} \tilde{A}_{i,t}, \\ &\quad \text{clip}(\frac{\pi_\phi(y_{i,t}|q, y_{i,<t})}{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})}, 1-\epsilon, 1+\epsilon) \tilde{A}_{i,t}] \\ &\quad - \beta \mathbb{D}_{KL}[\pi_\theta || \pi_{ref}] \Bigg\}, \end{aligned} \quad (11)$$

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:

$$\begin{aligned} \mathbb{D}_{KL}[\pi_\phi || \pi_{\phi_{old}}] = &\frac{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})}{\pi_\phi(y_{i,t}|q, y_{i,<t})} \\ &- \log \frac{\pi_{\phi_{old}}(y_{i,t}|q, y_{i,<t})}{\pi_\phi(y_{i,t}|q, y_{i,<t})} - 1. \end{aligned} \quad (12)$$

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 1: 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.

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].

B.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 2:

$$\pi^*(y|x) = \frac{1}{Z(x)}\pi_{ref}(y|x)\exp(\frac{1}{\beta}r(x,y)). \quad (13)$$

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 13, leading to a new expression:

$$r(x,y) = \beta\log\frac{\pi^*(y|x)}{\pi_{ref}(y|x)} + \beta\log Z(x). \quad (14)$$

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:

$$\mathbb{E}_{(x,y_w,y_l)\in D, y_w\succ y_l}[\sigma(r_\theta(x,y_w) - r_\theta(x,y_l))]. \quad (15)$$

By bringing Equation 14 into Equation 15, the objective of reward modeling can be transformed into directly learning the policy model:

$$\mathbb{E}_{(x,y_w,y_l)\in D}[\log\sigma(\beta\log\frac{\pi_r(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_r(y_l|x)}{\pi_{ref}(y_l|x)})]. \quad (16)$$

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 4.1.2.

- **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., 2024a, 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.

B.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:

$$r(x, y) = \beta \log \frac{\pi_\theta(x, y)}{\pi_{ref}(y|x)}. \tag{17}$$

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., 2024b] proof 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:

$$r_o(y) \cdot \log \sigma(r_\theta(y)) + (1 - r_o(y)) \cdot \log(1 - \sigma(r_\theta(y))), \tag{18}$$

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:

$$r_\theta(y_t) = \beta \log \frac{\pi_\theta(y_t|x, y_{<t})}{\pi_{\text{ref}}(y_t|x, y_{<t})}, \tag{19}$$

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.

B.2 More Preference Optimization Paradigms for Reasoner

Solution-level 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. [2024b] 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. [2024c] utilizes 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] proposes 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.

Token-level 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 Appendix B.1.5. Notably, as highlighted in the O1 blog [OpenAI, 2024b], achieving substantial improvements in complex reasoning capabilities may ultimately necessitate the integration of RL techniques [DeepSeek-AI et al., 2025], underscoring the need for more advanced optimization frameworks.

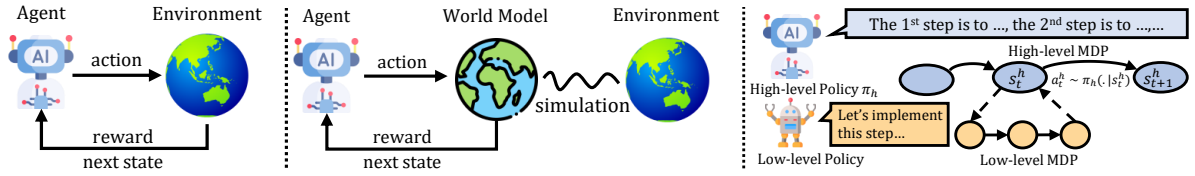
B.3 More RL Paradigms for Reasoner

B.3.1 Model-based RL

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] demonstrates that LLMs can serve as world models for planning tasks and He et al. [2024c] applies 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.



(a) Online Model-free RL (b) Model-based RL (c) Hierarchical RL

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

B.3.2 Hierarchical RL

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 optimizes the high-level MDP using policy gradient algorithms [Lee et al., 2024d], it does not address 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.

B.4 More Training Data Construction for Evaluator

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-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 is clearly impractical, necessitating more automated labeling methods. 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. [2024h] decompose problems into subproblems and directly extracted 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 prompt 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.

B.5 Training Format for Evaluator

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 utilizes 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:

$$\max \mathbb{E}_{(x,y^+,y^-)\in D} \log(\sigma(r(x, y^+) - r(x, y^-))), \tag{20}$$

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] classifies solutions sampled from multiple LLMs into preference pairs based on answer correctness, and train reward models using SimPO.

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.

Verbal Recognizing the limitations of models in self-correction [Huang et al., 2024a], Xi et al. [2024] introduce an effective method for error localization in responses. Their approach involves deliberately introducing errors at specific reasoning steps to construct flawed reasoning paths, followed by generating step-level critiques based on these errors. These critique data are then used to fine-tune a critique model via SFT, enabling it to identify erroneous steps in responses. In code generation tasks, Xie et al. [2025] proposes leveraging RL to develop the ability to evaluate reasoning processes and generate verbal feedback. This approach enables iterative evaluation and refinement until reasoning is achieved.

In reward model optimization, leveraging natural language feedback has emerged as a key strategy for enhancing scoring reliability and interpretability. One approach extracts generation probabilities of specific tokens from natural language feedback as the basis for scoring. For example, Mahan et al. [2024], Zhang et al. [2024f] fine-tune models to perform next-token prediction, generating scores that not only improve interpretability but also effectively guide model optimization.

Another approach integrates natural language critiques to assist in score generation. Ankner et al. [2024b], Gao et al. [2024a] propose training models to first generate critiques and then refine reward models to produce scores based on these critiques. By incorporating natural language critiques, this method significantly enhances the reliability and interpretability of scoring, offering a more transparent and robust evaluation framework.

C 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) Can the self-evolution architecture converge? (2) Does self-evolution follow a Scaling Law?

[RQ1] Can the self-evolution architecture converge?

To address the first question, Singh et al. [2023] provide an explanation from the perspective of Expectation Maximization (EM) [Moon, 1996]. Specifically, this work formulates the reasoning task as $p(a = \hat{a}|x)$, where x represents the question and $\hat{a}$ 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., $a = \hat{a}$. The final optimization objective is formulated as:

$$\begin{aligned}
&\max_{\phi} \log p_\phi(O=1|x) \\
&= \max_{\phi} \log \mathbb{E}_{q(y|x)}\left[\frac{p_\phi(O=1,y|x)}{q(y|x)}\right] \\
&\geq \max_{\phi} \mathbb{E}_{q(y|x)} \log\left[\frac{p_\phi(O=1,y|x)}{q(y|x)}\right] \\
&= \max_{\phi} \mathbb{E}_{q(y|x)} \log\left[\frac{p_\phi(O=1,y|x)}{q(y|x)}\right] \\
&= \max_{\phi} -\mathbb{D}_{KL}[q(y|x)||p_\phi(O=1,y|x)] \\
&= \max_{\phi} \mathbb{E}_{q(y|x)}[p_\phi(O=1|x,y)] \\
&\quad - \mathbb{D}_{KL}[q(y|x)||p_\phi(y|x)].
\end{aligned} \tag{21}$$

For optimization problems involving latent variables, the Expectation-Maximization (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:

$$\begin{aligned}
&\min_{\phi} \mathbb{D}_{KL}(q(y|x)||p_\phi(y|x)) \\
&\quad = \max_{\phi} q(y|x) \log p_\phi(y|x).
\end{aligned} \tag{22}$$

Therefore, the M-Step aims to train the reasoning model $p_\phi(y|x)$ using the generated data, leveraging

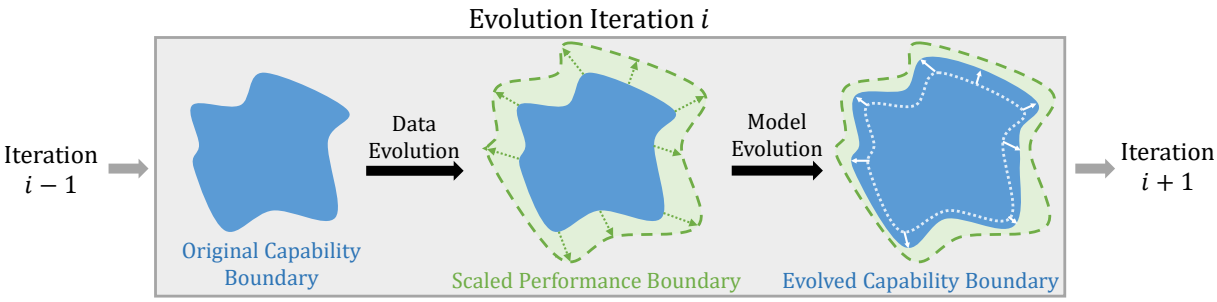


Figure 8: An intuitive understanding of self-evolution. Each iteration consists of data evolution and model evolution: the system first achieves a scaled performance boundary beyond its original capability boundary through data evolution (primarily via search), presented as higher-quality solutions serving as training data. Subsequently, the system expands its capability boundary by learning from these data through model evolution.

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.

[RQ2] Does self-evolution follow a Scaling Law?

While we aspire reasoning systems to achieve continuous self-evolution, a practical and unavoidable question arises: does self-evolution have an inherent upper limit—in other words, does self-evolution also follow a scaling law? To explore this question, we conducted a thought experiment. We begin by acknowledging that self-evolution encompasses multiple dimensions, including task evolution, CoT evolution, model evolution, and diverse evolutionary strategies and patterns. As such, we hypothesize that the potential for self-evolution is governed by the synergistic interaction of these four components. To formalize this relationship, we propose a concise formula to quantify the potential for self-evolution:

$$S_{\text{self-evolution}} = \min(S_{\text{task}} \times S_{\text{CoT}}, S_{\text{model}}) \times S_{\text{strategies \& patterns}}. \quad (23)$$

Here, S_{task}, S_{CoT}, S_{model}, and $S_{\text{strategies \& patterns}}$ denote the size of the task space, the evolutionary potential of CoT, the evolutionary potential of model capabilities, and the potential gains derived from employing diverse evolutionary strategies and patterns, respectively. The inclusion of the "min" function due to the inherent limitation of a model's capacity (determined primarily by its parameter count and architecture). Beyond this capacity, even an increase in data may not enable the model to learn additional reasoning patterns. On one hand, the number of modules within a system and the diversity of evolutionary strategies and patterns are inherently limited, suggesting that the potential for improvement through these mechanisms is likewise constrained. On the other hand, CoT evolution and model evolution align with inference-time computing and post-training, respectively.

The scaling law for inference-time computation has been empirically validated, indicating that the gains from scaling inference-time computation possess inherent limits [Wu et al., 2024b]. However, whether a scaling law exists for post-training remains an open question. OpenAI has previously investigated scaling laws in the context of RL [Hilton et al., 2023], while focusing on traditional RL tasks, such as Dota2, rather than LLM reasoning tasks. Zeng et al. [2024b] also discussed the scaling law for RL in LLMs, but without definitive conclusions being reached. If a post-training scaling law exists—implying that post-training also has an upper limit—then self-evolution would similarly be governed by a scaling law. Conversely, if no such scaling law exists for post-training, the upper bound of self-evolution would be limited by the $S_{\text{task}} \times S_{\text{CoT}}$. At this situation, whether self-evolution has an upper bound may be determined by task evolution. However, the existence of the scaling law of task evolution remains an open question and requires further investigation. Of course, this remains merely a hypothesis, and definitive conclusions will require further experimental validation.

Assuming our perspective holds, the remaining critical question is how to achieve effective self-evolution. Zeng et al. [2024a] notes 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] analyzes 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] proposes 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 convergence performance of self-evolution is the diversity of reasoning trajectories encountered by the LLM during training. When this diversity decreases, the evolution also diminishes. Therefore, in addition to explicitly enhancing the exploration of the reasoner model, possible solutions include: 1) increasing task diversity and difficulty could directly enhances reasoning trajectory diversity and improves system generalization [Li et al., 2024a]; 2) enhancing the system's self-evaluation and self-correction abilities can significantly improve the system's robustness and generalization.

D 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., 2024] 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., 2024c] 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 trial-and-error search capabilities.

Slow Thinking with LLMs

***Part 1*:** Part 1 of Slow Thinking [Jiang et al., 2024b] 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 computing. Subsequently, DPO is applied to optimize both the reasoner and evaluator. The MCTS-based inference-time computing represents an heuristic tree search within 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 the deeply implicit 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 heuristic 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 works, 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 5.2. This holistic evolution of three components explains why R1 and Kimi-k1.5 significantly outperform previous systems that evolved only one or two modules.

E Challenges for Research Fields

E.1 Challenges for Evaluation

E.1.1 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].

E.1.2 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.

E.2 Challenges for Post-Processing

E.2.1 Theory behind Self-Correction Mechanism

Early research indicates that in-context learning alone struggles to enable self-correction in LLMs [Huang et al., 2024b, Tyen et al., 2024, Jiang et al., 2024a]. To address this, training-based methods have been explored, such as supervised fine-tuning (SFT) on curated datasets [Yan et al., 2024]. However, some studies argue that SFT alone is insufficient, and reinforcement learning (RL) may be a more effective alternative [Kumar et al., 2024], which we will discuss in Section 4. Nevertheless, a key unresolved question is why a model requiring correction can produce improved answers—suggesting it already possesses the necessary knowledge. One explanation is the knowledge boundary theory [Yin et al., 2023, Huang et al., 2023], which posits a gap between stored knowledge and its effective expression. Corrections, through reflection or feedback, may help surface latent knowledge [Yin et al., 2024]. Understanding this could clarify the limits of LLMs' self-correction. Furthermore, a critical challenge remains: how to enable models to fully utilize their knowledge during initial generation, reducing reliance on correction. Addressing this is an urgent research priority.

E.3 Challenges for Self-Evolution

E.3.1 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.

E.3.2 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.

F Future 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. Current methods remain relatively simplistic, and further research is needed to generate more diverse, complex, and effective tasks.

- **How can we further enhance the self-evaluation and self-correction capabilities of LLMs during trial-and-error search?** Works like R1 do not explicitly train for self-reflection abilities; instead, they rely on outcome rewards to guide the learning of step-by-step reasoning and self-reflection. This approach results in the final reward being granted as long as the final answer is correct, even if errors are repeatedly encountered during the process. This may explain why tokens of Long CoT in R1 and similar works gradually increase over the training steps. The key to addressing this challenge lies in improving the efficiency of self-evaluation and

self-correction while simultaneously advancing overall reasoning performance.

- **How can we develop more effective reward modeling?** Recent work (e.g., R1) has shown that using ORM alone can lead to strong reasoning performance. R1 also presents a detailed failure analysis of MCTS+PRM, casting doubt on the practical effectiveness of PRM. Compared to learnable PRMs, the rule-based ORM used in R1 offers clear benefits in generalization and robustness against reward hacking. However, R1 also observes that the number of generated tokens increases over training, raising questions about whether frequent self-reflection truly helps extend the CoT [Chen et al., 2024f, Wang et al., 2025]. A plausible explanation is that ORM rewards only the correctness of the final output, without shaping the intermediate steps. These limitations of PRM—poor generalization and susceptibility to reward hacking—highlight an open challenge: how can we design a more reliable and effective PRM?

  Moreover, the rule-based ORM in R1 is evaluated primarily on tasks like math reasoning and code generation, where correctness is well-defined. Extending this training paradigm to more open-ended domains such as retrieval-augmented generation (RAG) [Zheng et al., 2025, Jin et al., 2025], creative writing, or agentic tool use [Feng et al., 2025] introduces additional challenges. In these settings, rule-based ORMs are insufficient due to the absence of clear gold standards or the non-uniqueness of valid outputs. To generalize this framework, it is critical to develop ORMs that better align with human preferences and exhibit stronger accuracy and generalization across diverse task formats [Liu et al., 2025].

How can self-evolution be applied to multimodal 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, Wu et al., 2025a]. To achieve this goal, several challenges require to be addressed. First, a comprehensive understanding of multimodal data is essential as the foundation for multimodal reasoning. Second, the format of CoT might be redefined, such as considering whether CoT should include tokens composed of multimodal data [Li et al., 2025a]. Third, leveraging the existing complex reasoning capabilities of LLMs for cross-modal data transfer is crucial. Additionally, many reasoning tasks in multimodal scenarios (e.g., embodied AI) face challenges such as high costs of environmental interaction and limited data resources during training [He et al., 2024a]. Overall, multimodal reasoning, especially embodied AI, grounded in multimodal scenarios is expected to become one of the primary pathways for AI to serve humans in the real world [Li et al., 2025b]. Reasoning serves as the foundational capability that enables embodied agents to tackle a wide range of practical tasks. To remain effective, such reasoning must continually evolve through ongoing interactions with complex and dynamic environments.