# *OpenR*: An Open Source Framework for Advanced Reasoning with Large Language Models

Jun Wang[1], Meng Fang[2], Ziyu Wan[3], Muning Wen[3], Jiachen Zhu[3], Anjie Liu[4], Ziqin Gong[4], Yan Song[1], Lei Chen[4], Lionel M. Ni[4], Linyi Yang[5], Ying Wen[3], Weinan Zhang[3]

[1] University College London [2] University of Liverpool [3] Shanghai Jiao Tong University
[4] Hong Kong University of Science and Technology (GZ) [5] Westlake University

## Abstract

In this technical report, we introduce *OpenR*, an open-source framework designed to integrate key components for enhancing the reasoning capabilities of large language models (LLMs). *OpenR* unifies data acquisition, reinforcement learning training (both online and offline), and non-autoregressive decoding into a cohesive software platform. Our goal is to establish an open-source platform and community to accelerate the development of LLM reasoning. Inspired by the success of OpenAI's o1 model, which demonstrated improved reasoning abilities through step-by-step reasoning and reinforcement learning, *OpenR* adopts a model-based approach, moving beyond traditional autoregressive methods. We demonstrate the efficacy of *OpenR* by evaluating it on the MATH dataset, utilizing publicly available data and search methodologies. Initial experiments confirm substantial gains (10-20% relative improvements), with test-time computing leading to improved reasoning and performance aided by process reward models. These models serve as both performance enhancers and guidance mechanisms during generation.

The *OpenR* framework, including code, models, and datasets, is accessible at `https://openreasoner.github.io`.

## 1 Introduction

OpenAI has recently unveiled o1 [OpanAI, 2014,0912], a groundbreaking large language model (LLM) that represents a giant leap forward in strong AI. The model is reported to be five times more proficient in math and coding compared to the previous GPT-4o, specifically displaying exceptional performance across various domains: it ranks in the 89th percentile for competitive programming, places among the top 500 students in a prestigious US math olympiad qualifier, and surpasses human PhD-level accuracy in physics, biology, and chemistry benchmarks. Trained using reinforcement learning techniques, o1 excels in complex reasoning tasks by explicitly embedding a *native* "Chain-of-Thought" (NCoT) process in LLMs, which allows it to "deep think" through step-by-step reasoning before generating responses. A key innovation of o1 is that it allows spending more time reasoning during the inference process, marking a shift from fast, direct responses to slow, deliberate, multi-step inference-time computation, as illustrated in Figure 1.

Interestingly, in human cognition, two correlated yet distinct modes of cognitive processing are presented to guide human decision-making and behaviours [Kahneman, 2011], each of which has the partial distinction between brain circuits and neural pathways. System 1 thinking is fast, automatic, and intuitive, operating effortlessly and often unconsciously. It relies on neural pathways that enable rapid processing, especially in situations needing quick reactions or when cognitive resources are

Correspondence to: Jun Wang and Meng Fang.

Technical Report.

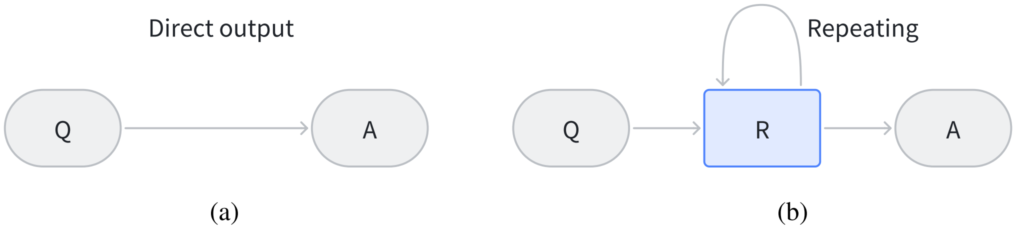


Figure 1: Inference-time computation. (a) An autoregressive LLM directly generates an answer (A) by conditioning the given question (Q). (b) The concept of chain of thought, or step-by-step thinking, involves incorporating intermediate reasoning steps (R) before arriving at the final answer (A). These repeated operations allow for 1) revisiting and revising prior outputs, 2) progressing to subsequent reasoning stages, and 3) exploring multiple reasoning paths or trajectories.

constrained. System 2 thinking is deliberate, effortful, and conscious, involving focused attention and analytical reasoning. It processes information more slowly and is used for complex problem-solving, logical reasoning, and decision-making tasks.

o1 model is an exciting development for AI, as LLMs can now not only generate rapid responses using learned patterns but, more significantly, simulate complex reasoning processes through mechanisms like chain-of-thought or other forms of search, similar to how humans engage in deeper, step-by-step thinking. o1's improved reasoning skills induce implications for multiple fields, including science, coding, and mathematics. In coding competitions, a specialised version of o1 achieved impressive results, scoring in the 49th percentile in the 2024 International Olympiad in Informatics and outperforming 93% of human competitors in simulated Codeforces contests. Beyond its technical capabilities, o1 also represents progress in AI safety and alignment. The model's chain of thought reasoning provides new opportunities for integrating human values and principles, resulting in improved performance on safety evaluations and jailbreak tests.

The idea of chain-of-thought reasoning [Wei et al., 2022] and step-by-step thinking in large language models (LLMs) is not new. Previous research has shown that simply adding instructions like "describe your reasoning in steps" or "explain your answer step by step" to the input questions or providing a few shot examples can trigger LLMs to generate intermediate reasoning steps (as illustrated in Figure 1) and subsequently improve problem-solving, especially in tasks like math and coding [Wei et al., 2022, Nye et al., 2021]. However, these approaches build on existing LLMs without truly embedding the chain of thought ability within the models themselves. As a result, LLMs cannot inherently learn this reasoning capability, leading to active research on how to integrate it directly into model training. Proposed methods range from collecting specialised training data to building reward models [Ouyang et al., 2022, Li et al., 2022, Luo et al., 2024] and increasing the computational complexity of decoding [Snell et al., 2024, Wu et al., 2024], but none have yet achieved significant performance breakthroughs at scale.

It remains unclear whether o1's innovation is rooted in the model itself, rather than relying on external prompting systems. If it indeed involves explicitly embedding step-by-step reasoning natively within the architecture, this would represent a significant breakthrough. Building on substantial performance gains, o1 has shown that the scaling principles traditionally applied during training [Kaplan et al., 2020, Snell et al., 2024] are now relevant to the inference phase. We should reallocate our computational focus, balancing pre-training efforts with efficient use of inference-time computation. Allowing LLMs to enhance their outputs with increased test-time computing is an essential step towards creating generally self-improving agents capable of managing open-ended strong reasoning and decision-making tasks. This direction, which we refer to as LLM-Native Chain-of-Thought (NativeCoT), should be able to inherently mirror the deliberate, analytical process possessed by human's System 2 thinking [Kahneman, 2011].

In this report, we present *OpenR*, an open-source framework built on the principles behind OpenAI's o1 model, designed to replicate and extend its reasoning capabilities. Our approach focuses on improving LLM reasoning by integrating process supervision, reinforcement learning (RL), and inference-time computation strategies such as guided search. *OpenR* implements key components such as data augmentation for process supervision, policy learning via RL, and efficient decoding algorithms. By doing so, it shifts the focus from merely scaling model parameters during pre-training to leveraging smarter inference strategies at test time. These techniques help the model refine its reasoning step by step, allowing it to pause, evaluate intermediate reasoning, and select better solution pathways during test-time computation. Through experiments on publicly available benchmarks like the MATH dataset, we demonstrate how the combination of process reward models and guided search improves test-time reasoning performance by 10-20%.

In summary, we introduce *OpenR*, an open-source framework that integrates test-time computation and process supervision to enhance reasoning in LLMs, providing an open platform with models, data, and code to foster collaboration and accelerate research in LLM reasoning. The framework includes reinforcement learning algorithms designed to optimize decision-making during training, enabling more accurate and deliberate step-by-step reasoning. Additionally, *OpenR* provides tools for generating synthetic process reward data, reducing dependence on costly human annotations and supporting scalable process supervision. Through experiments, we demonstrate the effectiveness of process reward models and test-time guided search.

## 2 Related Work

Key references in the field of improving reasoning capabilities in large language models (LLMs) highlight several innovative approaches, including inference-time computing, process reward models, and data acquisition methods.

**Inference-time Computing.** To discuss the role of inference-time computation in large language models (LLMs), recent studies have focused on optimizing the efficiency and effectiveness of reasoning during the inference process rather than merely relying on the scaling law of training-time computing. A pivotal study, Feng et al. [2024] demonstrates the benefits of using MCTS as a decoding mechanism, which enhances inference computation by actively planning and selecting higher-quality responses. This approach aligns with the reasoning-as-planning approach proposed in Hao et al. [2023], where reasoning is viewed as a process similar to planning in decision-making processes, further underscoring the centrality of step-wise reasoning at inference time. In recent, the work [Snell et al., 2024] reinforces that optimizing inference strategies can yield superior performance gains compared to simply increasing model size, underscoring the critical role of test-time computation. Finally, this is complemented by the findings of work [Goyal et al., 2023], which introduces an implicit reasoning model by incorporating pause tokens to encourage deliberate reasoning during generation. Collectively, these recent advances suggest the growing recognition of inference-time optimization – whether through planning-based reasoning models or computational optimization – as a critical factor in improving LLM capabilities, advocating for strategies that enhance reasoning, planning, and compute efficiency beyond mere training-time scaling.

**From Outcome Supervision to Process Supervision.** The shift from Outcome Supervision to Process Supervision in language model training has gained prominence in recent research, particularly with respect to enhancing reasoning capabilities. The foundational work by Cobbe et al. [2021a] introduces Outcome-supervised Reward Models (ORM) and the widely used math reasoning dataset, GSM8K, where verifiers are trained to assess the final correctness of generated solutions. While ORM plays a crucial role in the early stage, it primarily focuses on evaluating the end result rather than the reasoning steps leading to the final output.

Building on this, the concept of process reward models (PRM) is introduced as a more granular and transparent approach. With both ORM and PRM, DeepMind proposes the idea of supervising intermediate reasoning steps alongside the final outcome, allowing for more detailed feedback during the reasoning process [Uesato et al., 2022]. This research laid the groundwork for subsequent developments in process-based verification. On the other hand, OpenAI's work [Lightman et al., 2023] continues this trend by refining PRM through a follow-up study that emphasizes verifying each intermediate step in reasoning tasks by providing a high-quality human-labeled process-supervision dataset, namely PRM800K, which has been enriched in our work.

Similarly, the integration of verifier models with majority voting schemes, as highlighted in Li et al. [2022], showcases the practical application of PRM. This method uses a verifier to scrutinize each reasoning step while incorporating majority voting to increase the reliability of the final result. Furthermore, Yu et al. [2024] introduce another approach using reinforcement learning to enhance the planning and reasoning process in LLMs, providing a hybrid of both outcome- and process-

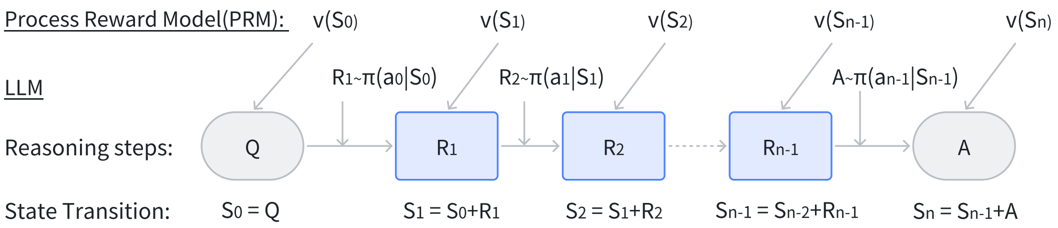


Figure 2: In this MDP formulation, the LLM is tasked with generating reasoning steps and the final answer to a question in a step-by-step manner. The LLM policy operates by generating tokens, which form higher-level reasoning constructs. The states represent the sequence of reasoning steps so far, and actions correspond to the selection of new reasoning steps or the final answer. The LLM policy governs the choice of actions, and the process reward model (PRM) provides feedback on the quality of reasoning steps and the final answer. By optimising the policy to maximise the reward, the LLM can be guided by PRM to generate accurate and meaningful reasoning processes.

supervised techniques. Besides the traditional scalar-based reward models, the recently proposed generative reward model (GenRM) [Zhang et al., 2024] attracts significant attention since the verifier and generator can interact with each other in a more information-dense text-based manner. When the reward model monitors the policy, it not only outputs the score of each answer but also details the reason for the error. This trend in recent research reflects a broader move toward more sophisticated process-supervision methods, which has been fully covered in this project.

**Data Acquisition.** The problem of Data Acquisition for PRM has evolved significantly, focusing on automating the extraction of step-by-step reasoning data, which is crucial for training models capable of complex reasoning tasks. The STaR technique [Zelikman et al., 2022] presents a novel self-taught reasoning approach where models generate and bootstrap their own reasoning processes for further training, thus improving reasoning capabilities without extensive labeled datasets. Building upon the foundation laid by STaR, Zelikman et al. [2024] demonstrate how these techniques could be generalized beyond specific domains like mathematical problem-solving. By extending the reasoning process to arbitrary tasks and incorporating the methodology into pre-training, Quiet-STaR highlights the versatility of automated process supervision across various tasks, marking a significant step in scaling data acquisition for reasoning tasks. In addition, Luo et al. [2024] represent the latest advancement in the field, specifically focusing on mathematical reasoning. This work refines the methods for automated data acquisition, making the process more robust and applicable to increasingly complex problem-solving scenarios. Moreover, Wang et al. [2024a] take the concept of automatic process supervision a step further by proposing a practical solution for training models without relying on human-labeled data. Finally, the empirical results in Wang et al. [2024b] extend these approaches by testing their applicability on coding tasks, demonstrating that **process supervision** can be effectively induced by the model itself. These works underscore the increasing reliance on automated data acquisition methods, where models are equipped to extract and verify their self-reasoning processes. To facilitate the research in this direction, we make the generated dataset and code publicly available.

In summary, advanced reasoning in models such as OpenAI's o1 relies heavily on careful data selection, sophisticated PRM training, and enhanced decoding methods. Approaches such as tree-based search, reinforcement learning, and step-aware verifiers enable these models to tackle more complex tasks. As research progresses, LLMs are expected to further enhance their autonomous reasoning, planning, and problem-solving capabilities. Our project aims to serve as a starting point for transparently investigating and evaluating the potential of inference-time computation.

## 3 The *OpenR* LLM Reasoning Framework

To model the process of reasoning in tasks such as question-answering or problem-solving, we structure the reasoning task using the $Q \rightarrow \{R\} \rightarrow A$ sequence, where:

- $Q$ represents the question or prompt that initiates the reasoning process;

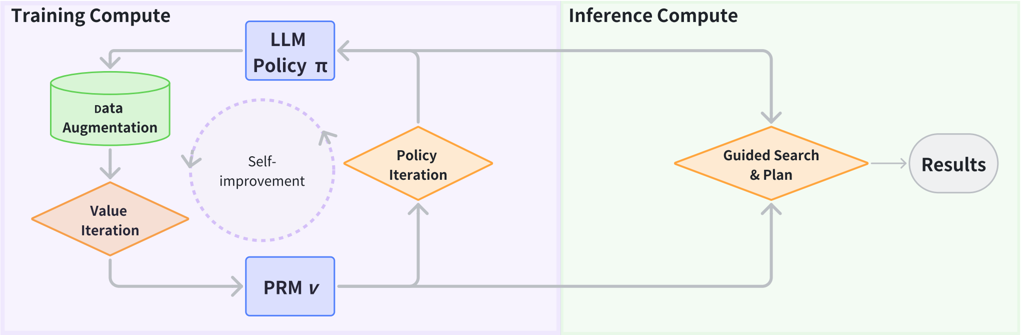


Figure 3: The *OpenR* framework for LLM reasoning. Combining the value function from the PRM with the LLM's policy generation ensures guided and controlled results. During training, the generation produced by the LLM's policy and the evaluation provided by the PRM reinforce each other, leading to continuous self-improvement and refinement of both components.

- $R$ represents the sequence of intermediate reasoning steps the model generates to build toward the solution;
- $A$ represents the final answer or solution produced after the reasoning steps.

This structure allows the LLM to generate a sequence of reasoning steps that logically connect the question $Q$ to the final answer $A$. We can define the reasoning process as a Markov Decision Process (MDP) [Bellman, 1958]. An MDP representation offers a flexible framework for modelling reasoning. It allows the model to generate sequential reasoning steps toward the final answer step by step while also enabling a tree structure by sampling multiple paths at each step for alternative reasoning trajectories. By combining both approaches - sequential and branching reasoning - the model can explore diverse solutions, creating a versatile and comprehensive reasoning process.

In an MDP (as illustrated in Figure 2), the LLM policy functions by generating tokens that combine to form higher-level reasoning constructs. States represent the sequence of reasoning steps taken up to the current point, while actions involve selecting the next reasoning step or the final answer. The LLM policy generates these action choices, and the process reward model (PRM) [Lightman et al., 2023, Uesato et al., 2022] offers feedback on the quality of both the reasoning steps and the final answer. The PRM guides the LLM toward producing accurate and meaningful reasoning processes by optimising the policy to maximise the reward.

### 3.1 System Design

The process reward model (PRM) plays a crucial role in enhancing the LLM's policy in two key ways. First, during training, the PRM improves the LLM policy through policy optimisation techniques (Policy Iteration as shown in Figure 3). Second, during the decoding phase, the PRM guides the LLM's search process, steering the reasoning toward more effective outcomes (as shown in Figure 3). As we will show next, the LLM policy also helps identify missing intermediate reasoning steps, which in return enables further training and refinement of the PRM. As shown in Figure 3, this iterative interaction allows the LLM and PRM to unlock each other's potential for improved reasoning continuously.

### 3.2 Data Augmentation

For a solution or chain-of-thought provided by large language models (LLMs), we use more precise and fine-grained feedback instead of relying solely on the final answers. We collect data for process supervision, which provides step-wise feedback for a given solution. Formally, a PRM computes $p_t = \text{PRM}([q, x_{1:t-1}], x_t)$, where $x_{1:t} = [x_1, \cdots, x_t]$ represents the first $t$ steps of the solution. This method provides more precise and fine-grained feedback compared to outcome reward models (ORMs), as it identifies the exact location of errors within the problem-solving process [Lightman et al., 2023].

**MATH-APS.** We augment the data by automatically generating synthetic samples. In addition to the PRM800k dataset [Lightman et al., 2023], which relies on costly human annotation and is difficult to scale, we introduce a new dataset called MATH-APS, based on MATH [Hendrycks et al., 2021], using automated methods such as OmegaPRM [Luo et al., 2024]. This approach reduces the reliance on expensive human annotations, enabling more scalable data collection. Automatic methods such as OmegaPRM, Math-Shepherd [Wang et al., 2024a] and MiPS [Wang et al., 2024b] efficiently collect high-quality process supervision data. While Math-Shepherd and MiPS provide automatic annotation for process supervision, they require lots of policy calls, making them computationally expensive. OmegaPRM improves this process by iteratively dividing the solution, performing rollouts, and identifying the first incorrect step in a model's solution.

We follow OmegaPRM [Luo et al., 2024] and collect PRM training examples by constructing a state-action tree using LLMs. For each question, a tree is built where each node contains the question $q$, the solution prefix $s$, and all previous rollouts $\{(s, r_i)\}_{i=1}^{k}$ (with $r_i$ indicating the $i$-th rollout). Each edge represents a single step or a sequence of steps from the node. For each node, we calculate the Monte Carlo estimation $MC(s)$ and the value function $Q(s, r)$ to guide the selection of rollouts during tree traversal. The value function is defined as: $Q(s, r) = \alpha \cdot \frac{1}{1-MC(s)} \cdot \beta \cdot \frac{\text{len}(r)}{L}$, where $\alpha$, $\beta$, and $L$ are constants, and len$(r)$ is the length of the rollout. We also compute the exploration term: $U(s) = c_{\text{puct}} \cdot \frac{\sqrt{\sum_i N(s_i)}}{1+N(s)}$, where $N(s)$ is the visit count and $c_{\text{puct}}$ is a constant encouraging exploration. During the selection phase, a rollout is chosen using a variant of the PUCT algorithm: $(s, r) = \arg\max_{(s,r)}[Q(s, r) + U(s)]$. This heuristic selects the most valuable rollouts. A binary search is then used to identify the first error in the selected rollouts, and rollouts with $0 < MC(s)$ are added to the candidate pool. All positions before the first error become new states for further exploration.

### 3.3 Supervised Training for PRMs

In PRMs, the goal is to determine whether the sequence of the solution process is currently on the right track, so it should output a binary indicator of correctness. Specifically, we assign a score $y_t$ between 0 and 1 given a problem $q$ and a sequence of solution steps $x_1 \rightarrow x_t$. This score represents the correctness of the current problem-solving process. As a result, the problem is reframed as $y_t = \text{PRM}(q, x_1, x_2, \cdots, x_t)$, which can be treated as a binary classification task. The PRM is trained through supervised fine-tuning on a LLM, with the correct/incorrect distinction serving as the classification label. We then use the LLM to predict the next token of the step token.

**Math-psa.** The PRM is trained through supervised fine-tuning on an LLM, with the correct/incorrect distinction serving as the classification label. We train a PRM named Math-psa using datasets such as PRM800K [Lightman et al., 2023], Math-Shepherd [Wang et al., 2024a], and our MATH-APS dataset (see Section 3.2). These datasets are structured into three components: *question*, *process*, and *label*. The input consists of a concatenation of the *question* and the *process*. In the *process*, the solution is divided into multiple steps, each separated by a special step token ("\n\n\n\n\n"), marking the end of each step where the PRM can make predictions. The *label* is a classification of the entire process, with each step labeled as either + 'or -' based on the correctness of the solution.

During training, the data is fed to the LLM as a next-token prediction task. The model is trained to predict a positive or negative token immediately following each step token. As described in the data section, the input consists of the *question* concatenated with the *process*, and the step tokens separate the steps in the process. The labels are assigned such that at the positions of the step token, the label is either a positive or negative token, while all other positions are ignored during the loss computation. The attention mask is set to 1 for all tokens except the step token positions, ensuring that during training, the LLM focuses only on the input sequence $(q, x_1 \rightarrow x_t)$ and does not attend to the step tokens themselves.

### 3.4 Policy Learning for LLMs

We transform math problems into a language-augmented Markov Decision Process (MDP) $\mathcal{M} = (\mathcal{V}, \mathcal{S}, \mathcal{A}, \mathcal{T}, R, \gamma)$ [Van Otterlo and Wiering, 2012, Carta et al., 2023]. Given $\mathcal{V}$ the vocabulary and $w \in \mathcal{V}$ the tokens, $\mathcal{A} \subset \mathcal{V}^N$, $\mathcal{S} \subset \mathcal{V}^N$ are action and state space, respectively, i.e., actions and states are sequences of tokens. $\mathcal{T} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$ is the state transition function. $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ is the reward function that responds to each action, and $\gamma$ is the discounted factor that typically less than 1. An initial state $s_0$ represents a given problem in mathematical problem-solving scenarios. A language model receives this input and generates an intermediate reasoning step, denoted as an action $a_0$. This action $a_0$ is then concatenated with the initial problem $s_0$ to form the subsequent state $s_1$, which is used to infer the next action $a_1$, This iterative process continues, with each state-action pair successively informing the following state, i.e., $\mathcal{T}: s_{t+1} = \{s_t, a_t\}$ at time step $t$, until the model arrives at the final answer. After inferring each action $a_t$, the model receives a reward signal $r_t^{PRM} = R(s_t, a_t)$ from a well-trained PRM. Following this process with trajectories of a maximum timestep $T$, the agents earn a discounted cumulative return of $R^\gamma = \sum_{t=0}^{T} \gamma^t r_t^{PRM}$, which is aimed to be maximised by RL algorithms. We correspondingly implement this MDP as a reinforcement learning environment like OpenAI's Gym. In these environments, math problems are presented as tasks where the model takes sequential actions to solve the problem, receiving rewards for correct actions and penalties for incorrect ones, which enables the model to iteratively learn and refine its problem-solving strategies through trial and error, ultimately enhancing its mathematical reasoning skills.

**RL Training.** Training LLMs with reinforcement learning often involves Proximal Policy Optimization (PPO) [Schulman et al., 2017] to align generated language outputs with desired actions. This approach bridges the gap between language understanding and actionable outputs by reinforcing the generation of responses that are both contextually accurate and aligned with predefined goals, effectively linking language comprehension with strategic planning. We provide both traditional PPO and an efficient variant of PPO, i.e., Group Relative Policy Optimization (GRPO) [Shao et al., 2024]. The primary distinction between these two lies in their approaches to advantage value estimation. Specifically, PPO utilizes a network to approximate the state value function, leveraging the Generalized Advantage Estimation (GAE) technique [Schulman et al., 2015] to derive the advantage. In contrast, GRPO simplifies this process by directly employing a normalized reward signal to estimate an action's advantage, i.e., $A(s_t, a_t) = \frac{r_t^{PRM} - \text{mean}(r^{PRM})}{\text{std}(r^{PRM})}$. Compared with PPO, GRPO bypasses the need for an extra critic network and reduces the resources consumed during training, however, it emphasizes the stability of PRMs more.

### 3.5 Decoding: Inference-Time Guided Search and Planning

Following Snell et al. [2024], we use PRMs to assess the accuracy of each solution step. Once a high-quality process reward model is trained, we integrate it into the decoding process alongside the language model, enabling guided search and scoring or voting across multiple generations.

To use PRMs as verifiers, we define a method for evaluating the correctness of LLM-generated solutions. Specifically, we map the scores of individual steps $\{r_t^{PRM}\}_{t=0}^{T}$ to a final score. Following the strategies outlined by Lightman et al. [2023] and Snell et al. [2024], we employ two approaches:

- *PRM-Min*: choose the minimum value among all scores, i.e., $v = \min\{r_t^{PRM}\}_{t=0}^{T}$.
- *PRM-Last*: choose the last step's score as the final score, i.e., $v = r_T^{PRM}$. This strategy has been shown to be as good as *PRM-Min* in Snell et al. [2024].

Once multiple answers are generated by scaling test-time computations, we need strategies to select the best answer based on their scores. We adopt three strategies from Feng et al. [2024]:

- *Majority-Vote*: Aggregate answers using majority vote: $f^* = \arg\max_f \sum_{\mathbf{y}^j} \mathbf{1}_{\text{final\_ans}(\mathbf{y}^j)=f}$, where $\mathbf{1}$ is the indicator function.
- *RM-Max*: Given an outcome reward model, the aggregation can choose the answer $f$ with maximum final reward, $f^* = \text{final\_ans}(\arg\max_{\mathbf{y}^j} v(\mathbf{y}^j|\mathbf{x}))$.
- *RM-Vote*: Given an outcome reward model, the aggregation can choose the answer $f$ with the sum of rewards, namely $f^* = \arg\max_f \sum_{\mathbf{y}^j;\text{final\_ans}(\mathbf{y}^j)=f} v(\mathbf{y}^j|\mathbf{x})$.

Combining these strategies, we can define multi-answer weighting methods, such as *PRM-Last-Max*, which refers to using *PRM-Last* with *RM-Max*.

Our framework allows us to select among various search algorithms — such as beam search, best-of-N selection, and others — each with unique advantages depending on the quality of PRMs. Complex search algorithms may yield better performance on more difficult tasks, while simpler methods, such as best-of-N, often perform adequately for less challenging cases [Snell et al., 2024].

We mainly employ two strategies:

- *Best-of-N*: Given a base model, the best-of-N sampling approach generates $N$ outputs in parallel and selects the answer with the highest score according to a learned process using PRMs. This method is similar to previous work that leverages verifiers or reward models [Cobbe et al., 2021b, Lightman et al., 2023]. While simple, it is an effective baseline that leverages test-time computation to improve the performance of LLMs. PRMs can act as dense verifiers [Lightman et al., 2023, Wang et al., 2024a], and it is intuitive that providing a strong signal can lead to improved outcomes. Since dense rewards can be obtained for a base model's solution, we need to consider how best to use this feedback to optimise test-time computation.
- *Beam Search*: The LLM generates $N$ different outputs for the first step, which are then scored using PRMs. These $N$ outputs are scored using PRMs, and the top $N/m$ ($\frac{N}{m} \in \mathbb{Z}$) highest-scoring outputs are retained. We then keep only these $N/m$ outputs for the current step. For each of these outputs, we sample $M$ potential next steps via the base model, returning to $N$ total outputs. The process is repeated: new candidates are scored, filtered, and sampled for subsequent steps. The scores from the PRMs are central to guiding this search. As with the best-of-N approach, we use both last vote and majority vote strategies to aggregate the scores, with the latter relying on the sum of scores across the PRMs as in [Wang et al., 2022].

We are going to continuously work on developing more complicated inference-time guided search decoding methods such as Monte Carlo Tree Search (MCTS), which has been already covered in the codebase of OpenR and other methods like sequential revision [Snell et al., 2024].

# 4 Experiments

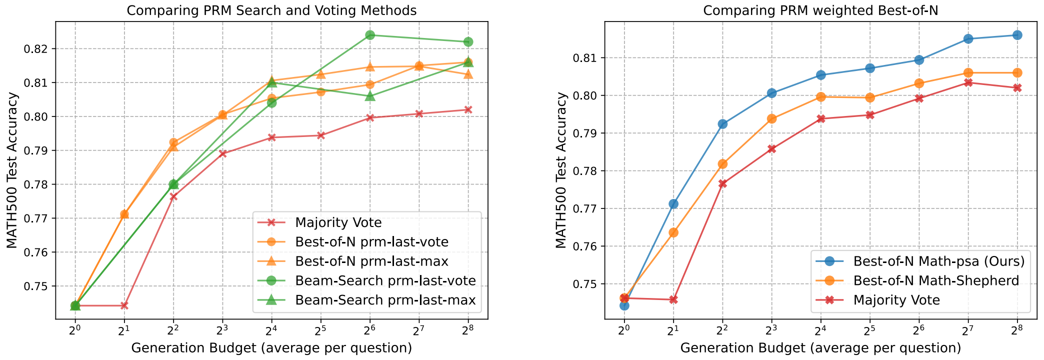
To demonstrate the capabilities of our OpenR framework, we present quantitative results on large language model inference and training. We evaluate our open framework using the MATH dataset [Hendrycks et al., 2021], which includes a wide range of high-school competition-level math problems. This makes it an ideal benchmark for testing reasoning skills. To ensure fair comparisons with previous work and reduce overfitting, we follow Lightman et al. [2023] and use a subset of 500 problems for evaluation, known as MATH500, in which the problems are sampled randomly.

## 4.1 Scaling LLM Test-Time Compute

**Setting.** Our PRM model, Math-psa, is fine-tuned from the Qwen2.5-Math-7B-Instruct [Yang et al., 2024] model using multiple datasets, including PRM500K [Lightman et al., 2023], Math-Shepherd [Wang et al., 2024a], and our MATH-APS dataset (500k state-value pairs). In the meantime, we also experiment with Math-Shepherd PRM for comparison purposes. Following Snell et al. [2024], we employ best-of-N and beam search algorithms for test-time computation. We compare multiple test-time computation schemes across pre-defined budgets of token generation. Among different aggregation strategies, we select PRM-Last as a representative. The LLM inference server is implemented using FastChat [Zheng et al., 2023].

**Results.** Figure 4a compares the performance of these search and voting methods during inference. The y-axis represents testing accuracy on the MATH500 dataset, while the x-axis shows the generation budget (average tokens per question), reflecting the computational effort or token usage per question. The figure indicates that both Best-of-N and Beam-Search methods significantly outperform Majority Vote, especially as the generation budget increases, showing a similar pattern with previous findings Snell et al. [2024]. Under low test-time computation budgets ($< 2^4$), Best-of-N methods demonstrate better performance compared to Beam Search, whereas Beam Search can reach matching performance given higher budgets, or even surpass Best-of-N with $PRM - Last$ strategy used at budgets larger than $2^5$.

On the other hand, Figure 4b investigates how different PRMs affect test-time computation. We compare the performance of best-of-N methods with different PRM guidance. The figure shows that our PRM (Math-aps) can achieve the highest testing accuracy across all tested computation budgets. This indeed has verified that our PRM training pipeline can provide effective learning of process supervision.



(a) Comparision of different PRM guided search and voting methods.

(b) Comparison of different reward models in best-of-N test time computation.

Figure 4: Test-time computation experiments on search methods (a) and PRMs (b).

## 4.2 Online Policy Learning for LLM

**Setting.** In the policy learning experiment, we use the Qwen2.5-1.5B-Math-Instruct model as the policy model for training, with the Math-Shepherd model [Wang et al., 2024a] serving as the PRM to provide feedback during RL. In addition to the MATH500 dataset, we test the performance of the model on a specific math problem: *"How many positive whole-number divisors does 196 have?" with the final answer being "9."*

**Results.** Figure 5 illustrates the reward obtained by a reinforcement learning (RL) algorithm with process reward models (PRM) on a single math problem. The rewards steadily increase over time, showing consistent improvement, with performance stabilizing after around 6 hours of training. This indicates that the model becomes more accurate in solving the specific problem as training progresses.

On the MATH500 dataset, results exhibit more fluctuations in rewards. This suggests that the PPO algorithm with PRM faces a more complex challenge due to the diversity of problems in the dataset. Although the rewards increase over time, their variability highlights the difficulty of generalizing across a broader set of problems. This indicates the need for further improvement of the algorithm to enhance its adaptability for diverse problem sets in future work.

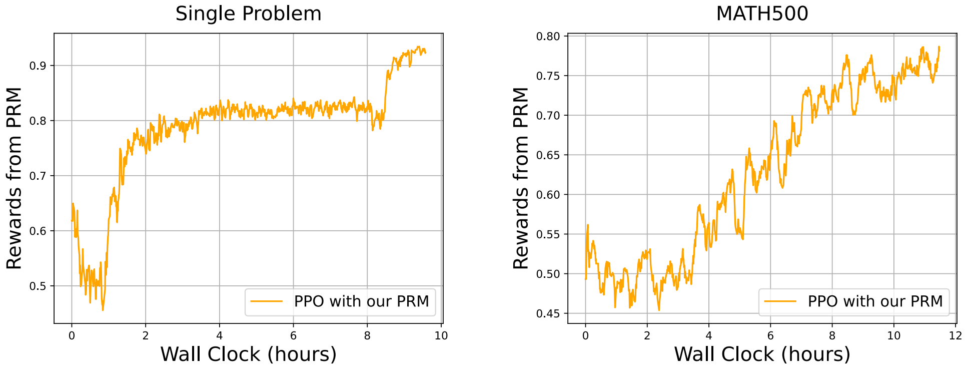


Figure 5: Performance on a single math problem and the MATH500 dataset over training time.

### 4.3 Generated Examples

**Comparison of the Math-psa PRM and the Math-Shepherd PRM.** In Figure 6 and Figure 7, we compare the responses of different PRMs to a given reasoning step. The PRM scores for each step are represented as [Math-psa (ours) Score, Math-Shepherd PRM Score].

**Q&A Session 1**

Question:

What is the remainder of 2004 divided by 12?

Reason:

**Step 1**: To find the remainder of 2004 divided by 12, we can use the division algorithm, which states that for any integers $a$ and $b$ (with $b > 0$), there exist unique integers $q$ (the quotient) and $r$ (the remainder) such that $a = bq+r$ and $0 \leq r < b$. [0.958, 0.988]

**Step 2**: Here, $a = 2004$ and $b = 12$. We need to find $q$ and $r$ such that:

$$2004 = 12q + r$$

where $0 \leq r < 12$. [0.924, 0.987]

**Step 3**: First, we perform the division of 2004 by 12:

$$2004 \div 12 = 167$$

This means the quotient $q$ is 167. Now, we can find the remainder $r$ by multiplying the quotient by the divisor and subtracting the result from the original number:

$$2004 - 12 \times 167 = 2004 - 2004 = 0$$

[0.777, 0.719]

**Step 4**: Thus, the remainder $r$ is:

$$r = 0$$

[0.777, 0.378]

**Step 5**: Therefore, the remainder when 2004 is divided by 12 is: $\boxed{0}$. [0.622, 0.349]

Answer:

An integer is divisible by 12 if and only if it is divisible by both 3 and 4.
Because $2 + 0 + 0 + 4 = 6$ is divisible by 3, 2004 is divisible by 3.
Also, the last two digits of 2004 form a multiple of 4, so 2004 is divisible by 4 as well.
Therefore, 2004 is divisible by 12 and hence leaves a remainder of $\boxed{0}$ when divided by 12.

Figure 6: Math-psa (ours) is able to assign more reasonable scores to reasoning steps. Particularly at **Step 4** and **Step 5**, Math-psa shows higher confidence in the output whereas the Math-Shepherd PRM appears to be less certain about the given correct response.

**Q&A Session 2**

Question:

Let $f$ be the function defined by $f(x) = x^3 - 49x^2 + 623x - 2015$, and let $g(x) = f(x+5)$. Compute the sum of the roots of $g$.

Reason:

**Step 1**: To find the sum of the roots of the function $g(x) = f(x+5)$, we first need to understand the relationship between the roots of $g(x)$ and the roots of $f(x)$. $[0.905, 0.810]$

**Step 2**: Given the function $f(x) = x^3 - 49x^2 + 623x - 2015$, we know that the sum of the roots of $f(x)$ can be found using Vieta's formulas. For a cubic polynomial $ax^3 + bx^2 + cx + d$, the sum of the roots (denoted as $\alpha + \beta + \gamma$) is given by:

$$\alpha + \beta + \gamma = -\frac{b}{a}$$

In our case, $a = 1$ and $b = -49$. Therefore, the sum of the roots of $f(x)$ is:

$$\alpha + \beta + \gamma = -\frac{-49}{1} = 49$$

$[0.706, 0.715]$
**Step 3**: Next, we consider the function $g(x) = f(x+5)$. This represents a horizontal shift of the function $f(x)$ by 5 units to the left horizontal shifts do not change the sum of the roots of the polynomial; they only shift the positions of the roots along the x-axis. $[0.593, 0.788]$

**Step 4**: Thus, the sum of the roots of $g(x)$ is the same as the sum of the roots of $f(x)$. Therefore, the sum of the roots of $g(x)$ is:

$$\boxed{49}$$

$[0.182, 0.665]$

Answer:

Let $a$, $b$, $c$ be the roots of $x^3 - 49x^2 + 623x - 2015$. Then by Vieta's formulas, $a + b + c = 49$.
The roots of $g(x) = f(x+5)$ are $a-5$, $b-5$, and $c-5$, and their sum is $a+b+c-15 = 49 - 15 = \boxed{34}$.

Figure 7: The given reasoning process leads to an incorrect answer. It shows that Math-psa PRM assigns a lower score to **Step 3** since horizontal shifts do affect the sum of the roots of a polynomial. Math-psa PRM further discourages **Step 4** whereas Math-Shepherd PRM still holds high confidence.

**Comparison between LLMs before and after Policy Learning.** We show the examples from models before policy learning in Figure 8 and after policy learning with with PRMs in Figure 9 the policy learning process with Math-psa PRM.

**Q&A Session 3**

Question:

A regular hexagon can be divided into six equilateral triangles. If the perimeter of one of the triangles is 21 inches, what is the perimeter, in inches, of the regular hexagon?

Reason:

**Step 1**: A regular hexagon can be divided into six equilateral triangles.
**Step 2**: The perimeter of one of the triangles is 21 inches.
**Step 3**: Since the hexagon is regular, all sides are equal.
**Step 4**: The perimeter of the hexagon is the sum of all six sides.
**Step 5**: The perimeter of the hexagon is 21 * 6 = 126 inches. [Wrong Step]

Answer:

The answer is 126.

Figure 8: This example is collected from the vanilla policy of Qwen2.5-Math-1.5B-Instruct without a policy learning process. We observe that it generates a wrong step where the perimeter of a triangle is mistakenly used as the side length, and thus leads to a wrong answer.

**Q&A Session 4**

Question:

A regular hexagon can be divided into six equilateral triangles. If the perimeter of one of the triangles is 21 inches, what is the perimeter, in inches, of the regular hexagon?

Reason:

**Step 1**: A regular hexagon can be divided into six equilateral triangles.
**Step 2**: The perimeter of one of the triangles is 21 inches.
**Step 3**: Since the hexagon is regular, all sides of the hexagon are equal.
**Step 4**: The perimeter of the hexagon is the sum of the lengths of all its sides.
**Step 5**: Since the hexagon is divided into six equilateral triangles, each side of the hexagon is equal to the side of one of the triangles.
**Step 6**: The perimeter of the hexagon is 6 times the length of one side of the triangle.
**Step 7**: The length of one side of the triangle is 21 / 3 = 7 inches.
**Step 8**: The perimeter of the hexagon is 6 * 7 = 42 inches.

Answer:

The answer is 42.

Figure 9: On the same examples. Models after policy learning with Math-psa PRM can address the issue, indicating that the RL process with proper PRM can potentially help reasoning.

**Comparison of Different Searching Strategies.** In Figure 10, 11 and 12, we compare how different test-time compute methods affect reasoning outputs.

**Q&A Session 5**

Question:

Kathy needs money for her trip to Europe. If she has 300 US dollars in the bank but wants to withdraw half of it in British pounds and half of it in euros, how many more euros than pounds will she have? Assume 1 pound = 1.64 USD and 1 euro = 1.32 USD, and round to the nearest whole number.

Reason:

**CoT Reasoning**

**Step 1**: First, calculate the amount of money Cathy will withdraw in pounds and euros:

$$\text{Amount withdrawn in pounds} = \frac{300}{2} = 150 \text{ USD}$$

$$\text{Amount withdrawn in euros} = \frac{300}{2} = 150 \text{ USD}$$

**Step 2**: Next, convert the amounts from pounds and euros back to US dollars using the given exchange rates:

$$\text{Value of 150 pounds in USD} = 150 \times 1.64 = 246 \text{ USD}$$

$$\text{Value of 150 euros in USD} = 150 \times 1.32 = 198 \text{ USD}$$

**Step 3**: Now, find the difference between the amounts in euros and pounds:

$$\text{Difference} = 198 - 246 = -48 \text{ USD}$$

So the answer is:

$$\boxed{48}$$

Answer:

After Kathy has exchanged half of her money ($300 USD$\div 2 = 150$ USD) into pounds, she will have:

$$150 \text{ USD} \times \frac{1 \text{ pound}}{1.64 \text{ USD}} \approx 91.46 \text{ pounds}$$

After exchanging the other half of her money into euros, she will have:

$$150 \text{ USD} \times \frac{1 \text{ euro}}{1.32 \text{ USD}} \approx 113.64 \text{ euros}$$

Subtracting these two values, we have:

$$113.64 - 91.46 = 22.18$$

Since the question calls for the nearest whole number, we round 22.18 to the final answer of:

$$\boxed{22}$$

Figure 10: Example of CoT reasoning. The answer is wrong.

**Q&A Session 6**

Question:

Kathy needs money for her trip to Europe. If she has 300 US dollars in the bank but wants to withdraw half of it in British pounds and half of it in euros, how many more euros than pounds will she have? Assume 1 pound = 1.64 USD and 1 euro = 1.32 USD, and round to the nearest whole number.

Reason:

**Best-of-N Reasoning**
**Step 1**: Kathy has 300 US dollars in the bank. She wants to withdraw half of it in British pounds and the other half in euros. First, we calculate the amount she will withdraw in each currency:

$$\text{Amount withdrawn in pounds} = \frac{300}{2} = 150 \text{ USD}$$

$$\text{Amount withdrawn in euros} = \frac{300}{2} = 150 \text{ USD}$$

**Step 2**: Next, we convert these amounts into pounds and euros using the given exchange rates. The exchange rate for pounds is 1 pound = 1.64 USD, so the amount in pounds is:

$$\text{Amount in pounds} = \frac{150}{1.64} \approx 91.50 \text{ pounds}$$

**Step 3**: The exchange rate for euros is 1 euro = 1.32 USD, so the amount in euros is:

$$\text{Amount in euros} = \frac{150}{1.32} \approx 113.64 \text{ euros}$$

**Step 4**: Now, we find the difference between the amounts in euros and pounds:

$$\text{Difference} = 113.64 - 91.50 = 22.14$$

**Step 5**: Rounding 22.14 to the nearest whole number, we get:

$$\boxed{22}$$

Answer:

After Kathy has exchanged half of her money ($300 USD$\div$2 $=$ 150 USD) into pounds, she will have:

$$150 \text{ USD} \times \frac{1 \text{ pound}}{1.64 \text{ USD}} \approx 91.46 \text{ pounds}$$

After exchanging the other half of her money into euros, she will have:

$$150 \text{ USD} \times \frac{1 \text{ euro}}{1.32 \text{ USD}} \approx 113.64 \text{ euros}$$

Subtracting these two values, we have:

$$113.64 - 91.46 = 22.18$$

Since the question calls for the nearest whole number, we round 22.18 to the final answer of:

$$\boxed{22}$$

Figure 11: Example of Best-of-N reasoning. The answer is correct.

**Q&A Session 7**

Question:

Kathy needs money for her trip to Europe. If she has 300 US dollars in the bank but wants to withdraw half of it in British pounds and half of it in euros, how many more euros than pounds will she have? Assume 1 pound = 1.64 USD and 1 euro = 1.32 USD, and round to the nearest whole number.

Reason:

**Beam Search Reasoning**
**Step 1**: Kathy has 300 US dollars (USD) in the bank. She wants to withdraw half of it in British pounds (GBP) and half of it in euros (EUR). Let's calculate the amount of money she will withdraw in each currency. First, we calculate the amount to be withdrawn in pounds:

$$\text{Amount in GBP} = \frac{300}{2} = 150 \text{ USD}$$

**Step 2**: Next, we convert this amount into pounds using the exchange rate (1 GBP = 1.64 USD):

$$\text{Amount in GBP in pounds} = \frac{150}{1.64} \approx 91.50 \text{ GBP}$$

Rounding to the nearest whole number, she will withdraw 92 pounds.
**Step 3**: Now, we calculate the amount to be withdrawn in euros:

$$\text{Amount in EUR} = \frac{300}{2} = 150 \text{ USD}$$

**Step 4**: Next, we convert this amount into euros using the exchange rate (1 EUR = 1.32 USD):

$$\text{Amount in EUR in euros} = \frac{150}{1.32} \approx 113.64 \text{ EUR}$$

Rounding to the nearest whole number, she will withdraw 114 euros.
**Step 5**: Finally, we find the difference between the amount of euros and pounds she will have:

$$\text{Difference} = 114 - 92 = 22$$

So, she will have $\boxed{22}$ more euros than pounds.

Answer:

After Kathy has exchanged half of her money ($300 USD$\div$2 = 150 USD) into pounds, she will have:

$$150 \text{ USD} \times \frac{1 \text{ pound}}{1.64 \text{ USD}} \approx 91.46 \text{ pounds}$$

After exchanging the other half of her money into euros, she will have:

$$150 \text{ USD} \times \frac{1 \text{ euro}}{1.32 \text{ USD}} \approx 113.64 \text{ euros}$$

Subtracting these two values, we have:

$$113.64 - 91.46 = 22.18$$

Since the question calls for the nearest whole number, we round 22.18 to the final answer of:

$$\boxed{22}$$

Figure 12: Example of Beam Search reasoning. The answer is correct.

The three question-and-answer sessions demonstrate that **Best-of-N** and **Beam Search Reasoning** correctly addressed the questions, understanding them thoroughly and providing accurate calculations. In contrast, **CoT Reasoning** misinterpreted the question, leading to calculation errors starting from **Step 2**. This highlights that **Best-of-N** and **Beam Search Reasoning** likely benefit from a comparatively larger search space, allowing them to explore a wider range of possible reasoning paths and converge on the correct solution.

## 5 Conclusion

In this work, we have introduced *OpenR*, an open-source framework designed to advance reasoning capabilities in large language models (LLMs) through the integration of test-time computation, reinforcement learning, and process supervision. Our framework provides an open and accessible platform for experimenting with reasoning tasks in LLMs, showcasing how test-time compute, search algorithms, and process reward models (PRMs) can be combined to improve reasoning performance.

We believe *OpenR* will serve as a valuable resource for the research community, offering a comprehensive platform for further exploration of reasoning in LLMs. By making our models, data, and code publicly available, we aim to accelerate advancements in AI reasoning, fostering collaboration and innovation in the field. In future work, we plan to extend the framework to support a wider range of reasoning tasks and optimise inference-time compute strategies for even more efficient and scalable reasoning models.

## References

OpanAI. Learning to reason with llms. https://openai.com/index/learning-to-reason-with-llms/, 2014,0912.

Daniel Kahneman. *Thinking, Fast and Slow*. Farrar, Straus and Giroux, New York, 2011.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*, 2022.

Maxwell Nye, Anders J Andreassen, Guy Gur-Ari, Henryk Michalewski, David Dohan, Jackie Jiang, John Schulman, William Fedus, and Charles Sutton. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*, 2021.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35: 27730–27744, 2022.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*, 2022.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, Jiao Sun, et al. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*, 2024.

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

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*, 2024.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

Xidong Feng, Ziyu Wan, Muning Wen, Ying Wen, Weinan Zhang, and Jun Wang. Alphazero-like tree-search can guide large language model decoding and training. In *ICML 2024*, 2024.

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

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. Think before you speak: Training language models with pause tokens. *arXiv preprint arXiv:2310.02226*, 2023.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021a.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Fei Yu, Anningzhe Gao, and Benyou Wang. Ovm, outcome-supervised value models for planning in mathematical reasoning. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 858–875, 2024.

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

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. Quiet-star: Language models can teach themselves to think before speaking. *arXiv preprint arXiv:2403.09629*, 2024.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9426–9439, 2024a.

Zihan Wang, Yunxuan Li, Yuexin Wu, Liangchen Luo, Le Hou, Hongkun Yu, and Jingbo Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*, 2024b.

Richard Bellman. Dynamic programming and stochastic control processes. *Information and control*, 1(3):228–239, 1958.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Martijn Van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement learning: State-of-the-art*, pages 3–42. Springer, 2012.

Thomas Carta, Clément Romac, Thomas Wolf, Sylvain Lamprier, Olivier Sigaud, and Pierre-Yves Oudeyer. Grounding large language models in interactive environments with online reinforcement learning. In *International Conference on Machine Learning*, pages 3676–3713. PMLR, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021b.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023.