Improve Mathematical Reasoning in Language Models by Automated Process Supervision

Liangchen Luo[1*], **Yinxiao Liu**[1*], **Rosanne Liu**[1], **Samrat Phatale**[1], **Meiqi Guo**[1], **Harsh Lara**[1], **Yunxuan Li**[2], **Lei Shu**[1], **Yun Zhu**[1], **Lei Meng**[2], **Jiao Sun**[2] **and Abhinav Rastogi**[1]
[1]Google DeepMind, [2]Google

Complex multi-step reasoning tasks, such as solving mathematical problems or generating code, remain a significant hurdle for even the most advanced large language models (LLMs). Verifying LLM outputs with an Outcome Reward Model (ORM) is a standard inference-time technique aimed at enhancing the reasoning performance of LLMs. However, this still proves insufficient for reasoning tasks with a lengthy or multi-hop reasoning chain, where the intermediate outcomes are neither properly rewarded nor penalized. Process supervision addresses this limitation by assigning intermediate rewards during the reasoning process. To date, the methods used to collect process supervision data have relied on either human annotation or per-step Monte Carlo estimation, both prohibitively expensive to scale, thus hindering the broad application of this technique. In response to this challenge, we propose a novel divide-and-conquer style Monte Carlo Tree Search (MCTS) algorithm named *OmegaPRM* for the efficient collection of high-quality process supervision data. This algorithm swiftly identifies the first error in the Chain of Thought (CoT) with binary search and balances the positive and negative examples, thereby ensuring both efficiency and quality. As a result, we are able to collect over 1.5 million process supervision annotations to train Process Reward Models (PRMs). This fully automated process supervision alongside the weighted self-consistency algorithm is able to enhance LLMs' math reasoning performances. We improved the success rates of the instruction-tuned Gemini Pro model from 51% to 69.4% on MATH500 and from 86.4% to 93.6% on GSM8K. Similarly, we boosted the success rates of Gemma2 27B from 42.3% to 58.2% on MATH500 and from 74.0% to 92.2% on GSM8K. The entire process operates without any human intervention or supervision, making our method both financially and computationally cost-effective compared to existing methods.

1. Introduction

Despite the impressive advancements achieved by scaling Large Language Models (LLMs) on established benchmarks (Wei et al., 2022a), cultivating more sophisticated reasoning capabilities, particularly in domains like mathematical problem-solving and code generation, remains an active research area. Chain-of-thought (CoT) generation is crucial for these reasoning tasks, as it decomposes complex problems into intermediate steps, mirroring human reasoning processes. Prompting LLMs with CoT examples (Wei et al., 2022b) and fine-tuning them on question-CoT solution pairs (Ouyang et al., 2022; Perez et al., 2021) have proven effective, with the latter demonstrating superior performance. Furthermore, the advent of Reinforcement Learning with Human Feedback (RLHF; Ouyang et al., 2022) has enabled the alignment of LLM behaviors with human preferences through reward models, significantly enhancing model capabilities.

Beyond prompting and further training, developing effective decoding strategies is another crucial avenue for improvement. Self-consistency decoding (Wang et al., 2023) leverages multiple reasoning paths to arrive at a voted answer. Incorporating a verifier, such as an off-the-shelf LLM (Huang et al., 2022; Luo et al., 2023), can further guide LLMs in reasoning tasks by providing a feedback loop to verify final answers, identify errors, and suggest corrections. However, the gain of such

*Equal contributors. Correspondence to {luolc,canoee}@google.com

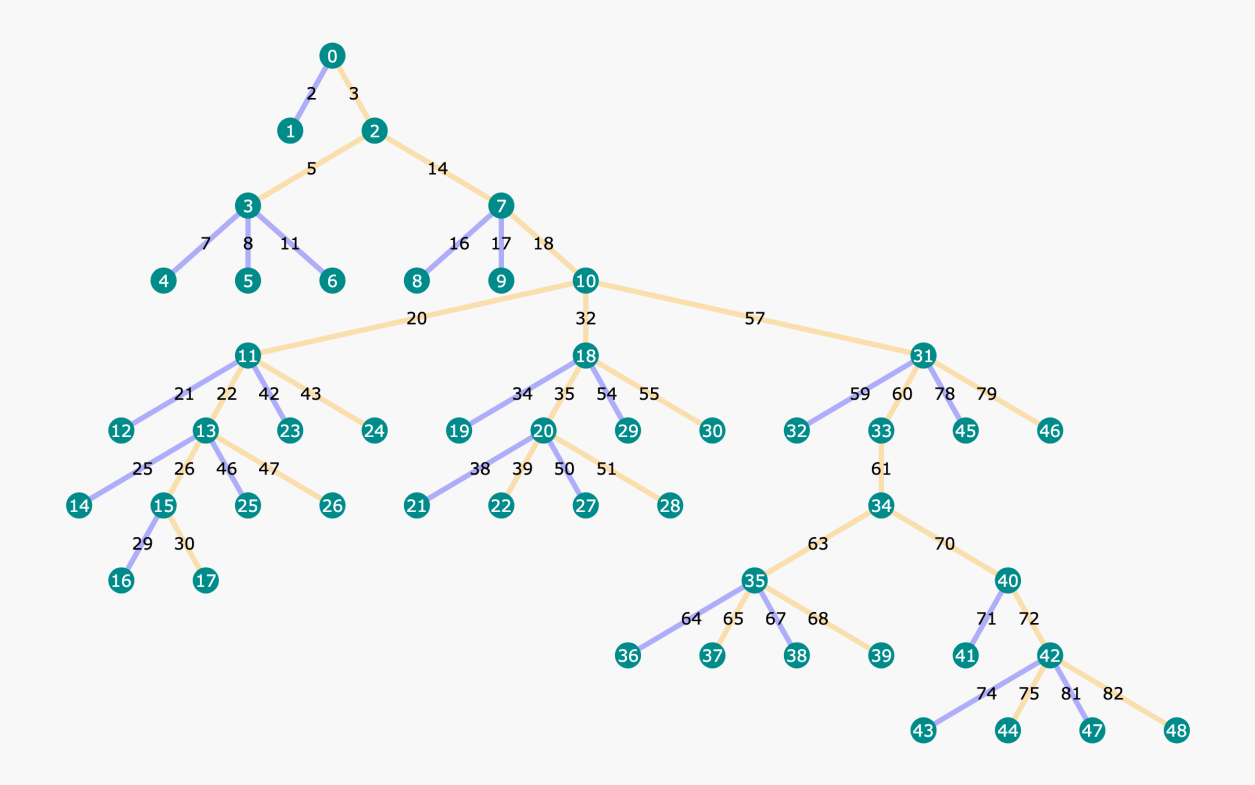


Figure 1 | Example tree structure built with our proposed OmegaPRM algorithm. Each node in the tree indicates a state of partial chain-of-thought solution, with information including accuracy of rollouts and other statistics. Each edge indicates an action, *i.e.*, a reasoning step, from the last state. Yellow edges are correct steps and blue edges are wrong.

approaches remains limited for complex multi-step reasoning problems. Reward models offer a promising alternative to verifiers, enabling the reranking of candidate outcomes based on reward signals to ensure higher accuracy. Two primary types of reward models have emerged: Outcome Reward Models (ORMs; Cobbe et al., 2021; Yu et al., 2024), which provide feedback only at the end of the problem-solving process, and Process Reward Models (PRMs; Li et al., 2023; Lightman et al., 2023; Uesato et al., 2022), which offer granular feedback at each reasoning step. PRMs have demonstrated superior effectiveness for complex reasoning tasks by providing such fine-grained supervision.

The primary bottleneck in developing PRMs lies in obtaining process supervision signals, which require supervised labels for each reasoning step. Current approaches rely heavily on costly and labor-intensive human annotation (Lightman et al., 2023; Uesato et al., 2022). Automating this process is crucial for scalability and efficiency. While recent efforts using per-step Monte Carlo estimation have shown promise (Wang et al., 2024a,b), their efficiency remains limited due to the vast search space. To address this challenge, we introduce OmegaPRM, a novel divide-and-conquer Monte Carlo Tree Search (MCTS) algorithm inspired by AlphaGo Zero (Silver et al., 2017) for automated process supervision data collection. For each question, we build a Monte Carlo Tree, as shown in Fig. 1, with the details explained in §3.3. This algorithm enables efficient collection of over 1.5 million high-quality process annotations without human intervention. Our PRM, trained on this dataset and combined with weighted self-consistency decoding, significantly improves the performance of instruction-tuned Gemini Pro from 51% to 69.4% on MATH500 (Lightman et al., 2023) and from 86.4% to 93.6% on GSM8K (Cobbe et al., 2021). We also boosted the success rates of Gemma2 27B from 42.3% to 58.2% on MATH500 and from 74.0% to 92.2% on GSM8K.

Our main contributions are as follows:

- We propose a novel divide-and-conquer style Monte Carlo Tree Search algorithm for automated process supervision data generation.
- The algorithm enables the efficient generation of over 1.5 million process supervision annotations, representing the largest and highest quality dataset of its kind to date. Additionally, the entire process operates without any human annotation, making our method both financially and computationally cost-effective.
- We combine our verifier with weighted self-consistency to further boost the performance of LLM reasoning. We significantly improves the success rates from 51% to 69.4% on MATH500 and from 86.4% to 93.6% on GSM8K for instruction-tuned Gemini Pro. For Gemma2 27B, we also improved the success rates of from 42.3% to 58.2% on MATH500 and from 74.0% to 92.2% on GSM8K.

2. Related Work

Improving mathematical reasoning ability of LLMs. Mathematical reasoning poses significant challenges for LLMs, and it is one of the key tasks for evaluating the reasoning ability of LLMs. With a huge amount of math problems in pretraining datasets, the pretrained LLMs (Gemini Team et al., 2024; OpenAI, 2023; Touvron et al., 2023) are able to solve simple problems, yet struggle with more complicated reasoning. To overcome that, the chain-of-thought (Fu et al., 2023; Wei et al., 2022b) type prompting algorithms were proposed. These techniques were effective in improving the performance of LLMs on reasoning tasks without modifying the model parameters. The performance was further improved by supervised fine-tuning (SFT; Cobbe et al., 2021; Liu et al., 2024; Yu et al., 2023) with high quality question-response pairs with full CoT reasoning steps.

Application of reward models in mathematical reasoning of LLMs. To further improve the LLM's math reasoning performance, verifiers can help to rank and select the best answer when multiple rollouts are available. Several works (Huang et al., 2022; Luo et al., 2023) have shown that using LLM as verifier is not suitable for math reasoning. For trained verifiers, two types of reward models are commonly used: Outcome Reward Model (ORM) and Process Reward Model (PRM). Both have shown performance boost on math reasoning over self-consistency (Cobbe et al., 2021; Lightman et al., 2023; Uesato et al., 2022), yet evidence has shown that PRM outperforms ORM (Lightman et al., 2023; Wang et al., 2024a). Generating high quality process supervision data is the key for training PRM, besides expensive human annotation (Lightman et al., 2023), Math-Shepherd (Wang et al., 2024a) and MiPS (Wang et al., 2024b) explored Monte Carlo estimation to automate the data collection process with human involvement, and both observed large performance gain. Our work shared the essence with MiPS and Math-Shepherd, but we explore further in collecting the process data using MCTS.

Monte Carlo Tree Search (MCTS). MCTS (Świechowski et al., 2021) has been widely adopted in reinforcement learning (RL). AlphaGo (Silver et al., 2016) and AlphaGo Zero (Silver et al., 2017) were able to achieve great performance with MCTS and deep reinforcement learning. For LLMs, there are planning algorithms that fall in the category of tree search, such as Tree-of-Thought (Yao et al., 2023) and Reasoning-via-Planing (Hao et al., 2023). Recently, utilizing tree-like decoding to find the best output during the inference-time has become a hot topic to explore as well, multiple works (Feng et al., 2023, 2024; Kang et al., 2024; Ma et al., 2023; Tian et al., 2024; Zhang et al., 2024) have observed improvements in reasoning tasks.

3. Methods

3.1. Process Supervision

Process supervision is a concept proposed to differentiate from outcome supervision. The reward models trained with these objectives are termed Process Reward Models (PRMs) and Outcome Reward Models (ORMs), respectively. In the ORM framework, given a query q (*e.g.*, a mathematical problem) and its corresponding response x (*e.g.*, a model-generated solution), an ORM is trained to predict the correctness of the final answer within the response. Formally, an ORM takes q and x and outputs the probability $p = \text{ORM}(q, x)$ that the final answer in the response is correct. With a training set of question-answer pairs available, an ORM can be trained by sampling outputs from a policy model (*e.g.*, a pretrained or fine-tuned LLM) using the questions and obtaining the correctness labels by comparing these outputs with the golden answers.

In contrast, a PRM is trained to predict the correctness of each intermediate step x_t in the solution. Formally, $p_t = \text{PRM}([q, x_{1:t-1}], x_t)$, where $x_{1:i} = [x_1, \ldots, x_i]$ represents the first i steps in the solution. This provides more precise and fine-grained feedback than ORMs, as it identifies the exact location of errors. Process supervision has also been shown to mitigate incorrect reasoning in the domain of mathematical problem solving. Despite these advantages, obtaining the intermediate signal for each step's correctness to train such a PRM is non-trivial. Previous work (Lightman et al., 2023) has relied on hiring domain experts to manually annotate the labels, which is and difficult to scale.

3.2. Process Annotation with Monte Carlo Method

In two closely related works, Math-Shepherd (Wang et al., 2024a) and MiPS (Wang et al., 2024b), the authors propose an automatic annotation approach to obtain process supervision signals using the Monte Carlo method. Specifically, a "completer" policy is established that can take a question q and a prefix solution comprising the first t steps $x_{1:t}$ and output the completion — often referred to as a "rollout" in reinforcement learning — of the subsequent steps until the final answer is reached. As shown in Fig. 2(a), for any step of a solution, the completer policy can be used to randomly sample k rollouts from that step. The final answers of these rollouts are compared to the golden answer, providing k labels of answer correctness corresponding to the k rollouts. Subsequently, the ratio of correct rollouts to total rollouts from the t-th step, as represented in Eq. (1), estimates the "correctness level" of the prefix steps up to t. Regardless of false positives, $x_{1:t}$ should be considered correct as long as any of the rollouts is correct in the logical reasoning scenario.

$$c_t = \text{MonteCarlo}(q, x_{1:t}) = \frac{\text{num(correct rollouts from } t\text{-th step)}}{\text{num(total rollouts from } t\text{-th step)}} \tag{1}$$

Taking a step forward, a straightforward strategy to annotate the correctness of intermediate steps in a solution is to perform rollouts for every step from the beginning to the end, as done in both Math-Shepherd and MiPS. However, this brute-force approach requires a large number of policy calls. To optimize annotation efficiency, we propose a binary-search-based Monte Carlo estimation.

Monte Carlo estimation using binary search. As suggested by Lightman et al. (2023), supervising up to the first incorrect step in a solution is sufficient to train a PRM. Therefore, our objective is locating the first error in an efficient way. We achieve this by repeatedly dividing the solution and performing rollouts. Assuming no false positives or negatives, we start with a solution with potential errors and split it at the midpoint m. We then perform rollouts for $s_{1:m}$ with two possible outcomes: (1) $c_m > 0$, indicating that the first half of the solution is correct, as at least one correct answer can be rolled out from m-th step, and thus the error is in the second half; (2) $c_m = 0$, indicating the error is

very likely in the first half, as none of the rollouts from m-th step is correct. This process narrows down the error location to either the first or second half of the solution. As shown in Fig. 2(b), by repeating this process on the erroneous half iteratively until the partial solution is sufficiently small (*i.e.*, short enough to be considered as a single step), we can locate the first error with a time complexity of $O(k \log M)$ rather than $O(kM)$ in the brute-force setting, where M is the total number of steps in the original solution.

3.3. Monte Carlo Tree Search

Although binary search improves the efficiency of locating the first error in a solution, we are still not fully utilizing policy calls as rollouts are simply discarded after stepwise Monte Carlo estimation. In practice, it is necessary to collect multiple PRM training examples (*a.k.a.*, triplets of question, partial solution and correctness label) for a question (Lightman et al., 2023; Wang et al., 2024a). Instead of starting from scratch each time, we can store all rollouts during the process and conduct binary searches from any of these rollouts whenever we need to collect a new example. This approach allows for triplets with the same solution prefix but different completions and error locations. Such reasoning structures can be represented as a tree, as described in previous work like Tree of Thought (Yao et al., 2023).

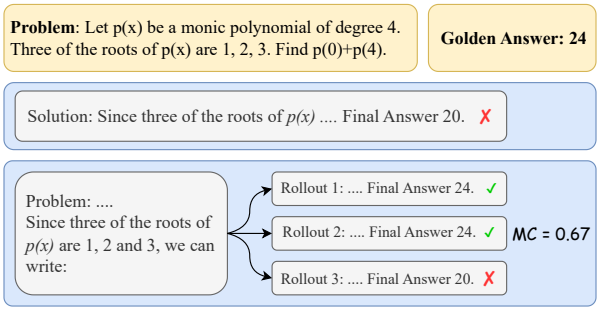
Formally, consider a *state-action tree* representing detailed reasoning paths for a question, where a state s contains the question and all preceding reasoning steps, and an action a is a potential subsequent step from a specific state. The root state is the question without any reasoning steps: $r_{\text{root}} = q$. The policy can be directly modeled by a language model as $\pi(a|s) = \text{LM}(a|s)$, and the state transition function is simply the concatenation of the preceding steps and the action step, *i.e.*, $s' = \text{Concatenate}(s, a)$.

Collecting PRM training examples for a question can now be formulated as constructing such a state-action tree. This reminds us the classic Monte Carlo Tree Search (MCTS) algorithm, which has been successful in many deep reinforcement learning applications (Silver et al., 2016, 2017). However, there are some key differences when using a language model as the policy. First, MCTS typically handles an environment with a finite action space, such as the game of Go, which has fewer than 361 possible actions per state (Silver et al., 2017). In contrast, an LM policy has an infinite action space, as it can generate an unlimited number of distinct actions (sequences of tokens) given a prompt. In practice, we use temperature sampling to generate a fix number of k completions for a prompt, treating the group of k actions as an approximate action space. Second, an LM policy can sample a full rollout until the termination state (*i.e.*, reaching the final answer) without too much overhead than generating a single step, enabling the possibility of binary search. Consequently, we propose an adaptation of the MCTS algorithm named **OmegaPRM**, primarily based on the one introduced in AlphaGo (Silver et al., 2016), but with modifications to better accommodate the scenario of PRM training data collection. We describe the algorithm details as below.

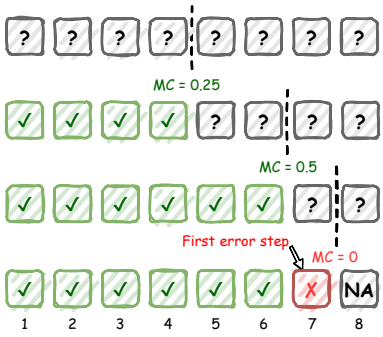
Tree Structure. Each node s in the tree contains the question q and prefix solution $x_{1:t}$, together with all previous rollouts $\{(s, r_i)\}_{i=1}^{k}$ from the state. Each edge (s, a) is either a single step or a sequence of consecutive steps from the node s. The nodes also store a set of statistics,

$$\{N(s), \text{MC}(s), Q(s, r)\},$$

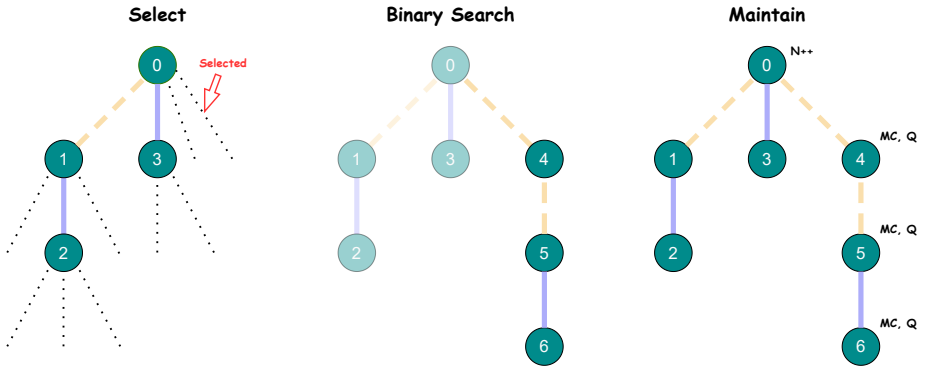
where $N(s)$ denotes the visit count of a state, $\text{MC}(s)$ represents the Monte Carlo estimation of a state as specified in Eq. (1), and $Q(s, r)$ is a state-rollout value function that is correlated to the chance of selecting a rollout during the selection phase of tree traversal. Specifically,



(a) Monte Carlo estimation of a prefix solution.



(b) Error locating using binary search.



(c) Three stages in an iteration of the MCTS process.

Figure 2 | Illustration of the process supervision rollouts, Monte Carlo estimation using binary search and the MCTS process. (a) An example of Monte Carlo estimation of a prefix solution. Two out of the three rollouts are correct, producing the Monte Carlo estimation $\text{MC}(q, x_{1:t}) = 2/3 \approx 0.67$. (b) An example of error locating using binary search. The first error step is located at the 7^{th} step after three divide-and-rollouts, where the rollout positions are indicated by the vertical dashed lines. (c) The MCTS process. The dotted lines in Select stage represent the available rollouts for binary search. The bold colored edges represent steps with correctness estimations. The yellow color indicates a correct step, *i.e.*, with a preceding state s that $\text{MC}(s) > 0$ and the blue color indicates an incorrect step, *i.e.*, with $\text{MC}(s) = 0$. The number of dashes in each colored edge indicates the number of steps.

$$Q(s, r) = \alpha^{1-\text{MC}(s)} \cdot \beta^{\frac{\text{len}(r)}{L}}, \tag{2}$$

where $\alpha, \beta \in (0, 1]$ and $L > 0$ are constant hyperparameters; while $\text{len}(r)$ denotes the length of a rollout in terms of number of tokens. Q is supposed to indicate how likely a rollout will be chosen for each iteration and our goal is to define a heuristic that selects the most valuable rollout to search with. The most straightforward strategy is uniformly choosing rollout candidates generated by the policy in previous rounds; however, this is obviously not an effective way. Lightman et al. (2023) suggests surfacing the *convincing wrong-answer* solutions for annotators during labeling. Inspired by this, we propose to prioritize *supposed-to-be-correct wrong-answer* rollouts during selection. We use the term *supposed-to-be-correct* to refer to the state with a Monte Carlo estimation $\text{MC}(s)$ closed to 1; and use *wrong-answer* to refer that the specific rollout r has a wrong final answer. The rollout contains mistakes made by the policy that should have been avoided given its high $\text{MC}(s)$. We expect

a PRM that learns to detect errors in such rollouts will be more useful in correcting the mistakes made by the policy. The first component in Eq. (2), $\alpha^{1-\text{MC}(s)}$, has a larger value as MC(s) is closer to 1. Additionally, we incorporate a length penalty factor $\beta^{\frac{\text{len}(r)}{L}}$, to penalize excessively long rollouts.

Select. The selection phase in our algorithm is simpler than that of AlphaGo (Silver et al., 2016), which involves selecting a sequence of actions from the root to a leaf node, forming a trajectory with multiple states and actions. In contrast, we maintain a pool of all rollouts $\{(s_i, r_j^i)\}$ from previous searches that satisfy $0 < \text{MC}(s_i) < 1$. During each selection, a rollout is popped and selected according to tree statistics, $(s, r) = \arg\max_{(s,r)}[Q(s, r) + U(s)]$, using a variant of the PUCT (Rosin, 2011) algorithm,

$$U(s) = c_{\text{puct}} \frac{\sqrt{\sum_i N(s_i)}}{1 + N(s)}, \tag{3}$$

where c_{puct} is a constant determining the level of exploration. This strategy initially favors rollouts with low visit counts but gradually shifts preference towards those with high rollout values.

Binary Search. We perform a binary search to identify the first error location in the selected rollout, as detailed in §3.2. The rollouts with $0 < \text{MC}(s) < 1$ during the process are added to the selection candidate pool. All divide-and-rollout positions before the first error become new states. For the example in Fig. 2(b), the trajectory $s[q] \to s[q, x_{1:4}] \to s[q, x_{1:6}] \to s[q, x_{1:7}]$ is added to the tree after the binary search. The edges $s[q] \to s[q, x_{1:4}]$ and $s[q, x_{1:4}] \to s[q, x_{1:6}]$ are correct, with MC values of 0.25 and 0.5, respectively; while the edge $s[q, x_{1:6}] \to s[q, x_{1:7}]$ is incorrect with MC value of 0.

Maintain. After the binary search, the tree statistics $N(s)$, MC(s), and $Q(s, r)$ are updated. Specifically, $N(s)$ is incremented by 1 for the selected (s, r). Both MC(s) and $Q(s, r)$ are updated for the new rollouts sampled from the binary search. This phase resembles the *backup* phase in AlphaGo but is simpler, as it does not require recursive updates from the leaf to the root.

Tree Construction. By repeating the aboved process, we can construct a state-action tree as the example illustrated in Fig. 1. The construction ends either when the search count reaches a predetermined limit or when no additional rollout candidates are available in the pool.

3.4. PRM Training

Each edge (s, a) with a single-step action in the constructed state-action tree can serve as a training example for the PRM. It can be trained using the standard classification loss

$$\mathcal{L}_{\text{pointwise}} = \sum_{i=1}^{N} \hat{y}_i \log y_i + (1 - \hat{y}_i) \log(1 - y_i), \tag{4}$$

where $\hat{y}_i$ represents the correctness label and $y_i = \text{PRM}(s, a)$ is the prediction score of the PRM. Wang et al. (2024b) have used the Monte Carlo estimation as the correctness label, denoted as $\hat{y} = \text{MC}(s)$. Alternatively, Wang et al. (2024a) have employed a binary labeling approach, where $\hat{y} = \mathbf{1}[\text{MC}(s) > 0]$, assigning $\hat{y} = 1$ for any positive Monte Carlo estimation and $\hat{y} = 0$ otherwise. We refer the former option as *pointwise soft* label and the latter as *pointwise hard* label. In addition, considering there are many cases where a common solution prefix has multiple single-step actions, we can also minimize the cross-entropy loss between the PRM predictions and the normalized pairwise preferences following

the Bradley-Terry model (Christiano et al., 2017). We refer this training method as *pairwise* approach, and the detailed pairwise loss formula can be found in Section Appendix B.

We use the pointwise soft label when evaluating the main results in §4.1, and a comparion of the three objectives are discussed in §4.3.

4. Experiments

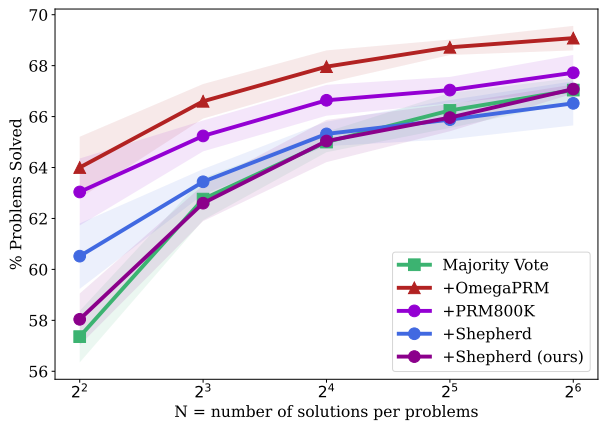
Data Generation. We conduct our experiments on the challenging MATH dataset (Hendrycks et al., 2021). We use the same training and testing split as described in Lightman et al. (2023), which consists of 12K training examples and a subset with 500 holdout representative problems from the original 5K testing examples introduced in Hendrycks et al. (2021). We observe similar policy performance on the full test set and the subset. For creating the process annotation data, we use the questions from the training split and set the search limit to 100 per question, resulting 1.5M per-step process supervision annotations. To reduce the false positive and false negative noise, we filtered out questions that are either too hard or too easy for the model. Please refer to Appendix A for details. We use $\alpha = 0.5$, $\beta = 0.9$ and $L = 500$ for calculating $Q(s, r)$ in Eq. (2); and $c_{\text{puct}} = 0.125$ in Eq. (3). We sample $k = 8$ rollouts for each Monte Carlo estimation.

Models. In previous studies (Lightman et al., 2023; Wang et al., 2024a,b), both proprietary models such as GPT-4 (OpenAI, 2023) and open-source models such as Llama2 (Touvron et al., 2023) were explored. In our study, we perform experiments with both proprietary Gemini Pro (Gemini Team et al., 2024) and open-source Gemma2 (Gemma Team et al., 2024) models. For Gemini Pro, we follow Lightman et al. (2023); Wang et al. (2024a) to initially fine-tune it on math instruction data, achieving an accuracy of approximately 51% on the MATH test set. The instruction-tuned model is then used for solution sampling. For open-source models, to maximize reproducibility, we directly use the pretrained Gemma2 27B checkpoint with the 4-shot prompt introduced in Gemini Team et al. (2024). The reward models are all trained from the pretrained checkpoints.

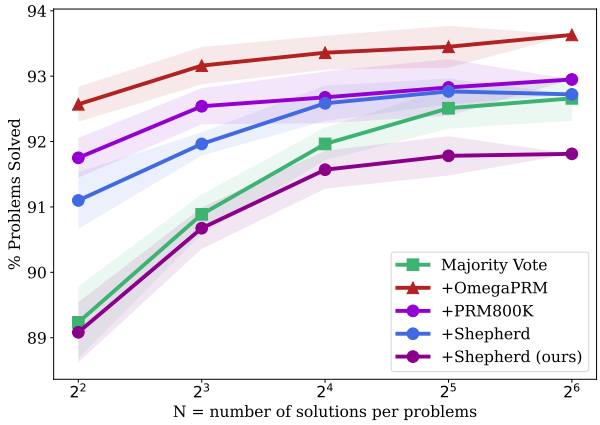
Metrics and baselines. We evaluate the PRM-based majority voting results on GSM8K (Cobbe et al., 2021) and MATH500 (Lightman et al., 2023) using PRMs trained on different process supervision data. We choose the product of scores across all steps as the final solution score following Lightman et al. (2023), where the performance difference between product and minimum of scores was compared and the study showed the difference is minor. Baseline process supervision data include PRM800K (Lightman et al., 2023) and Math-Shepherd (Wang et al., 2024a), both publicly available. Additionally, we generate a process annotation dataset with our Gemini policy model using the brute-force approach described in Wang et al. (2024a,b), referred to as Math-Shepherd (our impl) in subsequent sections.

4.1. Main Results

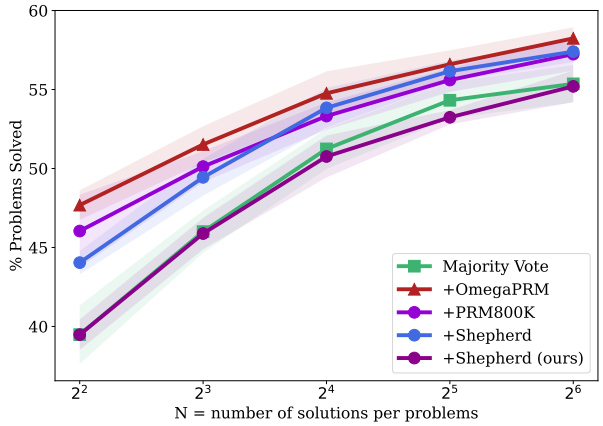
Table 1 and Fig. 3 presents the performance comparison of PRMs trained on various process annotation datasets. OmegaPRM consistently outperforms the other process supervision datasets. Specifically, the fine-tuned Gemini Pro achieves 69.4% and 93.6% accuracy on MATH500 and GSM8K, respectively, using OmegaPRM-weighted majority voting. For the pretrained Gemma2 27B, it also performs the best with 58.2% and 92.2% accuracy on MATH500 and GSM8K, respectively. It shows superior performance comparing to both human annotated PRM800K but also automatic annotated Math-Shepherd. More specifically, when the number of samples is small, almost all the PRM models outperforme the majority vote. However, as the number of samples increases, the performance of



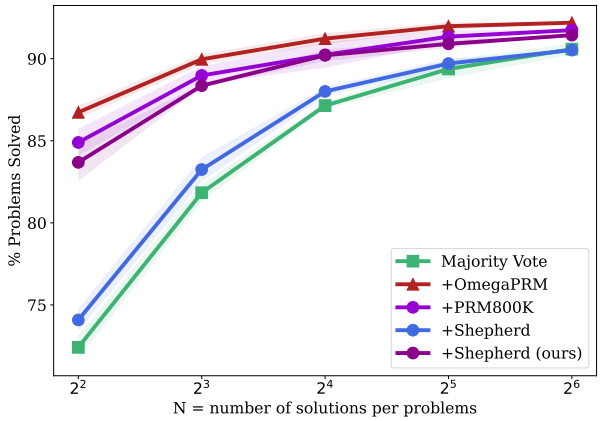
(a) Gemini Pro on MATH500.



(b) Gemini Pro on GSM8K.



(c) Gemma2 27B on MATH500.



(d) Gemma2 27B on GSM8K.

Figure 3 | A comparison of PRMs trained with different process supervision datasets, evaluated by their ability to search over many test solutions using a PRM-weighted majority voting. We visualize the variance across many sub-samples of the 128 solutions we generated in total per problem.

Table 1 | The performance comparison of PRMs trained with different process supervision datasets. The numbers represent the percentage of problems solved using PRM-weighted majority voting with $k = 64$.

	MATH500		GSM8K	
	Gemini Pro	Gemma 2 27B	Gemini Pro	Gemma 2 27B
MajorityVote@64	67.2	54.7	92.7	90.6
+ Math-Shepherd	67.2	57.4	92.7	90.5
+ Math-Shepherd (our impl)	67.2	55.2	91.8	91.4
+ PRM800K	67.6	57.2	92.9	91.7
+ OmegaPRM	**69.4**	**58.2**	**93.6**	**92.2**

other PRMs gradually converges to the same level of the majority vote. In contrast, our PRM model continues to demonstrate a clear margin of accuracy.

4.2. Step Distribution

An important factor in process supervision is the number of steps in a solution and the length of each step. Previous works (Lightman et al., 2023; Wang et al., 2024a,b) use rule-based strategies to split a solution into steps, *e.g.*, using newline as delimiters. In contrast, we propose a more flexible method for step division, treating any sequence of consecutive tokens in a solution as a valid step. We observe that many step divisions in Math-Shepherd lack semantic coherence to some extent. Therefore, we hypothesize that semantically explicit cutting is not necessary for training a PRM.

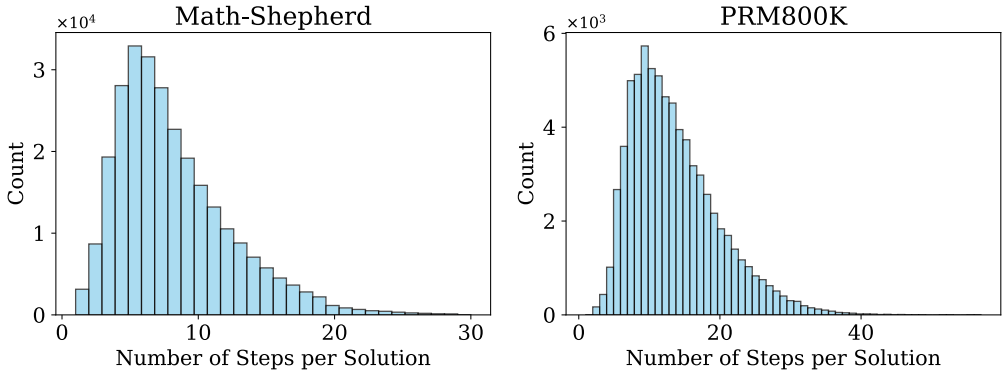


Figure 4 | Number of steps distribution.

In practice, we first examine the distribution of the number of steps per solution in PRM800K and Math-Shepherd, as shown in Fig. 4, noting that most solutions have less than 20 steps. During binary search, we aim to divide a full solution into 16 pieces. To calculate the expected step length, we divide the average solution length by 16. The binary search terminates when a step is shorter than this value. The resulting distributions of step lengths for OmegaPRM and the other two datasets are shown in Fig. 5. This flexible splitting strategy produces a step length distribution similar to that of the rule-based strategy.

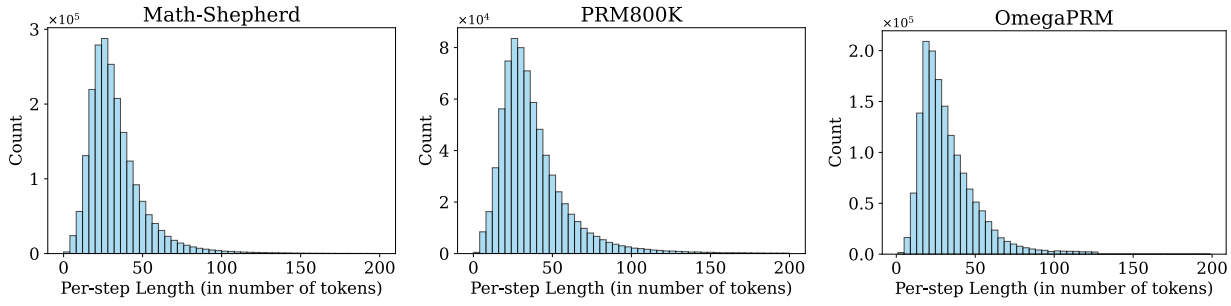


Figure 5 | Step length distribution in terms of number of tokens.

4.3. PRM Training Objectives

Table 2 | Comparison of different training objectives for PRMs.

	Soft Label	Hard Label	Pairwise
PRM Accuracy (%)	**70.1**	63.3	64.2

As outlined in §3.4, PRMs can be trained using multiple objectives. We construct a small process

supervision test set using the problems from the MATH test split. We train PRMs using pointwise soft label, pointwise hard label and pairwise loss respectively, and evaluate how accurately they can classify the per-step correctness. Table 2 presents the comparison of different objectives, and the pointwise soft label is the best among them with 70.1% accuracy.

4.4. Algorithm Efficiency

As described in Section §3.2 and §3.3, we utilize binary search and Monte Carlo Tree Search to improve the efficiency of OmegaPRM process supervision data collection by effectively identifying the first incorrect step and reusing rollouts in Monte Carlo estimation. To quantitatively measure the efficiency of OmegaPRM, we collected process supervision data using both brute-force-style method (Wang et al., 2024a,b) and OmegaPRM with the same computational budget. As a result, we were able to generate 200K data points using the brute-force algorithm compared to 15 million data points with OmegaPRM, demonstrating a 75-times efficiency improvement. In practice, we randomly down-sampled OmegaPRM data to 1.5 million for PRM training.

5. Limitations

There are some limitations with our paper, which we reserve for future work:

Automatic process annotation is noisy. Our method for automatic process supervision annotation introduces noise in the form of false positives and negatives, but experiments indicate that it can still effectively train a PRM. The PRM trained on our dataset performs better than one trained on the human-annotated PRM800K dataset. The precise impact of noise on PRM performance remains uncertain. For future research, a comprehensive comparison of human and automated annotations should be conducted. One other idea is to integrate human and automated annotations, which could result in more robust and efficient process supervision.

Human supervision is still necessary. Unlike the work presented in AlphaGo Zero (Silver et al., 2017), our method requires the question and golden answer pair. The question is necessary for LLM to start the MCTS and the golden answer is inevitable for the LLM to compare its rollouts with and determine the correctness of the current step. This will limit the method to the tasks with such question and golden answer pairs. Therefore, we need to adapt the current method further to make it suitable for open-ended tasks.

6. Conclusion

In conclusion, we introduce OmegaPRM, a divide-and-conquer Monte Carlo Tree Search algorithm, designed to automate the process supervision data collection for LLMs. By efficiently pinpointing the first error in the Chain-of-Thought and balancing data quality, OmegaPRM addresses the shortcomings of existing methods. Our automated approach enables the collection of over 1.5 million process supervision annotations, which are used to train a PRM. Leveraging this automated process supervision with the weighted self-consistency algorithm, we improve LLM mathematical reasoning performance, achieving a 69.4% success rate on the MATH benchmark — a 18.4% absolute increase over the base model which amounts to a relative improvement of 36%. Additionally, our method significantly reduces data collection costs compared to human annotation and brute force Monte-Carlo sampling. These findings highlight OmegaPRM's potential to enhance LLM capabilities in complex multi-step reasoning tasks.

7. Acknowledgements

We would like to express our deepest gratitude to Matt Barnes, Jindong Chen and Rif A. Saurous, for their support and helpful feedback to our work. We also thank Peiyi Wang for clarifying the details in Math-Shepherd and providing insightful suggestions.

References

P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep reinforcement learning from human preferences. *arXiv preprint arXiv:1706.03741*, 2017.

K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano, C. Hesse, and J. Schulman. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

X. Feng, Z. Wan, M. Wen, S. M. McAleer, Y. Wen, W. Zhang, and J. Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2023.

X. Feng, Z. Wan, M. Wen, S. M. McAleer, Y. Wen, W. Zhang, and J. Wang. Alphazero-like tree-search can guide large language model decoding and training. *arXiv preprint arXiv:2309.17179*, 2024.

Y. Fu, H. Peng, A. Sabharwal, P. Clark, and T. Khot. Complexity-based prompting for multi-step reasoning. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, May 2023.

Gemini Team, M. Reid, N. Savinov, D. Teplyashin, Dmitry, Lepikhin, T. Lillicrap, J. baptiste Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser, I. Antonoglou, R. Anil, S. Borgeaud, A. Dai, K. Millican, E. Dyer, M. Glaese, T. Sottiaux, B. Lee, F. Viola, M. Reynolds, Y. Xu, J. Molloy, J. Chen, M. Isard, P. Barham, T. Hennigan, R. McIlroy, M. Johnson, J. Schalkwyk, E. Collins, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, C. Meyer, G. Thornton, Z. Yang, H. Michalewski, Z. Abbas, N. Schucher, A. Anand, R. Ives, J. Keeling, K. Lenc, S. Haykal, S. Shakeri, P. Shyam, A. Chowdhery, R. Ring, S. Spencer, E. Sezener, L. Vilnis, O. Chang, N. Morioka, G. Tucker, C. Zheng, O. Woodman, N. Attaluri, T. Kocisky, E. Eltyshev, X. Chen, T. Chung, V. Selo, S. Brahma, P. Georgiev, A. Slone, Z. Zhu, J. Lottes, S. Qiao, B. Caine, S. Riedel, A. Tomala, M. Chadwick, J. Love, P. Choy, S. Mittal, N. Houlsby, Y. Tang, M. Lamm, L. Bai, Q. Zhang, L. He, Y. Cheng, P. Humphreys, Y. Li, S. Brin, A. Cassirer, Y. Miao, L. Zilka, T. Tobin, K. Xu, L. Proleev, D. Sohn, A. Magni, L. A. Hendricks, I. Gao, S. Ontanon, O. Bunyan, N. Byrd, A. Sharma, B. Zhang, M. Pinto, R. Sinha, H. Mehta, D. Jia, S. Caelles, A. Webson, A. Morris, B. Roelofs, Y. Ding, R. Strudel, X. Xiong, M. Ritter, M. Dehghani, R. Chaabouni, A. Karmarkar, G. Lai, F. Mentzer, B. Xu, Y. Li, Y. Zhang, T. L. Paine, A. Goldin, B. Neyshabur, K. Baumli, A. Levskaya, M. Laskin, W. Jia, J. W. Rae, K. Xiao, A. He, S. Giordano, L. Yagati, J.-B. Lespiau, P. Natsev, S. Ganapathy, F. Liu, D. Martins, N. Chen, Y. Xu, M. Barnes, R. May, A. Vezer, J. Oh, K. Franko, S. Bridgers, R. Zhao, B. Wu, B. Mustafa, S. Sechrist, E. Parisotto, T. S. Pillai, C. Larkin, C. Gu, C. Sorokin, M. Krikun, A. Guseynov, J. Landon, R. Datta, A. Pritzel, P. Thacker, F. Yang, K. Hui, A. Hauth, C.-K. Yeh, D. Barker, J. Mao-Jones, S. Austin, H. Sheahan, P. Schuh, J. Svensson, R. Jain, V. Ramasesh, A. Briukhov, D.-W. Chung, T. von Glehn, C. Butterfield, P. Jhakra, M. Wiethoff, J. Frye, J. Grimstad, B. Changpinyo, C. L. Lan, A. Bortsova, Y. Wu, P. Voigtlaender, T. Sainath, S. Gu, C. Smith, W. Hawkins, K. Cao, J. Besley, S. Srinivasan, M. Omernick, C. Gaffney, G. Surita, R. Burnell, B. Damoc, J. Ahn, A. Brock, M. Pajarskas, A. Petrushkina, S. Noury, L. Blanco, K. Swersky, A. Ahuja, T. Avrahami, V. Misra, R. de Liedekerke, M. Iinuma, A. Polozov, S. York, G. van den Driessche, P. Michel, J. Chiu, R. Blevins, Z. Gleicher, A. Recasens, A. Rrustemi, E. Gribovskaya, A. Roy, W. Gworek, S. M. R. Arnold, L. Lee, J. Lee-Thorp, M. Maggioni, E. Piqueras, K. Badola,

S. Vikram, L. Gonzalez, A. Baddepudi, E. Senter, J. Devlin, J. Qin, M. Azzam, M. Trebacz, M. Polacek, K. Krishnakumar, S. yiin Chang, M. Tung, I. Penchev, R. Joshi, K. Olszewska, C. Muir, M. Wirth, A. J. Hartman, J. Newlan, S. Kashem, V. Bolina, E. Dabir, J. van Amersfoort, Z. Ahmed, J. Cobon-Kerr, A. Kamath, A. M. Hrafnkelsson, L. Hou, I. Mackinnon, A. Frechette, E. Noland, X. Si, E. Taropa, D. Li, P. Crone, A. Gulati, S. Cevey, J. Adler, A. Ma, D. Silver, S. Tokumine, R. Powell, S. Lee, K. Vodrahalli, S. Hassan, D. Mincu, A. Yang, N. Levine, J. Brennan, M. Wang, S. Hodkinson, J. Zhao, J. Lipschultz, A. Pope, M. B. Chang, C. Li, L. E. Shafey, M. Paganini, S. Douglas, B. Bohnet, F. Pardo, S. Odoom, M. Rosca, C. N. dos Santos, K. Soparkar, A. Guez, T. Hudson, S. Hansen, C. Asawaroengchai, R. Addanki, T. Yu, W. Stokowiec, M. Khan, J. Gilmer, J. Lee, C. G. Bostock, K. Rong, J. Caton, P. Pejman, F. Pavetic, G. Brown, V. Sharma, M. Lučić, R. Samuel, J. Djolonga, A. Mandhane, L. L. Sjösund, E. Buchatskaya, E. White, N. Clay, J. Jiang, H. Lim, R. Hemsley, Z. Cankara, J. Labanowski, N. D. Cao, D. Steiner, S. H. Hashemi, J. Austin, A. Gergely, T. Blyth, J. Stanton, K. Shivakumar, A. Siddhant, A. Andreassen, C. Araya, N. Sethi, R. Shivanna, S. Hand, A. Bapna, A. Khodaei, A. Miech, G. Tanzer, A. Swing, S. Thakoor, L. Aroyo, Z. Pan, Z. Nado, J. Sygnowski, S. Winkler, D. Yu, M. Saleh, L. Maggiore, Y. Bansal, X. Garcia, M. Kazemi, P. Patil, I. Dasgupta, I. Barr, M. Giang, T. Kagohara, I. Danihelka, A. Marathe, V. Feinberg, M. Elhawaty, N. Ghelani, D. Horgan, H. Miller, L. Walker, R. Tanburn, M. Tariq, D. Shrivastava, F. Xia, Q. Wang, C.-C. Chiu, Z. Ashwood, K. Baatarsukh, S. Samangooei, R. L. Kaufman, F. Alcober, A. Stjerngren, P. Komarek, K. Tsihlas, A. Boral, R. Comanescu, J. Chen, R. Liu, C. Welty, D. Bloxwich, C. Chen, Y. Sun, F. Feng, M. Mauger, X. Dotiwalla, V. Hellendoorn, M. Sharman, I. Zheng, K. Haridasan, G. Barth-Maron, C. Swanson, D. Rogozińska, A. Andreev, P. K. Rubenstein, R. Sang, D. Hurt, G. Elsayed, R. Wang, D. Lacey, A. Ilić, Y. Zhao, A. Iwanicki, A. Lince, A. Chen, C. Lyu, C. Lebsack, J. Griffith, M. Gaba, P. Sandhu, P. Chen, A. Koop, R. Rajwar, S. H. Yeganeh, S. Chang, R. Zhu, S. Radpour, E. Davoodi, V. I. Lei, Y. Xu, D. Toyama, C. Segal, M. Wicke, H. Lin, A. Bulanova, A. P. Badia, N. Rakićević, P. Sprechmann, A. Filos, S. Hou, V. Campos, N. Kassner, D. Sachan, M. Fortunato, C. Iwuanyanwu, V. Nikolaev, B. Lakshminarayanan, S. Jazayeri, M. Varadarajan, C. Tekur, D. Fritz, M. Khalman, D. Reitter, K. Dasgupta, S. Sarcar, T. Ornduff, J. Snaider, F. Huot, J. Jia, R. Kemp, N. Trdin, A. Vijayakumar, L. Kim, C. Angermueller, L. Lao, T. Liu, H. Zhang, D. Engel, S. Greene, A. White, J. Austin, L. Taylor, S. Ashraf, D. Liu, M. Georgaki, I. Cai, Y. Kulizhskaya, S. Goenka, B. Saeta, Y. Xu, C. Frank, D. de Cesare, B. Robenek, H. Richardson, M. Alnahlawi, C. Yew, P. Ponnapalli, M. Tagliasacchi, A. Korchemniy, Y. Kim, D. Li, B. Rosgen, K. Levin, J. Wiesner, P. Banzal, P. Srinivasan, H. Yu, Çağlar Ünlü, D. Reid, Z. Tung, D. Finchelstein, R. Kumar, A. Elisseeff, J. Huang, M. Zhang, R. Aguilar, M. Giménez, J. Xia, O. Dousse, W. Gierke, D. Yates, K. Jalan, L. Li, E. Latorre-Chimoto, D. D. Nguyen, K. Durden, P. Kallakuri, Y. Liu, M. Johnson, T. Tsai, A. Talbert, J. Liu, A. Neitz, C. Elkind, M. Selvi, M. Jasarevic, L. B. Soares, A. Cui, P. Wang, A. W. Wang, X. Ye, K. Kallarackal, L. Loher, H. Lam, J. Broder, D. Holtmann-Rice, N. Martin, B. Ramadhana, M. Shukla, S. Basu, A. Mohan, N. Fernando, N. Fiedel, K. Paterson, H. Li, A. Garg, J. Park, D. Choi, D. Wu, S. Singh, Z. Zhang, A. Globerson, L. Yu, J. Carpenter, F. de Chaumont Quitry, C. Radebaugh, C.-C. Lin, A. Tudor, P. Shroff, D. Garmon, D. Du, N. Vats, H. Lu, S. Iqbal, A. Yakubovich, N. Tripuraneni, J. Manyika, H. Qureshi, N. Hua, C. Ngani, M. A. Raad, H. Forbes, J. Stanway, M. Sundararajan, V. Ungureanu, C. Bishop, Y. Li, B. Venkatraman, B. Li, C. Thornton, S. Scellato, N. Gupta, Y. Wang, I. Tenney, X. Wu, A. Shenoy, G. Carvajal, D. G. Wright, B. Bariach, Z. Xiao, P. Hawkins, S. Dalmia, C. Farabet, P. Valenzuela, Q. Yuan, A. Agarwal, M. Chen, W. Kim, B. Hulse, N. Dukkipati, A. Paszke, A. Bolt, K. Choo, J. Beattie, J. Prendki, H. Vashisht, R. Santamaria-Fernandez, L. C. Cobo, J. Wilkiewicz, D. Madras, A. Elqursh, G. Uy, K. Ramirez, M. Harvey, T. Liechty, H. Zen, J. Seibert, C. H. Hu, A. Khorlin, M. Le, A. Aharoni, M. Li, L. Wang, S. Kumar, N. Casagrande, J. Hoover, D. E. Badawy, D. Soergel, D. Vnukov, M. Miecnikowski, J. Simsa, P. Kumar, T. Sellam, D. Vlasic, S. Daruki, N. Shabat, J. Zhang, G. Su, J. Zhang, J. Liu, Y. Sun, E. Palmer, A. Ghaffarkhah, X. Xiong, V. Cotruta, M. Fink, L. Dixon, A. Sreevatsa, A. Goedeckemeyer, A. Dimitriev, M. Jafari, R. Crocker, N. FitzGerald,

A. Kumar, S. Ghemawat, I. Philips, F. Liu, Y. Liang, R. Sterneck, A. Repina, M. Wu, L. Knight, M. Georgiev, H. Lee, H. Askham, A. Chakladar, A. Louis, C. Crous, H. Cate, D. Petrova, M. Quinn, D. Owusu-Afriyie, A. Singhal, N. Wei, S. Kim, D. Vincent, M. Nasr, C. A. Choquette-Choo, R. Tojo, S. Lu, D. de Las Casas, Y. Cheng, T. Bolukbasi, K. Lee, S. Fatehi, R. Ananthanarayanan, M. Patel, C. Kaed, J. Li, S. R. Belle, Z. Chen, J. Konzelmann, S. Põder, R. Garg, V. Koverkathu, A. Brown, C. Dyer, R. Liu, A. Nova, J. Xu, A. Walton, A. Parrish, M. Epstein, S. McCarthy, S. Petrov, D. Hassabis, K. Kavukcuoglu, J. Dean, and O. Vinyals. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*, 2024.

Gemma Team, M. Riviere, S. Pathak, P. G. Sessa, C. Hardin, S. Bhupatiraju, L. Hussenot, T. Mesnard, B. Shahriari, A. Ramé, J. Ferret, P. Liu, P. Tafti, A. Friesen, M. Casbon, S. Ramos, R. Kumar, C. L. Lan, S. Jerome, A. Tsitsulin, N. Vieillard, P. Stanczyk, S. Girgin, N. Momchev, M. Hoffman, S. Thakoor, J.-B. Grill, B. Neyshabur, O. Bachem, A. Walton, A. Severyn, A. Parrish, A. Ahmad, A. Hutchison, A. Abdagic, A. Carl, A. Shen, A. Brock, A. Coenen, A. Laforge, A. Paterson, B. Bastian, B. Piot, B. Wu, B. Royal, C. Chen, C. Kumar, C. Perry, C. Welty, C. A. Choquette-Choo, D. Sinopalnikov, D. Weinberger, D. Vijaykumar, D. Rogozińska, D. Herbison, E. Bandy, E. Wang, E. Noland, E. Moreira, E. Senter, E. Eltyshev, F. Visin, G. Rasskin, G. Wei, G. Cameron, G. Martins, H. Hashemi, H. Klimczak-Plucińska, H. Batra, H. Dhand, I. Nardini, J. Mein, J. Zhou, J. Svensson, J. Stanway, J. Chan, J. P. Zhou, J. Carrasqueira, J. Iljazi, J. Becker, J. Fernandez, J. van Amersfoort, J. Gordon, J. Lipschultz, J. Newlan, J. yeong Ji, K. Mohamed, K. Badola, K. Black, K. Millican, K. McDonell, K. Nguyen, K. Sodhia, K. Greene, L. L. Sjoesund, L. Usui, L. Sifre, L. Heuermann, L. Lago, L. McNealus, L. B. Soares, L. Kilpatrick, L. Dixon, L. Martins, M. Reid, M. Singh, M. Iverson, M. Görner, M. Velloso, M. Wirth, M. Davidow, M. Miller, M. Rahtz, M. Watson, M. Risdal, M. Kazemi, M. Moynihan, M. Zhang, M. Kahng, M. Park, M. Rahman, M. Khatwani, N. Dao, N. Bardoliwalla, N. Devanathan, N. Dumai, N. Chauhan, O. Wahltinez, P. Botarda, P. Barnes, P. Barham, P. Michel, P. Jin, P. Georgiev, P. Culliton, P. Kuppala, R. Comanescu, R. Merhej, R. Jana, R. A. Rokni, R. Agarwal, R. Mullins, S. Saadat, S. M. Carthy, S. Perrin, S. M. R. Arnold, S. Krause, S. Dai, S. Garg, S. Sheth, S. Ronstrom, S. Chan, T. Jordan, T. Yu, T. Eccles, T. Hennigan, T. Kocisky, T. Doshi, V. Jain, V. Yadav, V. Meshram, V. Dharmadhikari, W. Barkley, W. Wei, W. Ye, W. Han, W. Kwon, X. Xu, Z. Shen, Z. Gong, Z. Wei, V. Cotruta, P. Kirk, A. Rao, M. Giang, L. Peran, T. Warkentin, E. Collins, J. Barral, Z. Ghahramani, R. Hadsell, D. Sculley, J. Banks, A. Dragan, S. Petrov, O. Vinyals, J. Dean, D. Hassabis, K. Kavukcuoglu, C. Farabet, E. Buchatskaya, S. Borgeaud, N. Fiedel, A. Joulin, K. Kenealy, R. Dadashi, and A. Andreev. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

S. Hao, Y. Gu, H. Ma, J. J. Hong, Z. Wang, D. Z. Wang, and Z. Hu. Reasoning with language model is planning with world model. *arXiv preprint arXiv:2305.14992*, 2023.

D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

J. Huang, S. S. Gu, L. Hou, Y. Wu, X. Wang, H. Yu, and J. Han. Large language models can self-improve. *arXiv preprint arXiv:2210.11610*, 2022.

J. Kang, X. Z. Li, X. Chen, A. Kazemi, Q. Sun, B. Chen, D. Li, X. He, Q. He, F. Wen, J. Hao, and J. Yao. Mindstar: Enhancing math reasoning in pre-trained llms at inference time. *arXiv preprint arXiv:2405.16265*, 2024.

Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*, 2023.

H. Lightman, V. Kosaraju, Y. Burda, H. Edwards, B. Baker, T. Lee, J. Leike, J. Schulman, I. Sutskever, and K. Cobbe. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

H. Liu, Y. Zhang, Y. Luo, and A. C.-C. Yao. Augmenting math word problems via iterative question composing. *arXiv preprint arXiv:2401.09003*, 2024.

L. Luo, Z. Lin, Y. Liu, L. Shu, Y. Zhu, J. Shang, and L. Meng. Critique ability of large language models. *arXiv preprint arXiv:2310.04815*, 2023.

Q. Ma, H. Zhou, T. Liu, J. Yuan, P. Liu, Y. You, and H. Yang. Let's reward step by step: Step-level reward model as the navigators for reasoning. *arXiv preprint arXiv:2310.10080*, 2023.

OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. *arXiv preprint arXiv:2203.02155*, 2022.

E. Perez, D. Kiela, and K. Cho. True few-shot learning with language models. *arXiv preprint arXiv:2105.11447*, 2021.

C. D. Rosin. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, 2011.

D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016. URL https://doi.org/10.1038/nature16961.

D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Y. Tian, B. Peng, L. Song, L. Jin, D. Yu, H. Mi, and D. Yu. Toward self-improvement of llms via imagination, searching, and criticizing. *arXiv preprint arXiv:2404.12253*, 2024.

H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, D. Bikel, L. Blecher, C. C. Ferrer, M. Chen, G. Cucurull, D. Esiobu, J. Fernandes, J. Fu, W. Fu, B. Fuller, C. Gao, V. Goswami, N. Goyal, A. Hartshorn, S. Hosseini, R. Hou, H. Inan, M. Kardas, V. Kerkez, M. Khabsa, I. Kloumann, A. Korenev, P. S. Koura, M.-A. Lachaux, T. Lavril, J. Lee, D. Liskovich, Y. Lu, Y. Mao, X. Martinet, T. Mihaylov, P. Mishra, I. Molybog, Y. Nie, A. Poulton, J. Reizenstein, R. Rungta, K. Saladi, A. Schelten, R. Silva, E. M. Smith, R. Subramanian, X. E. Tan, B. Tang, R. Taylor, A. Williams, J. X. Kuan, P. Xu, Z. Yan, I. Zarov, Y. Zhang, A. Fan, M. Kambadur, S. Narang, A. Rodriguez, R. Stojnic, S. Edunov, and T. Scialom. LLaMA 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

J. Uesato, N. Kushman, R. Kumar, F. Song, N. Siegel, L. Wang, A. Creswell, G. Irving, and I. Higgins. Solving math word problems with process-and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

P. Wang, L. Li, Z. Shao, R. X. Xu, D. Dai, Y. Li, D. Chen, Y. Wu, and Z. Sui. Math-Shepherd: Verify and reinforce LLMs step-by-step without human annotations. *arXiv preprint arXiv:2312.08935*, 2024a.

X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of the 11th International Conference on Learning Representations (ICLR)*, May 2023.

Z. Wang, Y. Li, Y. Wu, L. Luo, L. Hou, H. Yu, and J. Shang. Multi-step problem solving through a verifier: An empirical analysis on model-induced process supervision. *arXiv preprint arXiv:2402.02658*, 2024b.

J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. Chi, T. Hashimoto, O. Vinyals, P. Liang, J. Dean, and W. Fedus. Emergent abilities of large language models, 2022a.

J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th Conference on Neural Information Processing Systems (NeurIPS)*, Dec 2022b.

S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023.

F. Yu, A. Gao, and B. Wang. Ovm, outcome-supervised value models for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*, 2024.

L. Yu, W. Jiang, H. Shi, J. Yu, Z. Liu, Y. Zhang, J. T. Kwok, Z. Li, A. Weller, and W. Liu. MetaMath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.

D. Zhang, S. Zhoubian, Z. Hu, Y. Yue, Y. Dong, and J. Tang. Rest-mcts*: Llm self-training via process reward guided tree search. *arXiv preprint arXiv:2406.03816*, 2024.

M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *arXiv preprint arXiv:2103.04931*, 2021.

Appendix

A. Question Filtering

During the evaluation of partial solution correctness using MC estimation, false negative noise may be introduced when a question is too hard for the model, thus no correct rollout can be found even with correct partial solution. Or false positive noise may be introduced when a question is too easy, that model can conclude in correct answer given partial solution with wrong step. It is not possible to exclude such noise completely, but we can reduce the chance by filtering out questions that are either too hard or too easy for the model. Specifically, we ran a $k = 32$ rollouts for each question in the 12K training data, and filter out the questions that with no correct answer (too hard) or no wrong answer (too easy) in the 32 rollouts.

B. Pairwise Loss Formula

When training with pairwise labels, the Bradley-Terry model (people typically use this objective to train reward models in RLHF) generally accepts two probability scalars summing up to 1. When we select the two actions as a pair, there are two cases. The first case is that one sample with a zero MC value, and the other sample with a positive MC value. The second case is that both samples are with positive MC values. The first case is straight-forward, and a normalization step is required for the second case.

Assume the two MC values are p and q, and they follow the Bernoulli distribution: $P(X = 1) = p$ and $P(Y = 1) = q$. We need to calculate the probability that action X is preferred over action Y and vice versa.

$$\begin{aligned} P(X > Y) &= P(X = 1, Y = 0) = p(1 - q), \\ P(X < Y) &= P(X = 0, Y = 1) = (1 - p)q, \\ P(X = Y) &= P(X = 0, Y = 0) + P(X = 1, Y = 1) = (1 - p)(1 - q) + pq. \end{aligned} \tag{5}$$

For the tied situation, each action has half the chance of being preferred. Thus,

$$\begin{aligned} P(\text{action X is preferred}) &= P(X > Y) + 1/2 * P(X = Y) = 1/2 * (1 + p - q), \\ P(\text{action Y is preferred}) &= P(X < Y) + 1/2 * P(X = Y) = 1/2 * (1 + q - p). \end{aligned} \tag{6}$$

Now the MC values are normalized and we can train with the pairwise loss.