Boosting Policy and Process Reward Models with Monte Carlo Tree Search in Open-Domain QA

Anonymous ACL submission

Abstract

The recent introduction of OpenAI's o1/o3 model represents a significant milestone in developing strong reasoning capabilities in Large Language Models (LLMs). By introducing more computational budget during test-time, LLMs have the potential to explore more accurate and higher-quality solutions. However, such paradigms are primarily verified in domains that have well-defined criteria for responses, such as coding and mathematics. Inspired by the success of this paradigm, we aim to bridge it to more subtle open-domain question answering. Specifically, we utilize search mechanisms such as Monte Carlo Tree Search (MCTS) for both policy model improvement and reward model improvement that achieve better performance in test-time scaling strategies. Our contributions are summarized in two folds: For the training phase, we demonstrate that our approach surpasses previous SOTA automatic data annotation methods and various public instruction-tuning datasets, with fewer data points. This offers a more data-efficient solution for training robust models. For the inference phase, we utilize the intermediate values collected during training data construction to train a process reward model called **PRM+**. This model employs a novel two-stage training method to provide finer-grained guidance across the generation trajectory. This introduces no additional overhead during training data collection and further enhances performance by scaling test-time computation. Experimental results show that our method can effectively improve the performance of both the policy model and the reward model.

1 Introduction

Large Language Models (LLMs) have made remarkable progress in recent years, demonstrating strong reasoning capabilities across a wide range of tasks (Guo et al., 2025; Team et al., 2025; Jaech et al., 2024; OpenAI et al., 2025). A key development in this area is the ability of LLMs to generate long Chains of Thought (CoT) (Wei et al., 2022). This enables them to iteratively refine their outputs and improve response quality (Madaan et al., 2024; Qi et al., 2024). The process can be viewed as a search over the space of possible reasoning trajectories, where increased computational resources during inference allow LLMs to explore more accurate and higher-quality solutions (Zeng et al., 2024).

In addition to sequential self-refinement of their own outputs, parallel decoding is another effective strategy for scaling test-time computation. Previous work suggests that by simply applying repeated sampling to the same question (Brown et al., 2024), models can achieve relatively high performance. Moreover, when combined with a well-learned value function (Chen et al., 2024; Feng et al., 2023) that provides fine-grained step-level supervision, LLMs can navigate higher-quality reasoning paths to solve complex problems.

Although the above-mentioned methods have already proven highly effective in structured domains such as mathematical reasoning (Wang et al., 2024a; Guan et al., 2025; Wang et al., 2024b; Muennighoff et al., 2025) and code generation (Zhang et al., 2024b; Chen et al., 2021; Jaech et al., 2024; Li et al., 2022) where there are well-defined ground truths, their applicability in open-domain tasks remains underexplored. Unlike structured problems, open-domain Question Answering (QA) lacks explicit correctness criteria, making it challenging to systematically guide the search process toward optimal responses.

To address this gap, our work explores whether search-based methods can enhance open-domain QA in both training and inference phases. We hypothesize that guiding LLMs through a structured search process can help identify high-quality responses, leading to more robust model training and better inference-time generation. Specifically, we propose leveraging Monte Carlo Tree Search

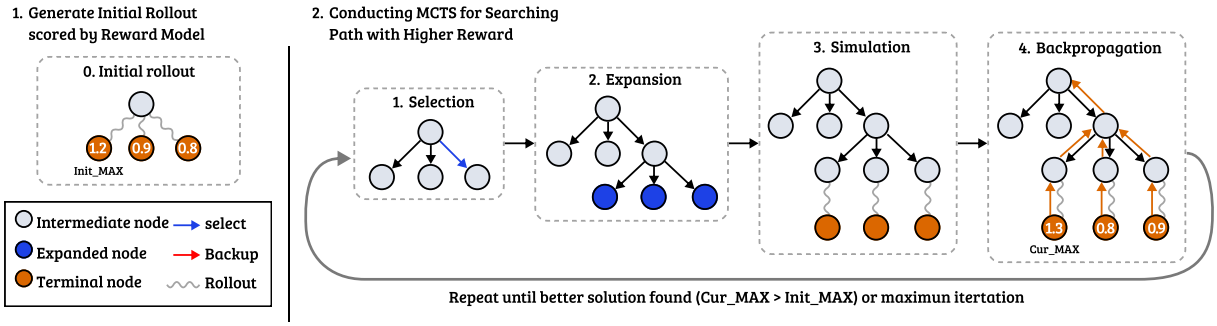


Figure 1: Our pipeline for utilizing MCTS to search for higher-quality responses.

(MCTS) (Silver et al., 2016, 2017) to explore multiple generation paths and select those that maximize a reward model's evaluation score. By treating generation as a search problem, we systematically improve the model's ability to generate informative and well-grounded answers. Our approach comprises two key phases: 1) the improvement of the policy model and 2) the improvement of the reward model.

In the training phase, we employ MCTS to collect high-quality trajectories by selecting the best-scoring generation paths during searching. These paths are then used to construct a Supervised Fine-Tuning (SFT) dataset. Additionally, we create a Direct Preference Optimization (DPO) dataset by pairing the highest and lowest-scoring responses. This allows the model to learn preference-based refinements directly from the data. Compared to previous automatic data annotation methods and publicly available instruction-tuning datasets (Ding et al., 2023; Xu et al., 2024; Lambert et al., 2024), our approach identifies superior response trajectories and achieves better downstream task performance using fewer data points. This leads to a more data-efficient training process overall.

During the inference phase, we further enhance the policy model's performance by introducing a Process Reward Model (PRM) to facilitate scalable test-time computation. Unlike conventional Outcome Reward Models (ORM), which evaluate only the final output, the PRM captures incremental improvements in generation quality and provides a more structured mechanism for refining responses at each step. Notably, collecting data to train the PRM does not introduce additional computational overhead, as the intermediate values used as supervision signals are by-products of the training data collection process. Furthermore, we propose a two-stage training method to develop an enhanced version of the PRM, referred to as **PRM+** in our paper. Experimental results demonstrate that our PRM+ outperforms various ORMs in scaled settings, validating the effectiveness of process-based supervision.

Our key contributions are as follows:

- We introduce a search-based data construction pipeline for open-domain QA, leveraging MCTS to systematically explore and select high-quality reasoning paths, which enhances both the policy model and the reward model.
- We demonstrate that our search-guided SFT and DPO data generation method outperforms existing annotation approaches and other public instruction-tuning datasets, while requiring fewer data points.
- We propose a two-stage training strategy to train our PRM+, which learns to provide process-level rewards, and demonstrate its effectiveness compared to ORM and other public PRM.

2 Related Work

Synthetic Data Generation The success of large language models heavily relies on high-quality data. Previous literature has demonstrated that higher-quality data can significantly enhance the performance of large language models (Zhou et al., 2023a; Ye et al., 2025). However, obtaining such data often depends on human annotation, which is costly and time-consuming. An alternative approach is synthetic data generation, which mimics real-world applications and retains high quality (Liu et al., 2024b). For example, leveraging language models to bootstrap and generate more instructions and responses has proven effective (Wang et al., 2023; Sun et al., 2023; Ding

et al., 2023). This success has been demonstrated across various domains, including coding (Luo et al., 2023b; Wei et al., 2024), mathematics (Luo et al., 2023a; Mitra et al., 2024), medical (Sun et al., 2024), and general domains (Xu et al., 2023; Taori et al., 2023). Our work falls into this category, utilizing search as a scalable and automatic method for synthetic data generation.

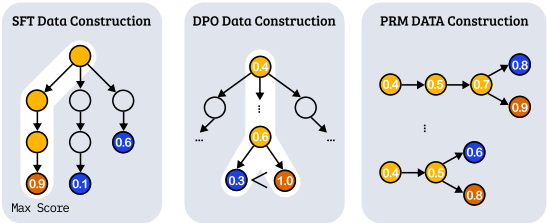


Figure 2: Illustration of data construction for training policy and reward model.

Inference-Time Scaling Recent advancements such as OpenAI's o1/o3 have sparked significant interest in scaling inference-time computation by generating more tokens before producing the final answer to enhance reasoning capabilities. This paradigm has inspired replication efforts within the community, including Deepseek R1 (Guo et al., 2025), Kimi 1.5 (Team et al., 2025), and others (Wang et al., 2024a; Zhang et al., 2024a). In contrast to sequential refinement strategies for scaling inference-time generation, another promising approach is parallel decoding (Snell et al., 2024; Wu et al., 2024). By generating multiple sequences for additional attempts or incorporating value functions to guide the search, these methods also demonstrate strong potential for improving performance through increased generation budgets. However, most approaches have been validated in domains with well-defined supervision signals, leaving their effectiveness in open-domain settings unverified.

Process Reward Model One of the key components for effectively scaling inference-time computation is the development of a reward model capable of evaluating the quality of generated candidates (Lee et al., 2023b; Ouyang et al., 2022). ORMs are widely adopted for this purpose. However, a primary challenge with ORMs is their coarse-grained feedback, which can lead to situations where, despite the final answer being correct, the solution paths may contain flaws. To address these limitations, process reward models have recently been developed. Currently, there are two primary annotation methods: 1) Human annotation, which requires manually rating intermediate steps of the solution paths, making it cost-intensive and time-consuming (Lightman et al., 2023; Uesato et al., 2022); and 2) Automatic annotation, which includes methods such as Monte Carlo estimation to construct stepwise supervision signals (Luo et al., 2024; Wang et al., 2024b), leveraging LLM-as-a-Judge to generate signals (Zhang et al., 2025), or deriving signals from outcome-based feedback (Yuan et al., 2024). Despite the significant progress and potential of these methods, they have primarily been validated in mathematical domains. In this work, we aim to explore the effectiveness of this paradigm in open-domain settings, which present more subtle optimization challenges.

3 Methodology

In this section, we first present the problem formulation, notation, and primary objective of our paper in Section 3.1. Following this, we detail our methodology for curating SFT datasets and DPO datasets to optimize the policy model in Section 3.2. Afterwards, we outline the data curation and training process for our PRM+, which is designed to explore inference-time scaling, in Section 3.3. We defer the details of the search algorithm employed in our work to Appendix A.

3.1 Problem Formulation

Following Hao et al. (2023), we formulate natural language generation as a Markov Decision Process (MDP), where an LLM acts as a policy generating responses step by step. The MDP consists of the following.

- **State Space** ($\mathcal{S}$): Each state s_t represents a partially generated sequence.
- **Action Space** ($\mathcal{A}$): The action a_t consists of generating a phrase, defined as a sequence of tokens that either (i) ends with the paragraph delimiter ("\n\n") or (ii) reaches the end of the response. Each phrase is sampled from the policy model π_θ (an LLM), such that:

$$a_t \sim \pi_\theta(s_t) \tag{1}$$

  where θ is the parameter of the policy model.
- **Transition Function** (T): The transition is deterministic, appending a_t to s_t.

$$s_{t+1} = T(s_t, a_t) = \text{Append}(s_t, a_t) \quad (2)$$

- **Reward Function** (R): An off-the-shelf reward model is used to assign a scalar score r_f to the final sequence, such that:

$$r_f = R(s_T) = R(\overbrace{[\overbrace{s_0, a_0}^{s_1}, a_1 \cdots a_T]}^{s_T}) \in \mathbb{R} \quad (3)$$

By formulating natural language generation as above, our objective can be summarized as two folds; 1) Optimizing the policy model π_θ that earns the maximum expected reward across the policy model training dataset $\mathcal{D}_{\text{policy}}$:

$$\underset{\theta}{\text{argmax}} \mathbb{E}_{x \sim \mathcal{D}_{\text{policy}}, \tau' \sim \pi_\theta(\tau|x)} r_f\left(\tau'\right) \quad (4)$$

where θ is the parameter of the policy model.

2) Optimizing the reward model (value function) V_ϕ that can accurately estimate the expected reward at state s_t across the reward model training dataset $\mathcal{D}_{\text{reward}}$. This can be written as follows.

$$\underset{\phi}{\text{argmin}} \mathbb{E}_{x \sim \mathcal{D}_{\text{reward}}, \tau' \sim \pi_\theta(\tau|x)} \left[\sum_{t=0}^{T} \left(V_\phi\left(\tau'_t\right) - Q\left(\tau'_t\right) \right)^2 \right] \quad (5)$$

where ϕ is the parameter of the reward model and $Q(\tau'_t)$ is the actual expected reward of the intermediate reward of trajectory τ at step t.

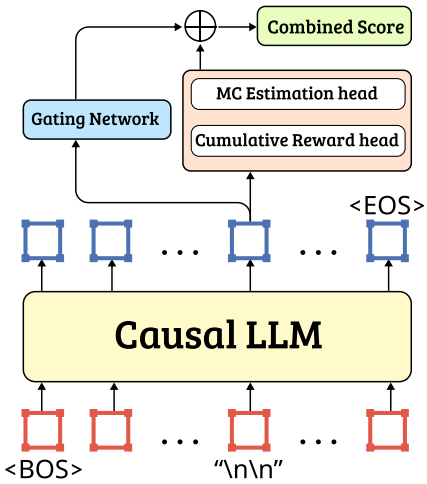


Figure 3: Architecture of our PRM+.

3.2 Policy Model Data Curation and Training

Our data curation method for enhancing the policy model's capabilities consists of two parts: 1)SFT data curation and 2) DPO data curation. As illustrated in Figure 1 and Figure 2, our full pipeline for data collection begins with generating a batch of initial rollouts given an instruction. Each rollout is then scored using an existing reward model. Following this, an MCTS process is carried out until a path with a reward greater than a threshold (By default, 1.3×Initial Maximal Score), or the maximum iteration limit is reached. Upon completing the search tree, we construct the SFT and DPO datasets. Further details of the collection process are provided in Algorithm 1.

3.3 Process Reward Model Data Curation and Training

In this section, we detail the process of collecting training data for the PRM and outline the training procedure. Previous approaches for automatic annotation of the process-level supervision signal have used MC estimation as the training signal (Wang et al., 2024b). In MC estimation, the reward is propagated backwards from the final reward, providing an estimate of the expected future reward. This reflects the likelihood of reaching a high-quality solution from a given state.

While using MC estimation as a training signal is a reasonable approach, we demonstrate in this paper that relying solely on this future reward estimation introduces noisy signals, consistent with the findings in Zhang et al. (2025). To mitigate this issue, we propose a complementary training signal, the cumulative reward, which offers a more stable and intuitive signal. The cumulative reward assigns a portion of the total reward to each step by evenly distributing the total reward across all steps. This ensures that the reward for each step reflects the cumulative progress made toward the final goal.

3.3.1 MC Estimation Synthetic

In order to construct the MC estimation dataset $D_{\text{PRM-MC}}$, we utilize the structure of the process at each step of the trajectory.Specifically, for each node that has children, we include the child with the highest and lowest MC estimation, calculated using Equation 11, along with its historical context into $D_{\text{PRM-MC}}$. Further details are provided in Algorithm 2.

	Base LLM = **Llama-3.1-8B-Base**	Generator	Quantity	**AlpacaEval 2** LC (%)	WR (%)	SD	**Arena-Hard** WR(%)	**Avg.** WR(%)
SFT	+Open Platypus (Lee et al., 2023a)	Mixed	25K	3.33	3.35	0.53	4.1	2.2
	+OpenHermes 2.5 (Teknium, 2023)	Mixed	1M	7.20	5.37	0.69	5.0	5.2
	+SlimOrca (Lian et al., 2023)	GPT-4	518K	4.60	3.66	0.57	2.4	3.0
	+UltraChat (Ding et al., 2023)	GPT-3.5-Turbo	208K	6.69	4.49	0.66	2.6	3.5
	+Tulu V3 Mix (Lambert et al., 2024) ↘	Mixed	940K	11.36	8.28	0.85	15.4	11.8
+ DPO	+Tulu V3 Mix-DPO	Mixed	273K	33.63	36.08	1.42	48.7	42.8
SFT	+Magpie-Air (Xu et al., 2024) ↘	Llama-3-8B-Instruct	300K	22.66	23.99	1.24	14.9	19.4
+ DPO	+Magpie-Air-DPO	Llama-3-8B-Instruct	100K	45.48	50.43	1.48	35.9	43.2
SFT	+Magpie-Pro (Xu et al., 2024) ↘	Llama-3-70B-Instruct	300K	25.08	29.47	1.35	18.9	24.2
+ DPO	+Magpie-Pro-DPO	Llama-3-70B-Instruct	100K	50.10	53.53	1.45	35.7	44.6
SFT	+OURS-Greedy	Llama-3.1-8B-Instruct	78K	19.53	17.81	1.18	18.1	18.0
	+OURS-BON	Llama-3.1-8B-Instruct	78K	21.93	20.49	1.21	22.9	21.7
	+OURS-MCTS ↘	Llama-3.1-8B-Instruct	78K	**23.05**	**25.03**	1.28	**25.9**	**25.5**
+ DPO	+OURS-DPO	Llama-3.1-8B-Instruct	55K	**49.70**	**54.52**	1.45	**40.4**	**47.5**
Llama-3.1-8B-Instruct (SFT+DPO)		-	>10M	22.92	22.57	1.26	20.6	21.6
Llama-3.1-70B-Instruct (SFT+DPO)		-	>10M	38.10	39.10	1.39	55.7	47.4

Table 1: The table compares the policy model based on Llama-3.1-8B-base trained on our datasets against baseline datasets. **Mixed** indicates that the dataset is annotated by more than one LLM (including human annotation). Numbers in **Bold** signify that they surpass the previous state-of-the-art (SOTA) results shown in Underline, which were achieved by a model of comparable size. Furthermore, by utilizing significantly fewer data points, our model outperforms the officially aligned model Llama-3.1-8B-Instruct and even achieves results comparable to the much larger Llama-3.1-70B-Instruct.

3.3.2 Cumulative Reward Synthetic

For the construction of the cumulative reward dataset $\mathcal{D}_{\text{PRM-CR}}$, we aim to assign a portion of the total reward to each step in the trajectory. Formally, we define the cumulative reward at step t in a trajectory as:

$$r_t^{\text{cumulative}} = \frac{r_{\text{f}}}{T} \tag{6}$$

where T represents the total number of steps in the trajectory, and t denotes the current step. This reward is uniformly assigned to each step. It is worth noting that other potential distribution methods exist, and we aim to explore these further in future work.

3.3.3 Synergy of Two Intermediate Signals

For each response, both values are predicted at each step. However, these dual-dimensional outputs need to be aggregated into a single scalar for comparing test examples. A straightforward approach is to manually design coefficients for the two values. However, this method is rigid and lacks generalization. In our paper, we instead train a gating network to synergize the two intermediate signals, inspired by the Mixture of Experts (MoE) architecture (Guo et al., 2025). The gating network takes as input the features extracted from the model π_θ and outputs a set of gating coefficients, which are then multiplied by the respective reward signals. This results in a composite reward, which is further used to calculate the final scalar score R_{PRM} for a given response. The combined reward is represented as:

$$R_{\text{PRM}} = \sum_{t=0}^{T} \sum_{i=1}^{2} g_{\Psi,i}\left(\pi_\theta(x_t)\right) \cdot r'_{i,t} \tag{7}$$

where $r'_{i,t}$ is the intermediate signal in step t, and g_ψ denotes the gating function that determines the weighting of each signal.

In this stage, we optimize our system using the Bradley-Terry loss. This loss function is well-suited for pairwise comparisons and allows the model to learn the optimal combination of the two reward signals. The objective is to minimize the following:

$$\min_{\phi,\psi} \mathbb{E}\left[-\log \sigma\left(R_{\text{PRM_Chosen}} - R_{\text{PRM_Rejected}}\right)\right] \tag{8}$$

where $R_{\text{PRM_Chosen}}$ and $R_{\text{PRM_Rejected}}$ are the preference scores for the chosen and rejected responses in each pairwise comparison. This training procedure enables the model to effectively combine the two reward signals and optimize for distinguishing the most superior response.

4 Experiments

4.1 Policy Model Training Setup

Data Curation for Policy Model During the MCTS process, we employ LLaMA-3.1-8B-Instruct as the policy model to generate responses

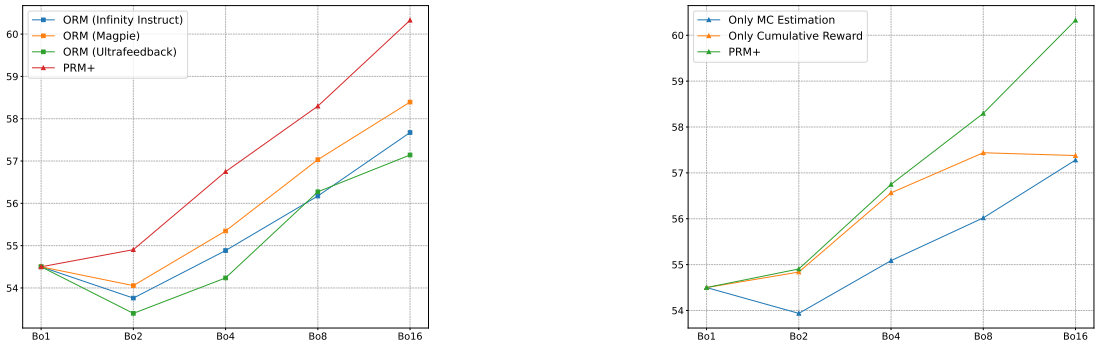


Figure 4: Scaling inference-time experiments based on ours-Llama-3.1-8B. **Left:** Comparison of PRM+ with existing ORM. **Right:** Comparison of PRM+ with PRM (only MC estimation) and PRM (only cumulative reward).

and utilize Skywork-Gemma-2-27B (Liu et al., 2024a) as the reward model to score the final terminated response. We set the default parameters as follows: temperature at 1.0, top-k at 30, branch factor at 16, and a maximum iteration limit of 20 during the search. For the SFT dataset construction, we randomly sample instructions from Infinity Instruct (BAAI, 2024) to form the initial pool. It is important to note that we do not use the original answers provided in the dataset. Detailed information on data statistics can be found in Appendix B. For the DPO dataset construction, we randomly sample instructions from Magpie-Pro (Xu et al., 2024) and follow Algorithm 1 to create the paired data. In total, this process results in 78K data points for SFT and 55K data points for DPO.

Training Details for Policy Model Our default training hyperparameters is as follows: for SFT, we set the learning rate as 2e-5; for DPO, we set the learning rate as 1e-6 and β as 0.01. Details for the implementation are shown in Appendix C.

Policy Evaluation We primarily evaluate the performance of our trained models using two widely adopted instruction-following benchmarks: AlpacaEval2 and Arena-Hard. The main metric for these benchmarks is the win rate (WR), which calculates the fraction of responses favored by the GPT evaluator. Additionally, AlpacaEval2 employs a Length-Controlled win rate (LC) to mitigate the influence of response length. Furthermore, we compare our model's performance on a broader range of benchmarks, including those focused on code and math. For detailed descriptions of the benchmarks and additional experimental results, please refer to Appendix E and Appendix D.

Baseline Comparison It is intuitive to compare models trained on our dataset with those trained on open-source instruction datasets. Competitors include **Open Platypus** (Lee et al., 2023a), **OpenHermes 2.5** (Teknium, 2023), **SlimOrca** (Lian et al., 2023) and **Tulu V3 Mix** (Lambert et al., 2024). Additionally, since our dataset is generated by searching over responses produced by Llama-3.1-8B-Instruct, we also include comparisons with two strong baselines: 1) model trained on the greedy decoding response, referred to as **Ours-Greedy**, and 2) model trained on the response with the highest reward from the initial rollout (as described in Algorithm 1), which we call **OURS-BoN**. For preference optimization, we compare our results with various open-source paired instruction datasets, including **Magpie-Air**, **Magpie-Pro** (Xu et al., 2024), and **Tulu V3 Mix-DPO** (Lambert et al., 2024).

4.2 Process Reward Model Training Setup

Experimental Details For training the process reward model, we employ a two-stage training method as outlined in Section 3.3. In the first stage, we use the same proportion of Infinity Instruct data that was utilized for SFT of the model. For each instruction in this dataset, we collect training data that includes step-wise responses along with their corresponding MC estimation and cumulative reward, as detailed in Algorithm 2. To model the dual intermediate values, we attach a linear regression layer $\omega \in R^{d\times 2}$ on top of a pretrained LLM. This layer takes as input a d-dimensional feature at the position of each delimiter ("\n\n") for different steps and uses the mean squared error (MSE) loss to fit both values. The learning rate is set to 2e-5 during this stage. For the second stage of training, we attach a shallow Multi-Layer Perceptron

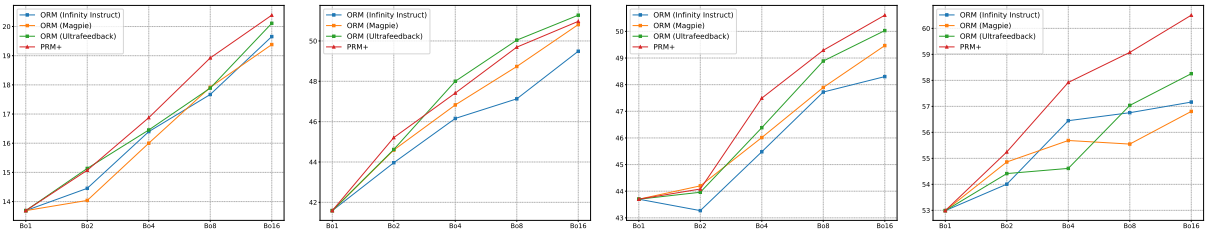


Figure 5: Inference-time scaling experiments based on different policy models. **Left** for Llama-3.2-1B. **Left Middle** for Llama-3.2-3B. **Right Middle** for Qwen-2.5-3B. **Right** for Qwen-2.5-7B.The results demonstrated the generalizability of our PRM+.

(MLP) with three fully-connected layers on top of the same pretrained LLM. This MLP takes as input a d-dimensional feature at the position of each delimiter ("\n\n") for different steps and outputs the coefficients as shown in Equation 7. During this stage, we use UltraFeedback (Cui et al., 2023) as the training dataset, with a learning rate of 2e-5 and train for one epoch. A brief illustration of the implementation architecture is provided in Figure 3. By default, we use Qwen-2.5-7B-Instruct as the backbone model.

Base LLM	AlpacaEval 2			Arena-Hard
	LC (%)	WR (%)	SD	WR(%)
Llama-3.2-3B-Base	32.10	41.59	1.44	22.2
Llama-3.2-1B-Base	8.26	13.69	0.99	5.5
Llama-3.2-3B-instruct	19.75	20.46	1.20	14.9
Llama-3.2-1B-instruct	8.18	9.34	0.88	5.1
Qwen-2.5-7B-Base	48.69	52.99	1.45	45.7
Qwen-2.5-3B-Base	38.31	43.70	1.46	33.7
Qwen-2.5-7B-instruct	29.74	30.06	1.37	52.0
Qwen-2.5-3B-instruct	18.15	19.97	1.20	27.3

Table 2: The table demonstrates that our curated datasets also achieve significant improvements across models of different sizes and series, outperforming their officially instruction-tuned version.

PRM Evaluation We primarily evaluate our PRM+ in two scaling scenarios: 1) **Best-of-N (BoN) Verification** and 2) **Guided Decoding**. For the BoN scenario, we set the temperature to 0.7 and top-k to 30 to generate diverse responses. We then use either a PRM or an ORM to assign a score to each response, selecting the highest-scoring response as the final answer. In this setting, we compare our trained PRM+ with ORM trained on various datasets, including Infinity Instruct, Magpie, and UltraFeedback. For the guided decoding scenario, we adopt a beam search decoding approach. At each step, we generate $Beam_{width}$ candidate partial responses and use a PRM to score these candidates, retaining the top-k candidates for further generation. Since ORM cannot score partial responses, we compare our PRM+ with the open-source Skywork-o1-PRM[1] and vanilla beam search, where the score of each candidate is determined by prior probability rather than a reward model.

4.3 Policy Model Results

Models Trained on Our Curated Datasets Show Superior Performance In Table 1, we compare the performance of the LLaMA-3.1-8B Base model trained on our dataset and on other public instruction tuning datasets. The results show that our method shows drastic improvement compared with other datasets in both AlpacaEval2 and Arena-Hard benchmarks by utilizing far fewer data points, showing the superiority of our curated dataset. Another highlight is that our dataset is generated by a relatively small size model LLaMA-3.1-8B-Instruct, set apart it from other instruction tuning data like Tulu V3 Mix that gather instances that come from much larger generation models such as gpt4 or even human-annotated responses. Additionally, with DPO techniques, our models' capability can be further enhanced, and even surpass the instruction versions of Llama-3.1-8B and Llama-3.1-70B that are trained by using millions of instruction tuning datasets. We also show additional results on more benchmark in Appendix D demonstrating that our method does not only surpass baseline models on subjective evaluation but also on various tasks such as expertise knowledge, mathematics, and code domain.

MCTS Searched Response is of Higher Quality than BoN and Greedy Another observation is that responses generated through MCTS-based searching are of higher quality compared to those

[1] https://huggingface.co/Skywork/Skywork-o1-Open-PRM-Qwen-2.5-7B

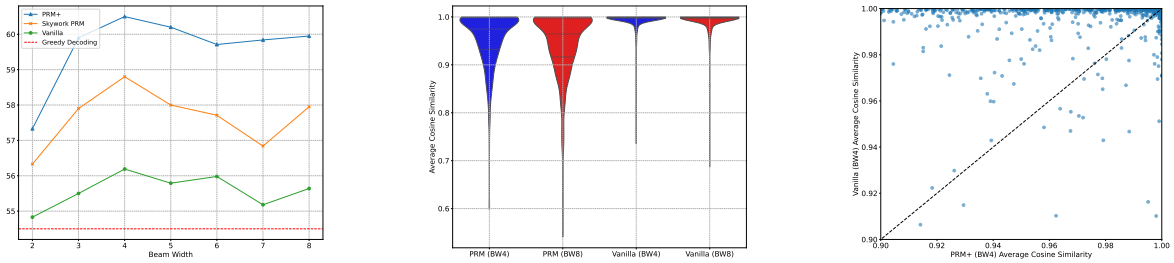


Figure 6: Beam search experiments. **Left:** Comparison of ours PRM+ with public PRM and vanilla beam search. **Middle:** Distribution of average cosine similarity across all instances. **Right:** Instances-wise average cosine similarity (PRM+ (BW4) vs Vanilla (BW4)) .

from BoN and greedy decoding. This is evidenced by the second row of Table 1, which shows that models trained on MCTS-searched responses achieve superior performance. Additionally, further analysis in Appendix B suggests that MCTS-searched responses exhibit 1) higher quality and 2) greater diversity.

Our Curated Dataset Shows Positive Effect on Different Models Table 2 demonstrates that our dataset also exhibits significant effectiveness when applied to models of different sizes and families. For instance, on AlpacaEval2, when trained on Llama-3.2-3B, it achieves a 12% absolute improvement compared to its instruction-version counterpart, while training on Qwen-2.5-7B results in an 18% absolute improvement over its instruction-version counterpart.

4.4 Inference-Time Scaling

In this section, we move forward to test the model's inference-time scaling ability under two scenario as introduced in Section 4.2. We summarize our findings as follows.

Best-of-N Verification As illustrated in the left panel of Figure 4, we increase the generation budget from 1 (2^0) to 16 (2^4) and use both PRM and ORM to score the candidates. We demonstrate that our PRM+ shows a clear scaling trend as the generation budget increases, while ORM struggles to exhibit a similar trend. Additionally, compared to ORMs trained on various datasets, our PRM+ consistently outperforms them across different budgets, highlighting the effectiveness of PRM+. Moreover, we show the necessity of incorporating both the MC estimation head and the cumulative reward head in our model. The right panel of Figure 4 indicates that neither single head's prediction can surpass PRM+. This suggests that combining both heads is essential for achieving optimal performance, thereby validating the effectiveness of our architectural design. The combination of PRM+ verification results across different models is depicted in Figure 5. We observe a similar trend, with PRM+ consistently demonstrating superior performance.

Guided Decoding In Figure 6, we compare the scaling effects of our PRM+, the publicly available Skywork-o1-PRM, and the vanilla beam decoding strategy. Our results show that domain-specific PRMs, such as Skywork-o1-PRM, underperform compared to our PRM+ due to the domain gap. This highlights the necessity of developing effective PRMs for open-domain tasks, which are rarely addressed in the current literature. Another key finding is that vanilla beam search, which relies on prior probability to select candidates, fails to exhibit scaling performance as the beam width increases. We attribute this to the lack of diversity among the selected candidates. This is supported by the middle and right panels of Figure 6, where we plot the average cosine similarity and per-instance cosine similarity, respectively. The results indicate that vanilla beam search leads to more homogeneous candidates, thereby reducing performance in scaled settings. This aligns with the findings in Chen et al. (2024).

5 Conclusion

In this paper, we introduce a search-based method to enhance both policy and reward models for open-domain QA. Our approach achieves data-efficient training, outperforming previous SOTA data synthetic methods while requiring fewer data points. Additionally, we propose a two-stage training strategy to build **PRM+** which demonstrates a superior scaling trend as computation increases during inference.

6 Limitations

Although we conduct extensive experiments to demonstrate the effectiveness of search-based methods in enhancing both policy models and reward models, our work has several limitations. First, due to the additional computational burden introduced by the search process, we do not scale our construction to a massive dataset. Even though our method outperforms various counterparts that use significantly more data points, there remains a trade-off between quality and quantity that needs to be explored in future work. Second, our approach lacks interpretability in intermediate annotations. While our work pioneers a novel method for improving the training of process reward models, resulting in the PRM+ presented in this paper (especially in the open-domain scenario), the automatically generated annotations, similar to those in previous literature (Wang et al., 2024b), are inevitably noisy and lack interpretability. Future work could focus on incorporating human annotations with automatic annotations to address this issue.

References

Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program synthesis with large language models. CoRR, abs/2108.07732.

BAAI. 2024. Infinity instruct. arXiv preprint arXiv:2406.XXXX.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. arXiv preprint arXiv:2407.21787.

Guoxin Chen, Minpeng Liao, Chengxi Li, and Kai Fan. 2024. Alphamath almost zero: process supervision without process. arXiv preprint arXiv:2405.03553.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. arXiv preprint arXiv:2110.14168.

Ganqu Cui, Lifan Yuan, Ning Ding, Guanming Yao, Wei Zhu, Yuan Ni, Guotong Xie, Zhiyuan Liu, and Maosong Sun. 2023. Ultrafeedback: Boosting language models with high-quality feedback. arXiv preprint arXiv:2310.01377.

Ning Ding, Yulin Chen, Bokai Xu, Yujia Qin, Zhi Zheng, Shengding Hu, Zhiyuan Liu, Maosong Sun, and Bowen Zhou. 2023. Enhancing chat language models by scaling high-quality instructional conversations. arXiv preprint arXiv:2305.14233.

Robert Dorfman. 1979. A formula for the gini coefficient. The review of economics and statistics, pages 146–149.

Xidong Feng, Ziyu Wan, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2023. Alphazero-like tree-search can guide large language model decoding and training. arXiv preprint arXiv:2309.17179.

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948.

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

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the math dataset. arXiv preprint arXiv:2103.03874.

Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. 2024. Openai o1 system card. arXiv preprint arXiv:2412.16720.

Levente Kocsis, Csaba Szepesvári, and Jan Willemson. 2006. Improved monte-carlo search. Univ. Tartu, Estonia, Tech. Rep, 1:1–22.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri,

Shane Lyu, et al. 2024. T\" ulu 3: Pushing frontiers in open language model post-training. arXiv preprint arXiv:2411.15124.

Ariel N Lee, Cole J Hunter, and Nataniel Ruiz. 2023a. Platypus: Quick, cheap, and powerful refinement of llms. arXiv preprint arXiv:2308.07317.

Harrison Lee, Samrat Phatale, Hassan Mansoor, Kellie Ren Lu, Thomas Mesnard, Johan Ferret, Colton Bishop, Ethan Hall, Victor Carbune, and Abhinav Rastogi. 2023b. Rlaif: Scaling reinforcement learning from human feedback with ai feedback. https://openreview.net/pdf?id=AAxIs3D2ZZ.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. 2024. From live data to high-quality benchmarks: The arena-hard pipeline.

Xuechen Li, Tianyi Zhang, Yann Dubois, Rohan Taori, Ishaan Gulrajani, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Alpacaeval: An automatic evaluator of instruction-following models. https://github.com/tatsu-lab/alpaca_eval.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022. Competition-level code generation with alphacode. Science, 378(6624):1092–1097.

Wing Lian, Guan Wang, Bleys Goodson, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Slimorca: An open dataset of gpt-4 augmented flan reasoning traces, with verification.

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

Chris Yuhao Liu, Liang Zeng, Jiacai Liu, Rui Yan, Jujie He, Chaojie Wang, Shuicheng Yan, Yang Liu, and Yahui Zhou. 2024a. Skywork-reward: Bag of tricks for reward modeling in llms. arXiv preprint arXiv:2410.18451.

Ruibo Liu, Jerry Wei, Fangyu Liu, Chenglei Si, Yanzhe Zhang, Jinmeng Rao, Steven Zheng, Daiyi Peng, Diyi Yang, Denny Zhou, et al. 2024b. Best practices and lessons learned on synthetic data for language models. arXiv preprint arXiv:2404.07503.

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

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

Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. 2023b. Wizardcoder: Empowering code large language models with evol-instruct. arXiv preprint arXiv:2306.08568.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2024. Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Systems, 36.

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

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. 2025. s1: Simple test-time scaling. arXiv preprint arXiv:2501.19393.

OpenAI, :, Ahmed El-Kishky, Alexander Wei, Andre Saraiva, Borys Minaev, Daniel Selsam, David Dohan, Francis Song, Hunter Lightman, Ignasi Clavera, Jakub Pachocki, Jerry Tworek, Lorenz Kuhn, Lukasz Kaiser, Mark Chen, Max Schwarzer, Mostafa Rohaninejad, Nat McAleese, o3 contributors, Oleg Mürk, Rhythm Garg, Rui Shu, Szymon Sidor, Vineet Kosaraju, and Wenda Zhou. 2025. Competitive programming with large reasoning models. Preprint, arXiv:2502.06807.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. Advances in neural information processing systems, 35:27730–27744.

Zhenting Qi, Mingyuan Ma, Jiahang Xu, Li Lyna Zhang, Fan Yang, and Mao Yang. 2024. Mutual reasoning makes smaller llms stronger problem-solvers. arXiv preprint arXiv:2408.06195.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pages 3505–3506.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2024. GPQA: A graduate-level google-proof q&a benchmark. In First Conference on Language Modeling.

Claude Elwood Shannon. 1948. A mathematical theory of communication. The Bell system technical journal, 27(3):379–423.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. nature, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. arXiv preprint arXiv:1712.01815.

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

Yuxuan Sun, Yunlong Zhang, Yixuan Si, Chenglu Zhu, Zhongyi Shui, Kai Zhang, Jingxiong Li, Xingheng Lyu, Tao Lin, and Lin Yang. 2024. Pathgen-1.6 m: 1.6 million pathology image-text pairs generation through multi-agent collaboration. arXiv preprint arXiv:2407.00203.

Zhiqing Sun, Yikang Shen, Qinhong Zhou, Hongxin Zhang, Zhenfang Chen, David Cox, Yiming Yang, and Chuang Gan. 2023. Principle-driven self-alignment of language models from scratch with minimal human supervision. Advances in Neural Information Processing Systems, 36.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B Hashimoto. 2023. Stanford alpaca: An instruction-following llama model.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. arXiv preprint arXiv:2501.12599.

Teknium. 2023. Openhermes 2.5: An open dataset of synthetic data for generalist llm assistants.

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

Jun Wang, Meng Fang, Ziyu Wan, Muning Wen, Jiachen Zhu, Anjie Liu, Ziqin Gong, Yan Song, Lei Chen, Lionel M Ni, et al. 2024a. Openr: An open source framework for advanced reasoning with large language models. arXiv preprint arXiv:2410.09671.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024b. 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.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. Self-instruct: Aligning language models with self-generated instructions. In Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 13484–13508, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in neural information processing systems, 35:24824–24837.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2024. Magicoder: Empowering code generation with oss-instruct. In Forty-first International Conference on Machine Learning.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-art natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pages 38–45, Online. Association for Computational Linguistics.

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

Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. 2023. Wizardlm: Empowering large language models to follow complex instructions. arXiv preprint arXiv:2304.12244.

Zhangchen Xu, Fengqing Jiang, Luyao Niu, Yuntian Deng, Radha Poovendran, Yejin Choi, and Bill Yuchen Lin. 2024. Magpie: Alignment data synthesis from scratch by prompting aligned llms with nothing. arXiv preprint arXiv:2406.08464.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. arXiv preprint arXiv:2502.03387.

Lifan Yuan, Wendi Li, Huayu Chen, Ganqu Cui, Ning Ding, Kaiyan Zhang, Bowen Zhou, Zhiyuan Liu, and Hao Peng. 2024. Free process rewards without process labels. arXiv preprint arXiv:2412.01981.

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

Di Zhang, Jianbo Wu, Jingdi Lei, Tong Che, Jiatong Li, Tong Xie, Xiaoshui Huang, Shufei Zhang, Marco Pavone, Yuqiang Li, et al. 2024a. Llama-berry: Pairwise optimization for o1-like olympiad-level mathematical reasoning. arXiv preprint arXiv:2410.02884.

Yuxiang Zhang, Shangxi Wu, Yuqi Yang, Jiangming Shu, Jinlin Xiao, Chao Kong, and Jitao Sang. 2024b. o1-coder: an o1 replication for coding. arXiv preprint arXiv:2412.00154.

Zhenru Zhang, Chujie Zheng, Yangzhen Wu, Beichen Zhang, Runji Lin, Bowen Yu, Dayiheng Liu, Jingren Zhou, and Junyang Lin. 2025. The lessons of developing process reward models in mathematical reasoning. arXiv preprint arXiv:2501.07301.

Chunting Zhou, Pengfei Liu, Puxin Xu, Srinivasan Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, et al. 2023a. Lima: Less is more for alignment. Advances in Neural Information Processing Systems, 36.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023b. Instruction-following evaluation for large language models. arXiv preprint arXiv:2311.07911.

A Search Algorithm: Monte Carlo Tree Search

To refine response generation, we apply Monte Carlo Tree Search (MCTS), iteratively improving candidate solutions. MCTS consists of:

Selection At each step, we select the action that maximizes the UCT score (Kocsis et al., 2006):

$$U(s,a) = Q(s,a) + c\sqrt{\frac{\ln N(s)}{N(s,a)}} \tag{9}$$

where $Q(s,a)$ is the estimated reward for taking action a in state s. $N(s)$ is the total count of visits to state s. $N(s,a)$ is the count of visits for action a at state s. c is an exploration constant.

Expansion If an action a has not been explored, we expand a new node s' by sampling the next action using the policy model π_θ, where the current state s serves as input to the policy. The policy model then generates the next action a, which is terminated when the sequence reaches a paragraph delimiter ('\n\n') or the end-of-sequence token '<eos>'.

Simulation From the newly expanded state s', the policy model generates the complete response by continuing to the sample actions. The process continues until the model generates the '<eos>' token, indicating the end of the sequence. The complete sequence's reward $Q(s_T) = Q(\mathcal{T}(s_{\text{T-1}}, a_{\text{T-1}})) = r_f$ is then scored using the Equation 3.

Backpropagation Once the simulation is completed and a final reward is obtained, the estimated reward $Q(s,a)$ and the visit count $N(s,a)$ for the current state action pair are updated with the reward from the simulated sequence. Afterwards, an update will propagated upward through the tree utill reaching the root node as follows.

$$N(s,a) \leftarrow N(s,a) + 1 \tag{10}$$

$$Q(s,a) = \frac{1}{|\mathcal{C}(s,a)|} \sum_{s' \in \mathcal{C}(s,a)} Q(s',a') \tag{11}$$

where $\mathcal{C}(s,a)$ represents the set of child nodes of node s.

B Dataset Statistics

B.1 Category Analysis

In this section, we present statistics for the curated datasets used in this study. Our dataset is designed to cover a wide range of cognitive and technical abilities, comprising a comprehensive collection of data points that reflect various categories, including problem-solving, logical reasoning, programming ability, and more. The overall distribution of these categories is shown in Figure 7a.

While the coarse categories provide a useful high-level perspective, the fine-grained ability analysis offers a deeper dive into specific skills and knowledge areas. This detailed analysis enables a more nuanced understanding of individual abilities and their contributions to the broader categories. For instance, within the Problem Solving category, the fine-grained analysis reveals distinct abilities such as Logical Reasoning (13.8%) and Analytical Reasoning (1.46%), each contributing uniquely to the overall problem-solving capability. The fine-grained ability analysis is depicted in Figure 7b.

B.2 Quality Analysis

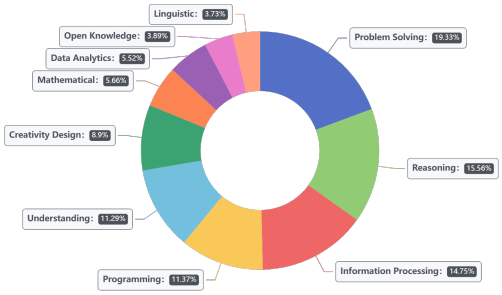
In this section, we evaluate the quality of our generated dataset and compare it with the baseline datasets introduced in Section 4.1. Specifically, we assess the performance of three variants: **Ours-MCTS**, **Ours-BoN**, and **Ours-Greedy**. The reward scores are computed using Skywork-Gemma-2-27B. The results are shown in Figure 8.

The reward distributions for all three datasets follow a normal distribution, with the majority of instances receiving mid-range rewards. The reward values are generally scattered between -10 and 20. An obvious distribution shift is observed, indicating that the quality of the datasets follows the order: MCTS > BoN > Greedy, as expected.

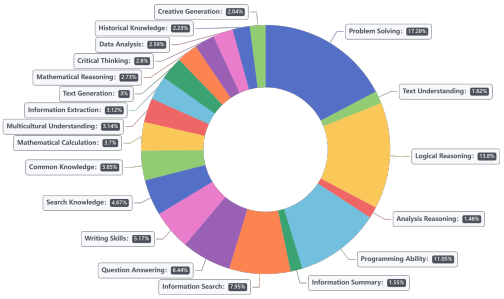
B.3 Diversity Analysis

In this section, we analyze the diversity of our three datasets using 2-gram statistics as our primary tool. Specifically, we sample 5,000 instances from each dataset and compute their 2-gram statistics. The absolute and relative frequencies are plotted in Figure 9.

We observe that the MCTS dataset has the most uniform 2-gram distribution, followed by BoN and then Greedy. A more uniform distribution implies greater diversity, as answers are less concentrated



(a) Category analysis of our dataset.



(b) Ability analysis of our dataset.

in specific patterns, resulting in a longer tail distribution.

Additionally, we conduct quantitative analysis by calculating the entropy (Shannon, 1948) and Gini coefficient (Dorfman, 1979), as shown in Equations 12 and 13, respectively.

- **Entropy**: Higher entropy indicates a more uniform distribution, meaning probabilities are spread more evenly across different categories (e.g., n-grams in responses), suggesting greater diversity.

- **Gini Coefficient**: Lower Gini coefficient indicates a more equal distribution, meaning no single category dominates, also suggesting greater diversity.

Thus, a combination of higher entropy and lower Gini coefficient indicates a more diverse distribution. The results are summarized in Table 3, showing that the diversity ranking is MCTS > BoN > Greedy.

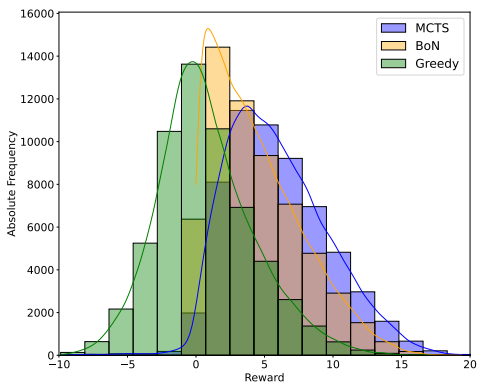


Figure 8: Reward distribution of three SFT datasets.

$$Entropy = -\sum_{i=1}^{N} p_i \log_2 p_i \tag{12}$$

$$Gini = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n}|p_i - p_j|}{2n\sum_{i=1}^{n} p_i} \tag{13}$$

where p_i is the frequency of the i_{th} 2-gram and n is the total number of the 2-gram.

Dataset Config	Entropy (↑)	Gini (↓)
MCTS	**0.19**	**0.28**
BoN	0.17	0.29
Greedy	0.17	0.32

Table 3: Entropy and Gini of the three datasets.

C More Implementation Details

We implement our policy model training using the default trainer from Hugging Face's `Transformers` toolkit (Wolf et al., 2020). For reward model training, we use the `RewardTrainer` and load the model from `AutoCausalLMwithValueHEAD` in Hugging Face's `trl` toolkit (Wolf et al., 2020). We modify the code to add an additional value head and a gating network to support training our **PRM+**. All models are trained on an 8×80GB NVIDIA H800 server. We employ full-parameter fine-tuning with the DeepSpeed ZeRO-2 configuration (Rasley et al., 2020) to optimize GPU memory utilization. A cosine learning rate scheduler is enabled, along with a default warmup period of 0.01 of the total training steps. All instances are truncated to a maximum length of 2048 tokens. For training, we set the per-device batch size to 2 and the gradient

accumulation steps to 4, resulting in a total batch size of $2 \times 4 \times 8$ (devices) $= 64$. During inference, we utilize vLLM (Kwon et al., 2023) to accelerate text generation.

D Additional Experiment Results

In this section, we present the additional performance of models fine-tuned on our dataset, as well as various instruction tuning datasets based on Llama-3.1-Base. The additional results cover a wide range of capabilities, including expertise knowledge, mathematics, and coding. The results are shown in Table 4. Notably, our model performs well across these benchmarks, surpassing OpenHermes 2.5, which contains 1 million data points—nearly 13 times the size of our dataset. Additionally, our model outperforms the Mapie series in several tasks. These results demonstrate the effectiveness, generalizability, and adaptability of our dataset curation method.

E Details About Evaluations

Below is a brief introduction to each task.

- AlpacaEval 2 (Li et al., 2023) is an evaluation system for LLMs that includes 805 representative instructions derived from real user interactions. It features a leaderboard that uses GPT-4-1106-preview as a judge to automatically evaluate and compare model responses.

- Arena-Hard (Li et al., 2024) is a high-quality benchmark, consisting of 500 challenging prompts, designed for evaluating LLMs. It has key features that robustly differentiate model capabilities and reflect human preferences in real-world use cases.

- GPQA (Rein et al., 2024) is a dataset of 448 difficult multiple-choice questions in biology, physics, and chemistry, created by experts. PhD-level experts have 65% accuracy, while skilled non-experts score 34%, even with web access.

- IFEval (Zhou et al., 2023b) offers 541 verified instructions tailored for code-based evaluation. These instructions encompass 25 distinct verifiable types, featuring tasks such as Keyword Frequency and Word Count.

- GSM8K (Cobbe et al., 2021) provides 8,790 (Train: 7,470; Test: 1,320) high-quality, linguistically diverse grade school math word problems to diagnose the shortcomings of current language models in multi-step mathematical reasoning.

- Math (Hendrycks et al., 2021) consists 12,500 (Train: 7500; Test: 5000) tough competition math problems. A subject's problems can vary in difficulty levels, ranging from '1' to '5'. Each problem includes a step-by-step solution.

- HumanEval (Chen et al., 2021) is a dataset designed to evaluate the code generation abilities of LLMs. Its purpose is to assess the functional accuracy of programs generated from docstrings. The dataset includes 164 unique programming challenges that test language understanding, algorithms, and fundamental mathematics.

- MBPP (Austin et al., 2021) is composed of 974 Python programming challenges sourced from the community, crafted to be achievable by beginner programmers and covering essentials such as programming fundamentals and standard library functions. Each challenge includes a task description, a code solution, and three automated test cases.

Base LLM = Llama-3.1-8B-Base	GPQA	IFEval	GSM8K	Math	HumanEval+	MBPP+	Avg.
Open Platypus	27.2	45.0	35.2	15.5	18.9	36.0	29.6
OpenHermes 2.5	**29.7**	53.0	**69.5**	18.6	35.4	36.8	40.5
SlimOrca	27.9	37.1	57.2	9.8	22.6	24.1	29.8
UltraChat	27.9	49.6	46.9	13.9	25.6	32.3	32.7
Tulu V3 Mix (subsampled 78K)	27.2	**68.7**	54.1	18.6	34.8	39.9	40.6
Magpie-Pro-SFT	27.0	52.0	34.3	4.2	36.0	36.8	31.7
Magpie-Air-SFT	25.6	57.1	28.2	8.8	33.5	38.6	32.0
OURS-MCTS	28.8	65.1	59.1	**20.4**	**40.9**	**42.1**	**42.7**

Table 4: The comparisons between our model and the baselines on different tasks.

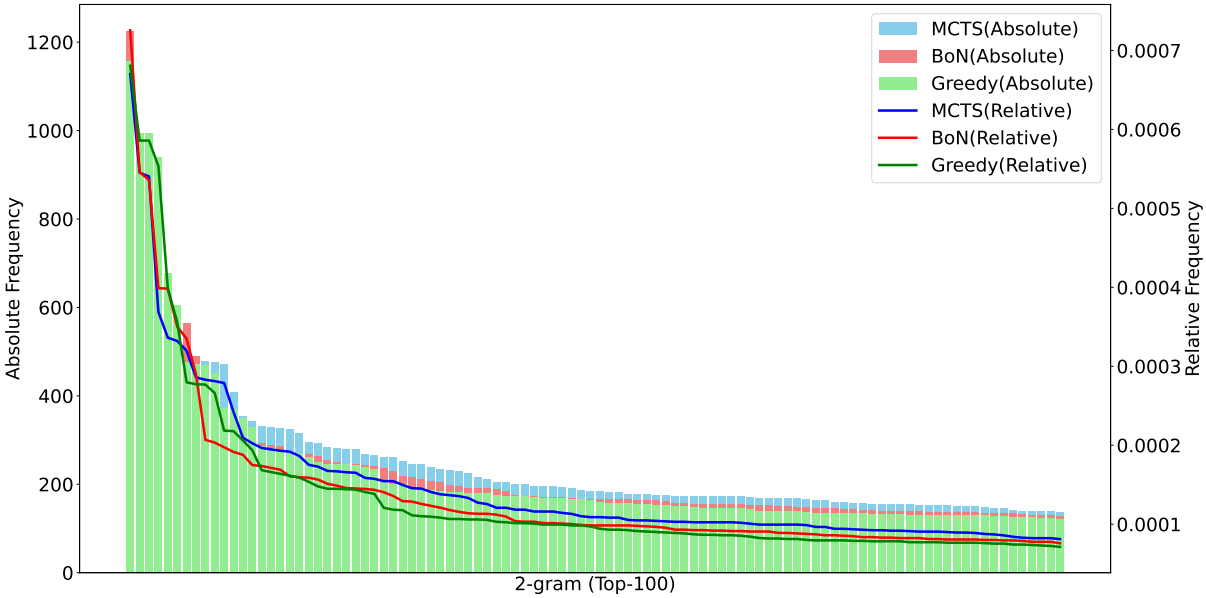


Figure 9: We sample 5,000 instructions in our SFT datasets (MCTS, BoN, Greedy) and compare the diverse 2-gram (Top-100). A more uniform distribution means that the dataset is more diverse and not concentrate to a specific pattern.

Algorithm 1 MCTS-Based Data Curation for SFT and DPO

Require: Dataset $\mathcal{D}$, Reward Model R, Max Iterations T, Score Threshold S_{thresh}
Ensure: SFT Dataset $\mathcal{D}_{\text{SFT}}$, DPO Dataset $\mathcal{D}_{\text{DPO}}$ Set $t \leftarrow 0$, $\mathcal{D}_{\text{SFT}} \leftarrow \emptyset$, $\mathcal{D}_{\text{DPO}} \leftarrow \emptyset$

```
for each d_i ∈ D do
    Initialize search tree T
    Rollout: Generate initial candidate answers {a_i} for d_i
    Evaluate each answer: s_i ← R(a_i)
    Set a* ← arg max_{a_i} s_i and S_best ← max s_i
    Store lowest-scoring answer a_worst ← arg min_{a_i} s_i
    while t < T and S_best < S_thresh do
        Selection: Traverse T using a selection policy (e.g., UCT)
        Expansion: Expand a new node by adding a set of new candidate answers
        Simulation: Generate and evaluate answer a' with reward s' = R(a')
        if s' > S_best then
            S_best ← s', a* ← a'
        end if
        Backpropagation: Update MC estimations along the path
        t ← t + 1
    end while
    D_DPO ← D_DPO ∪ {(d_i, a*, a_worst)}
    D_SFT ← D_SFT ∪ {(d_i, a*)}
end for
return D_SFT, D_DPO
```

Algorithm 2 PRM Data Collection from MCTS Searched Trees $\mathcal{T}_{\text{all}}$

Require: All Searched trees $\mathcal{T}_{\text{all}}$ in Dataset $\mathcal{D}$
Ensure: PRM(MC estimation) Dataset $\mathcal{D}_{\text{PRM-MC}}$, PRM(Cumulative Reward) Dataset $\mathcal{D}_{\text{PRM-CR}}$

```
Initialize D_PRM-MC ← ∅, D_PRM-CR ← ∅
for each T_i in T_all do
    for each node n ∈ T do
        Let C(n) be the set of child nodes of n
        if C(n) ≠ ∅ then
            Identify child with highest MC estimation: a_max ← arg max_{a∈C(n)} MC(a)
            Identify child with lowest MC estimation: a_min ← arg min_{a∈C(n)} MC(a)
            D_PRM-MC ← D_PRM-MC ∪ {(a_max, MC(a_max))} ∪ {(a_min, MC(a_min))}    ▷ D_PRM-MC collection
        end if
        Identify trajectory τ_max and τ_min with their corresponding cumulative reward calculated by Equation 6
        D_PRM-CR ← D_PRM-CR ∪ {(τ_max, CR(τ_max))} ∪ {(τ_min, CR(τ_min))}    ▷ D_PRM-CR collection
    end for
end for
return D_PRM-MC, D_PRM-CR
```