Multi-step Problem Solving Through a Verifier: An Empirical Analysis on Model-induced Process Supervision

Anonymous ACL submission

Abstract

Process supervision, using a trained verifier to evaluate the intermediate steps generated by a reasoner, has demonstrated significant improvements in multi-step problem solving. In this paper, to avoid the expensive effort of human annotation on the verifier training data, we introduce Model-induced Process Supervision (MiPS), a novel method for automating data curation. MiPS annotates an intermediate step by sampling completions of this solution through the reasoning model, and obtaining an accuracy defined as the proportion of correct completions. Inaccuracies of the reasoner would cause MiPS underestimating the accuracy of intermediate steps, therefore, we suggest and empirically show that verification focusing on high predicted scores of the verifier shall be preferred over that of low predicted scores, contrary to prior observations on human curated data. Our approach significantly improves the performance of PaLM 2 on math and coding tasks (accuracy +0.67% on GSM8K, +4.16% on MATH, +0.92% on MBPP compared with an output supervision trained verifier). Additionally, our study demonstrates that the verifier exhibits strong generalization ability across different reasoning models.

1 Introduction

Multi-step problem solving (e.g., math problems and coding challenges) showcases the capabilities of machine intelligence. While researchers have shown that model- and data-upscaling still hold powerful for large language models (LLMs) on multi-step problem solving (Achiam et al., 2023; Touvron et al., 2023; Team Gemini et al., 2023; Huang et al., 2022; Azerbayev et al., 2023; Luo et al., 2023a; Yu et al., 2023b), even the state-of-the-art LLMs still produce easily observable mistakes. Furthermore, standard fine-tuning directly does not yield consistent and significant improvements (Luo et al., 2023a; Yu et al., 2023b; Ni et al., 2022).

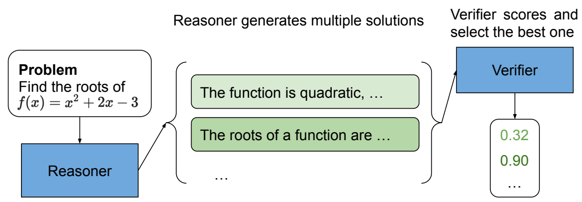


Figure 1: An illustration of the reasoner-verifier paradigm. The verifier predicts scores for the solutions generated by the reasoner, and selects the solution with the highest score.

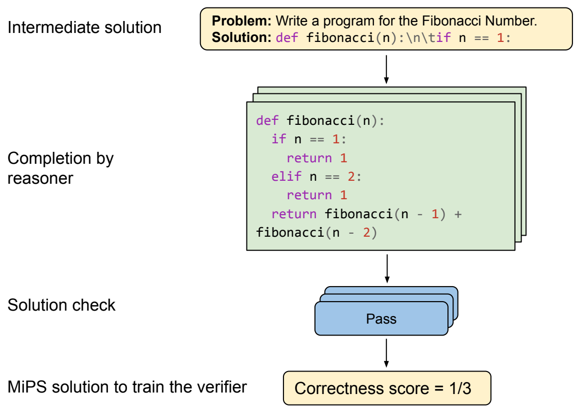


Figure 2: The Model-induced Process Supervision (MiPS) data construction method we introduce in this work. By completing an intermediate solution with a reasoner several times, we can obtain the percentage value of these completions being correct. These annotations are used to train a process supervised verifier.

The reasoner-verifier paradigm (Fig. 1) is as an inference-time technique where the goal is to pick one model-generated solution among many, since it is observed that there often are some correctly generated solutions. In particular, self-consistency (Wang et al., 2022) is a special case of the verifier that picks the solutions that shares the majority answer with others (e.g., math tasks where the answer is a number). LLM-based verifiers are more general, as they could apply to arbitrary text solutions (e.g., code that implements a function)

Training a verifier in a supervised fashion has demonstrated strong performance in both coding and math language problems. Cobbe et al. (2021) showed that by simply gathering correct and incorrect solutions to train a binary classification model and using such model to pick the highest confidence solution generated by the reasoner during inference time, the accuracy can be improved significantly. More recent studies suggest that verifying on intermediate steps could offer better guidance than training solely on the whole solutions (Li et al., 2022; Uesato et al., 2022; Paul et al., 2023; Lightman et al., 2023; Feng et al., 2023; Yu et al., 2023a; Liu et al., 2023a; Wang et al., 2023). As such, verifiers trained (and applied) on intermediate steps are called process supervised verifiers (PSV), whereas those trained on whole solutions are called output supervised verifiers (OSV). In prior work, process supervision data is either obtained by an ad-hoc algorithm (Li et al., 2022; Paul et al., 2023), or through expensive human annotations (Uesato et al., 2022; Lightman et al., 2023), lacking an automatic and generic way of constructing of annotations of intermediate solutions.

Training verifier models require solution wise or step-wise labels, which is expensive to collect. There have been a series of work following an LLM-as-a-verifier approach where an off-the-shelf LLM is employed to judge the solutions through prompting (Madaan et al., 2023; Kim et al., 2023; Pan et al., 2023). However, while such work may have seen improvements on language tasks, they haven't been very successful in math or coding problems (Huang et al., 2023; Luo et al., 2023b).

In conclusion, to achieve optimal quality, training data is needed for building a strong verifier model. On the other hand, manually collecting solution verification labels is expensive and non-scalable. In this work, we propose to use Monte Carlo Sampling on the completions of the intermediate solutions to obtain step-wise training annotations (Fig. 2). Specifically, for each intermediate solution, we complete the solution with the reasoner several times through a sample decoding mechanism, and the percentage of the completed solutions being correct is referred to as the correctness of the solution. The correctness scores are used to train a PSV. Because of the nature of involving the reasoning model's completion on the intermediate solutions, we call the construction of this data Model-induced Process Supervision (MiPS). While such an idea is also explored in a concurrent work (Wang et al., 2023), we supplement with analysis of using MiPS constructed data. We find that because the reasoner model, which completes the solutions, is not perfect, the noises it introduces would affect the design choices of training and using the process supervised verifier:

- We analyzed various ways to merge step-wise prediction scores to a single score value (we refer to this process as using an aggregation function) when using the verifier. Prior work used an aggregation function that focuses on low predicted scores and worked well for PSV trained on noise-free human annotated data (Lightman et al., 2023). For the noisy MiPS data, we suggest aggregation functions that focus on high predicted scores.
- We re-examine the usefulness of process supervision by isolating the trained PSV and studying the benefits of incorporating the predicted score from each intermediate step during verification. Our results reveal that (1) the verification scores from later intermediate steps are indeed useful even for a PSV trained on the noisy MiPS data, however, the earlier step scores could hurt the verification; and (2) only using the PSV predicted score of the last step, in similar fashion as OSV, can sometimes be much better than OSV itself, indicating process supervision data can regularize OSV training.
- We show that verifiers trained on MiPS data generated by a reasoner can transfer to validate solutions by a different (and more competent) reasoner. This indicates that MiPS would not produce verifiers that are overly biased towards mistakes of the reasoner that generated the data.

Following in this paper, we will provide a more complete review of related works, a precise definition of our method, and empirical results of the method and analysis on two math problem datasets and one coding dataset. The contributions of the paper are mainly (1) we propose MiPS to construct process supervision data automatically for training process supervision verifiers; (2) we extend the evaluation of problem solving verifiers to coding problems; (3) we provide empirical analysis on design choices and properties of the trained verifier from MiPS data.

2 Related Works

The advances of problem solving of LLMs can be broadly characterized into two regimes, first by *training* a better reasoning model and the second by *validating* the solution from the reasoning model at inference time.

Pre-training/Fine-tuning Better Reasoners. Standard training recipes also transfer to training better reasoners for problem solving. During pre-training, larger model sizes and training compute yields an LLM that is more competent in multiple aspects (Achiam et al., 2023; Touvron et al., 2023; Anil et al., 2023, *inter alia*). Within fine-tuning, it is also observed that transfer learning (Azerbayev et al., 2023) from a pile of generic math datasets, training on an augmented dataset of failure examples or diverse statements (Huang et al., 2022; Luo et al., 2023a; Yu et al., 2023b; Ni et al., 2022) leads to improvements. Despite these approaches, it is apparent that (1) the state-of-the-art LLM still can fail at simple mistakes during multi-step problem solving and (2) the improvement of a simple verification method by majority voting (self-consistency (Wang et al., 2022)) is still significant upon fine-tuning. Therefore, the exploration of verifiers to validate and pick the solutions is necessary.

Validating Through LLM-as-a-verifier. There have been numerous attempts on using the LLM reasoner itself to correct and validate its generated solutions. Madaan et al. (2023); Kim et al. (2023) and many methods surveyed in Pan et al. (2023) broadly follow the strategy where the LLM validates and provides feedback to the generated solutions through prompts. Huang et al. (2023) and Luo et al. (2023b) revisited these methods and found that LLMs are not good verifiers for equally competent solutions, as such methods improves marginally on math word problems.

Validating Through Trained Verifiers. In contrast, verifiers trained on a human labelled dataset does show significant improvements (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2023). Importantly, Lightman et al. (2023) showed that on a challenging competition-level mathematics problem set (Hendrycks et al., 2021), verifiers trained on annotated intermediate solutions (PSV) surpasses verifiers trained on final solutions by a large margin, and both substantially better than self-consistency. Other analysis also emphasize on the importance of step-wise feedback: Uesato et al. (2022) showed that PSV selects solutions that are more accurate in their reasonings and Yao et al. (2023); Feng et al. (2023); Liu et al. (2023b), *inter alia*, showed that during decoding, LLMs can be guided towards better solutions step-by-step. We believe that improving the training of a PSV, and especially, identifying a scalable solution to generate the process supervision data, is of imminent importance. Therefore, in this work, we identify an automatic and generic solution to generate process supervision data (MiPS), and conducted detailed analysis centered on the noises of this automatic process.

Math-Shepherd. Coincidentally, such an automatic process supervision data curation method was studied concurrently and independently by (Wang et al., 2023). We share a generally similar methodology with their work, with a few minor design differences we highlight in later sections. The empirical results of MiPS is similar on the two datasets we share (GSM8K and MATH) despite using different, but about competent, LLMs. Their work extended training the verifier by applying it to fine-tune the reasoner through reinforcement learning, while our work included an additional coding dataset (MBPP) and provided analysis on the design choices of using the verifier, addressing the data noises. We believe these two works compliment each other.

3 Model-induced Process Supervision

We consider the reasoner-verifier framework where we start with a fairly competent reasoner on a task, generate verifier training data on a given set of problems with the reasoner, and train a verifier on the data to validate some new generated solutions by a reasoner. We first discuss Model-induced Process Supervision, our data curation method that automatically creates process supervision data. Then, we discuss the details about the verification process.

3.1 Obtaining MiPS data

MiPS constructs process supervision data through Monte Carlo sampling. First, we employ a reasoner model r_g to generate a fix number of n_g solutions for each problem, using temperature based decoding with a temperature of t_g. Then, for each solution, we decompose them into individual steps (we treat each line in a solution as an individual step). After that, for each intermediate solution containing a prefix list of steps, we employ a reasoner model r_{mc} to generate again n_{mc} solutions, with a temperature of t_{mc}, completing the intermediate solution. For each completed intermediate solution, we calculate the percentage (out of n_{mc}) of them being correct, and these correctness values comprises the MiPS data. In all experiments in this paper, we consider $r_g = r_{mc}$, namely, the reasoner model that is used to estimate the intermediate solution's correctness is the same model that generates the solution data. This is particularly the most challenging case for MiPS, otherwise, using a more capable reasoner for the completion can enjoy a reduction of noise in MiPS data.

3.2 Training an Output Supervised Verifier

To understand how well the process supervised verifier (PSV) trained from MiPS is, it is necessary to consider the vanilla output supervised verifier (OSV), which uses the same amount of human labeling resources. The training data for OSV are the generations from the reasoner r_g with the same temperature value t_g. The verifier itself is nothing different from a standard language model, apart that it is appended with a classification head on the final token of the input. This is also known as the solution-level verifier in Cobbe et al. (2021).

3.3 Training a Process Supervised Verifier

The differences of training PSV and OSV are:

- To enable predicting a score at each step in the solution, we mark the last token of each step (e.g., if each step is represented as a single line, the last token will be the new line token), and optimize step-wise predictions at each step at the same time. During inference, we would also obtain a score for each step in a solution.
- While for the output supervision data, or human labelled process supervision data, the score is either 0 or 1, for MiPS data, the correctness scores are percentage values. The training objective considered in this work is to learn the exact percentage values c_i for the ith step in the solution directly. However, we note that it is possible to consider a different learning objective. For example, Wang et al. (2023) considered learning a binarized score:

$$\tilde{c_i} = \begin{cases} 1, & \text{if } c_i > 0.0 \\ 0. & \text{otherwise.} \end{cases}$$

  In later analysis, we compare these two objectives.

3.4 Aggregating Step-wise Predictions

The trained verifier is used to score the solutions generated by the reasoner. For OSV, the verifier prediction can be directly used as the score for the solution. For PSV, the verifier predictions are a list of predicted probabilities $p_1, p_2, \ldots, p_n$, one for each step in the solution. Aggregating the predictions into a final score is necessary. Lightman et al. (2023) considered two aggregation functions:

$$\texttt{min} = \min\{p_1, p_2, \ldots, p_n\},$$
$$\texttt{sum\_logprob} = \sum_{i=1}^{n} \log p_i = \log \prod_{i=1}^{n} p_i,$$

They claimed that both are equivalently good aggregation functions. In later analysis, we show that for MiPS data, these two functions are underperforming for the trained verifier, while,

$$\texttt{max} = \max\{p_1, p_2, \ldots, p_n\},$$
$$\texttt{sum\_logit} = \sum_{i=1}^{n} \log \frac{p_i}{1-p_i},$$
$$\texttt{mean\_odd} = \frac{\sum_{i=1}^{n} \frac{p_i}{1-p_i}}{n},$$

are much better. We provide an analysis with a much larger set of aggregation functions and suggest that MiPS data prefers aggregation functions that focus on high prediction scores rather than lower ones.

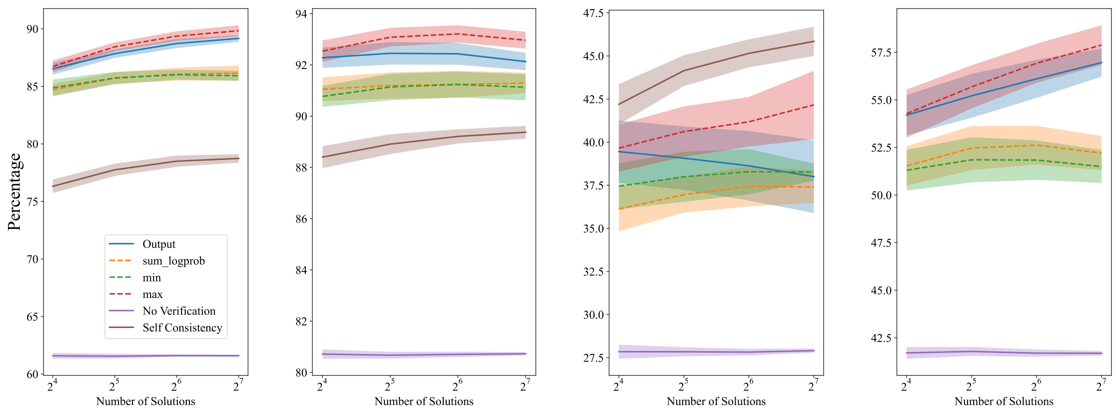
Dataset	GSM8K	MATH	MBPP
Domain	math	math	coding
Fine-tuning # Data	2000	4000	0
Verification Training # Data	5000	8000	384
Testing # Data	1319	500	500
Average Steps	4.5	11.0	7.0

Table 1: We show the statistics of the datasets we use in this paper. The average number of steps is depicted with a granularity of 0.5, using PaLM 2-S for GSM8K and MBPP, and PaLM 2-L for MATH. We note that these are not the most standard data splits, for reasons explained in Sec 4.2.

4 Analysis

4.1 Models

In our experiments, we consider two LLMs, PaLM 2-S and PaLM 2-L (Anil et al., 2023) to conduct our experiments on. We intend to understand the capability of MiPS data and analyze design choices of the verifier when trained on it. A concurrent work (Wang et al., 2023) conducted a similar experiment on a different set of LLMs, namely LLama2, LLemma, Mixtral, and Deepseek (Touvron et al., 2023; Azerbayev et al., 2023; Jiang et al., 2024; Bi et al., 2024). Detailed experimental settings and hyperparameters can be found in Appendix A.1.



(a) GSM8K, PaLM 2-S (b) GSM8K, PaLM 2-L (c) MATH, PaLM 2-L * (d) MBPP base model

Figure 3: We apply the trained output- and process-supervised verifiers on various combinations of model and datasets. Self-consistency scores are given as a reference, however, it would be not applicable to general multi-step reasoning tasks (e.g., figure (d), coding). We use the default training objective that directly learns the estimated accuracies and the `max` aggregation function for the process verifier. In the x-axis, we vary the number of generated solutions to apply the verifier on, and in the y-axis we plot the performance (accuracy %). The standard deviation is also given. As a reference, we note that the purple line, representing the average performance of the generated solutions of the reasoner without any verification, matches the expectation to be an (almost) flat horizontal line with decreasing standard deviation. *While the reasoner that generates MiPS data and the reasoner that the verifier validates on is PaLM 2-L, the verifier is trained from a PaLM 2-S.

4.2 Datasets

We use two math datasets and one coding dataset for evaluations in this paper.

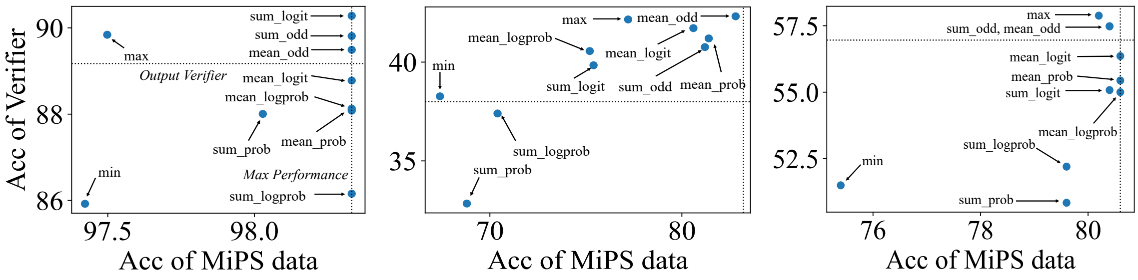
- **GSM8K** (Cobbe et al., 2021) is a dataset of grade school math problems.
- **MATH** (Hendrycks et al., 2021) is also a math word problems dataset. It consists of math problems of high school math competitions.
- **MBPP** is an entry-level Python programming dataset. The questions are coding challenges along with a test case that defines the function format and the solutions are Python code that is expected to solve several hidden test cases.

Table 1 contains detailed statistics about the datasets, and Appendix A contains more information on how we split these datasets into training and evaluation.

4.3 Directly Applying MiPS

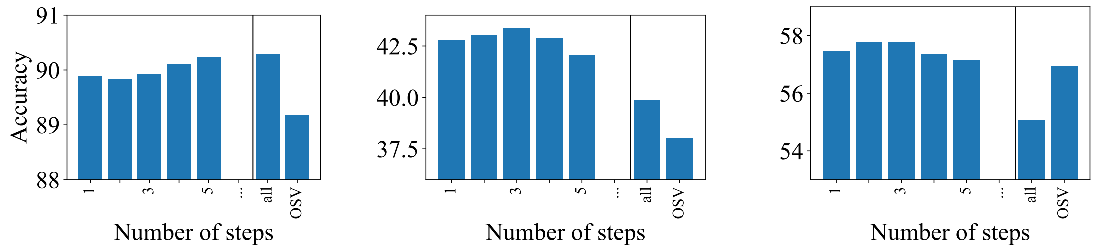
We first present the performance of the process verifier trained on MiPS data, using the default training objective on correctness scores directly, and the `max` aggregation function on the three datasets. For this experiment, we varied the number of solutions to be verified by the verifier from 2 to 128, to clearly depict the trend of the compared verifier's performance. The results are shown in Fig. 3. The plots convey several pieces of information.

- It is evident that using any verifier improves significantly upon no verification, matching with the initial assumption that verification plays a vital role in multi-step problem solving.
- In all experiments, the verifier trained on MiPS using the max aggregation function showed stronger results than output verification. On GSM8K, the process verification is better than self-consistency. On MATH, the performance lacks a bit. We note that this may be because we are training a less competent verifier (PaLM 2-S) than the reasoner (PaLM 2-L). Wang et al. (2023) showed improvements upon self-consistency when the verifier and reasoner are of the same sizes.
- The high performance of `max` may be unexpected, as `max` seemly would be biased towards the first few correct steps of an incorrect solution. In later analysis, we will show that (1) `max` favors high scores, similar to some other aggregation functions that perform well, that is preferred on MiPS data; (2) Due to higher noise in the earlier steps in MiPS data, the prediction scores of the earlier steps is of lower value (i.e., model confidence is lower), thus showing less affect to `max`.
- In all experiments, the `sum_logprob` (product of probabilities) and `min` aggregation function are much worse than `max` or even using OSV, never-

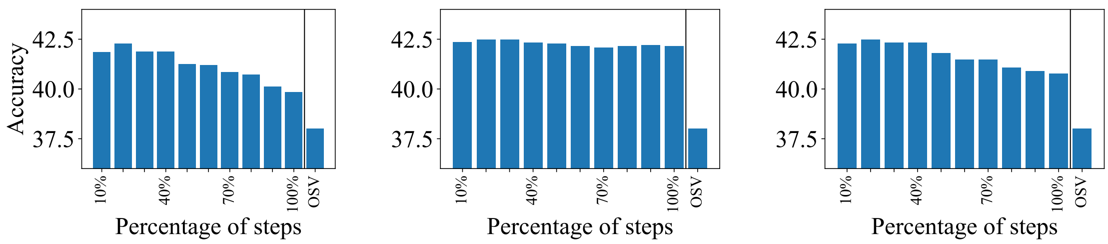


(a) GSM8K, PaLM 2-S (b) MATH, PaLM 2-L (c) MBPP, PaLM 2-S

Figure 4: For each aggregation function, we show its accuracy on the MiPS data generated and the accuracy of using it with the PSV trained. We additionally plot two lines for easier understanding of the figure, a horizontal line corresponding to the performance of OSV and a vertical line corresponding to the maximum accuracy achievable by the reasoner on the dataset (some problems are not solvable by the reasoner among the all solutions we generate).



(a) GSM8K, PaLM 2-S,`sum_logit` (b) MATH, PaLM 2-L, `sum_logit` (c) MBPP, PaLM 2-S, `sum_logit`



(d) MATH, PaLM 2-L, `sum_logit` (e) MATH, PaLM 2-L, `max` (f) MATH, PaLM 2-L, `sum_odd`

Figure 5: We plot the performance of using various aggregation functions with PSV while restricting it to predict only the last k steps or the last p percentage of steps.

theless still providing benefits over not using a verifier.

- For several verifiers, we observe that the performance of the verifier is on a decreasing trend when the number of generations is high. This is particularly interesting since the larger the generations, the closer the performance should approximate the true verification performance. This would indicate that the while the verifier might identify some correct solutions with high scores, it also incorrectly predicts some fewer incorrect solutions with even higher scores, a sign of improper generalization.

From these results, we focus our analysis on two subjects: (1) the choice of aggregation functions, and (2) the effect of noise on generalization of the PSV.

4.4 Aggregation Functions

To start with the analysis, we consider ten aggregation functions (sum and means of log probabilities, probabilities, logits, and odds, and maximum and minimum value over all steps). We obtain their performance on the MiPS dataset and plot it with the performance of the verifier using the aggregation function on the test set (Fig. 4). We first observe

Model	Llemma	MM-Llemma	MM-Mistral
Soft + `max`	54.7	72.4	80.3
Soft + `min`	51.2	70.1	77.8
Hard + `max`	50.2	68.9	78.1
Hard + `min`	52.4	70.8	79.2

Table 2: We test on GSM8K the effect of different training objectives and aggregation functions (Hard & `min`, the combination used in Wang et al. (2023), and Soft & `max`, the combination we suggest). All 3 base models are 7B in size, and MM denotes the MetaMath (Yu et al., 2023b) fine-tuned version of them.

that in general, the two performances have positive correlations, indicating that it is possible to select an aggregation function on the MiPS dataset and use it during inference. Second, we notice that both `min` and `sum_logprob` have low performances not only during inference, but also in MiPS. This indicates that the poor performance of them likely is related to the construct of MiPS. Indeed, we realize that `sum_logprob` does not have a high correlation with a correct solution, as it naturally penalizes long solutions. For `min`, the possibility for a set of solutions to be wrongly verified using `min` is when the solution with the largest minimum correctness over all steps turns out to be wrong. This is actually not an unlikely event to happen, particularly in consideration when the reasoner makes an erroneous continuation to an initially correct solution. To clarify these better, we answer the following questions:

What are common in good aggregation functions for MiPS data? We believe a rule of thumb of a good aggregation function is a function that **values high scores highly**. Consider two functions, one that values high scores (selects the solution with the highest high scores, e.g., `max`) and one that values the low scores (selects the solutions with the highest low scores, e.g., `min`). The first function is wrong only when the highest score solution is incorrect, in a simple case where there is only one observed step score for each solution, the probability is $1 - s_{\text{max}}$, where s_{max} is the score. Similarly, for `min`, the probability that the solution with the highest minimum score is wrong is $1 - s_{\text{min}}$. Since $s_{\text{max}} \geq s_{\text{min}}$, the first function shall be preferred. This is in line with the observation from Fig. 4 that aggregations of odds and logits are usually better than that of probabilities and log probabilities.

Why did `sum_logprob` and `min` work well in Lightman et al. (2023) and Wang et al. (2023)? In Lightman et al. (2023), the dataset is constructed by human identifying all (earliest) incorrect steps, which corresponds to a prediction of 0 for the verifier (i.e., following the analogy in the previous discussion, this indicates that $s_{\text{max}} = s_{\text{min}} = 1.0$). The `min` function would be correct on every instance in the training dataset, and if the verifier generalizes well, resembles human identification of mistakes on the test dataset. For Wang et al. (2023), we note a difference during MiPS data construction as their training objective is to predict the binary value of the correctness score, we discuss this more in Sec 4.6.

The aggregation function analysis would indicate that a good MiPS dataset score indicates a good aggregation function. This is not completely correct, since, a contradictory result is that the final step score, which is used to train the output supervised verifier, achieves 100% accuracy on the training dataset, while not as good as the process supervised verifier on the test set. This suggests that the output supervised verifier might encounter some generalization issues from the data, and MiPS data can help relieve them.

4.5 Different Length Aggregations

To understand the generalization issue, we illustrate the result of applying an aggregation function to only the last k steps or last p percentage steps of the solutions in Fig. 5. In the upper three plots, we show the performance of an aggregation function `sum_logit` on the three datasets with $1 \leq k \leq 5$. In the lower three plots, we show the performance of three aggregation functions on MATH with $10 \leq p \leq 100$. We only conducted this analysis on the MATH dataset, as it have solutions long enough such that looking at a percentage number of steps is sensible.

- For all experiments, the performance increases with a few more steps considered from the end. This indicates that the PSV predictions on the last steps brings in increasing value, suggesting that process scores indeed are beneficial.
- For most experiments, the performance starts to drop after including some early steps. This suggests that the quality of the predictions for the first steps are poor. We believe this is because MiPS has a poorer estimation of the first steps than the last steps, since intuitively it is hard to predict the correctness of a very early solution, causing burden for the verifier to learn.
- For `max`, the performance does not change significantly across including more earlier steps. We

GSM8K PaLM 2-S	No Verifier	Self Consistency	OSV	PSV w/ sum_logit
→ PaLM 2-S	61.6	78.7	89.5	90.5
→ PaLM 2-L	80.7	89.4	92.1	92.6
→ gpt-turbo-3.5	72.5	86.2	88.0	89.1
MBPP PaLM 2-S	No Verifier	OSV	PSV w/ sum_logit	PSV w/ sum_logit (last 3 steps)
→ PaLM 2-S	41.7	56.8	54.2	57.8
→ PaLM 2-L	42.4	56.6	55.0	57.4
→ gpt-turbo-3.5	66.2	67.6	67.6	68.2

Table 3: We train a verifier based on PaLM 2-S and data generated by PaLM 2-S and test its applicability to transfer to validate solutions generated by two different reasoners, PaLM 2-L and gpt-turbo-3.5. A reference score of validating solutions generated by PaLM 2-S itself is also given. We evaluate this on two tasks, GSM8K and MBPP.

examined the predicted scores and find the usually earlier steps are smaller in value, causing it to contribute little to max. This is another evidence that PSV trained on MiPS data might suffer from noise in the earlier steps.

- In all experiments, using the last-step process verifier predicted value is more beneficial than output supervision alone. Recall that this is not because of the problem of data quantity, as we upscaled the data to train the output verifier. We suggest that this is because the process supervision data is of more diverse context, thus helping the model in generalization.

4.6 Different Training Objectives

The main difference in the method of ours and Wang et al. (2023) is the training objective of the verifier, where we train the verifier to directly predict the estimated accuracies (Soft Objective), and they train the verifier to predict a binarized value (non-zeroness) of the accuracy (Hard Objective). In our previous analysis, we noted that since the reasoner is imperfect, MiPS would provide underestimated accuracies of the intermediate steps, which is harmful to aggregation functions that focus on low values (e.g., min). In contrast, the non-zeroness of the accuracy would cause an overestimation of the accuracy, which, by the same argument, would be harmful to aggregation functions that focus on high values (e.g., max). To verify this, we conduct the experiments using the same language model as Wang et al. (2023) on the GSM8K dataset, using both training objectives and aggregation functions.

The experiment setting is detailed in Appendix A.2. The results are in Table 2. It is observed that, indeed, the max aggregation is better for the soft objective and the min aggregation is better for the hard objective. It also turns out that soft objective with the max aggregation consistently outperforms hard objective with min aggregation. We believe this to be a strong motivation for the use of the soft objective in MiPS.

4.7 Transferring to a Different Reasoner

Finally, we provide an auxiliary experiment to check whether the trained verifiers would transfer to different reasoning models. We apply the verifiers trained on reasoning data generated by a PaLM 2-S and use it to valid solutions generated by stronger reasoners (reasoners having higher *No Verifier* accuracy). We find the sum_logit aggregation function working well in this case. The result is shown in Tab. 3, which shows that the trained verifier transfers to different and stronger reasoners with a strong validation ability, indicating that the verifier is not learning something overly specific to the reasoner that generates the data.

5 Conclusion

In this work, we introduce MiPS to automatically annotate intermediate solutions for multi-step problem solving. Such data can be used to train a process supervised verifier that validates solutions generated by a reasoner. On two math datasets and one coding dataset, we demonstrated that MiPS improves the ability of picking the correct solution over an otherwise trained output supervised verifier. We conduct analysis on the aggregation function used to pick the solution and suggest that compared to verifiers trained on human-annotated process supervision, MiPS data trained verifiers prefer different aggregation functions. We also showed that such verifiers do not overly emphasize on the mistakes of the reasoner that produced the data, and can be transferred to different reasoners. Future work could explore creating a scalable way to obtain MiPS data for each token in solutions to train a more competent verifier and use it to tune the reasoner via reinforcement learning.

6 Limitation

6.1 Underperformance on the MATH dataset

In our work, we did not manage to conduct all experiments using the same, large model. Especially for the MATH dataset, we had to train a smaller verifier to compensate of the long sequence length and data size. This probably led to us finding a lower performance of process and output verifier than the straightforward self-consistency. We believe in general that this is not true, as Lightman et al. (2023) and Wang et al. (2023) both showed that process/output verifier should output self-consistency on the MATH dataset.

6.2 Efficiency

MiPS, while automatic, requires a non-trivial amount of computation effort in generating the dataset to train the verifier. We did not attempt to reduce the computational effort, as we'd like to show the most direct comparison with no verifiers and output supervised verifiers. We do believe it is very possible to reduce the computation costs, for example, by avoiding creating data on every intermediate solutions, and we suggest future work to explore this direction.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

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

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

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

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

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language models can solve computer tasks. *arXiv preprint arXiv:2303.17491*.

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

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*.

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

Jiacheng Liu, Andrew Cohen, Ramakanth Pasunuru, Yejin Choi, Hannaneh Hajishirzi, and Asli Celikyilmaz. 2023b. Making ppo even better: Value-guided monte-carlo tree search decoding. *arXiv preprint arXiv:2309.15028*.

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, Zi Lin, Yinxiao Liu, Lei Shu, Yun Zhu, Jingbo Shang, and Lei Meng. 2023b. Critique ability of large language models. *arXiv preprint arXiv:2310.04815*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *arXiv preprint arXiv:2303.17651*.

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Alex Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. 2022. Learning math reasoning from self-sampled correct and partially-correct solutions. In *The Eleventh International Conference on Learning Representations*.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*.

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2023. Refiner: Reasoning feedback on intermediate representations. *arXiv preprint arXiv:2304.01904*.

Team Gemini, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

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*.

Peiyi Wang, Lei Li, Zhihong Shao, RX Xu, Damai Dai, Yifei Li, Deli Chen, Y Wu, and Zhifang Sui. 2023. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. *CoRR, abs/2312.08935*.

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

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

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023a. Outcome-supervised verifiers for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

A Datasets

- **GSM8K** (Cobbe et al., 2021) is a dataset of grade school math problems. The solution is given in a one-line-per-step format with an exact numerical answer in the last line in the format of ####{answer}. To enforce the reasoner following this format, we use the first 2000 instances in its training set to fine-tune the reasoner model to follow such a format. The solutions to fine-tune are from the training set. We use the coming 5000 data to train the verifier and evaluate the verifier on solutions generated by the reasoner on the test set.
- **MATH** (Hendrycks et al., 2021) is also a math word problems dataset. It consists of math problems of high school math competitions. The solutions are given in a format that mixes latex code and natural language. A dedicated solution checker was developed (Hendrycks et al., 2021; Lightman et al., 2023). While the dataset itself does not resemble steps into different lines, we prompted GPT-4 to break down the reference solutions into one step per line, and fine-tuned the reasoner on the line separated dataset to make it follow the format. We use the test split suggested in Lightman et al. (2023).
- **MBPP** is an entry-level Python programming dataset. The questions are coding challenges along with a test case that defines the function format and the solutions are Python code that is expected to solve several hidden test cases. We treat each individual line in the generated code as a step. For languages like Python, this resembles one statement per step. Due to the small dataset size, we can not afford to fine-tune the reasoner model, and decide to use 3 prompts in the validation split as in context examples to make sure the model generates code in the expected format.

A.1 Settings

Throughout the paper, we choose to use a temperature value of $r_g = r_{mc} = 0.7$ for both constructing MiPS and generating solutions on the test set. The number of generations for constructing MiPS n_g is set to 32 for GSM8K and MBPP, and 8 for MATH. The number of completions n_{mc} is also

32 for GSM8K and MBPP, and 8 for MATH. For GSM8K, we experiment with both PaLM 2-S and PaLM 2-L in the reasoner-verifier framework. For MATH, due to compute constraints, the reasoner we use is the PaLM 2-L, and the verifier trained is the PaLM 2-S. For MBPP, we find marginal differences in performance between using PaLM 2-S and PaLM 2-L as the reasoner, therefore we experiment only with the PaLM 2-S. During generation, all models are 8-bit quantized, and during training, we use a bfloat16 precision. Since MiPS contains an annotation for each intermediate step in the solution, it is naturally the number of steps times larger than output supervision. Therefore, we additionally generate more data to train the OSV. For training, we follow standard reward model training recipes, with an exception on the training epochs. Similar to Lightman et al. (2023), we also find it better to train the OSV for 1 epoch and the PSV for 2 epochs. For the OSV on MATH data, we find that training with a small 0.2 epochs (essentially training on less data) is better than training longer. For all experiments, we report the results of the average of 5 independently trained verifiers with different random seeds.

A.2 Settings of the Objective Experiment

To reduce the cost, when conducting the experiment to compare the two training objectives, we scaled down the experiment. On GSM8K, we used 2000 data points for verification training data generation, and $n_g = n_{mc} = 8$.