Self-Improvement for Math Problem-Solving in Small Language Models

Stanford CS224N Custom Project

Roberto Garcia
Institute of Computational and Mathematical Engineering
Stanford University
robgarct@stanford.edu

Shubhra Mishra
Dept. of Computer Science
Stanford University
shubhra@stanford.edu

Artyom Shaposhnikov
Dept. of Computer Science
Stanford University
artyom@stanford.edu

Abstract

Large language models (LLMs) have achieved impressive performance gains by scaling model and dataset sizes, but this data-hungry approach risks hitting bottlenecks from limited availability of high-quality labeled data. We investigate an alternative approach of self-improving language models using unlabeled data. Focusing on the challenging task of math problem-solving, we enable `Phi-2`, Microsoft's 2.7B parameter model, to iteratively improve itself on the `GSM8K` dataset without access to ground truth solutions. We observe that maintaining solution diversity during self-improvement is crucial - techniques that cause diversity to collapse quickly halt further progress. We propose several novel solution sampling approaches that encourage diverse solutions, including rephrasing problems to create linguistic variations and a new filtering method based on selecting for execution trace diversity. Combining these techniques, `Phi-2` achieves a 6% increase in mean accuracy on `GSM8K` ($66\% \rightarrow 72\%$), without accuracy drops on out-of-domain tasks. Moreover, on a difficult subset of problems, where `Phi-2` initially averages just 36%, our approach boosts performance to 42% by progressively learning to solve problems it previously failed on. Through empirical analysis, we highlight how 1. enhancing diversity through rephrasing and 2. focusing training on a curated set of high-confidence solutions is key to effective self-improvement, especially on challenging problems.

1 Key Information to include

- Mentor: Rohan Taori
- Team contributions: Shubhra helped with writing, some code, and created a synthetic comparison dataset. Artyom helped with model sampling and finetuning code frameworks, algorithms and running the code on the GPUs, and writing. Roberto helped with the rephrase phi-model, dataset processing tools and writing.

2 Introduction

In recent years, large language models (LLMs) have shown rapid increase in performance across a variety of tasks. Prompting techniques such as Chain-of-Thought (CoT) and few-shot prompting have improved LLM performance margins without the need for additional training (Brown et al., 2020; Wei et al., 2023). However, significant improvement in LLMs is usually attributed to scaling laws and the use of pre-existing data (Kaplan et al., 2020; Chowdhery et al., 2022), or the use of high-quality synthetic data (Gunasekar et al., 2023). Another significant direction in the study of LLMs has been the creation of Small Language Models (SLMs) that boast LLM-like capabilities at a fraction of their size (Javaheripi and Bubeck, 2023; Liu et al., 2023; Eldan and Li, 2023; Pichai, 2023).

Because creating training datasets is difficult, and because of work towards a generalized agent that learns without additional data, research in self-improving language models has grown. Huang et al. (2023) use self-improvement techniques that incorporate self-consistency (a process that involves sampling multiple reasoning paths, and selecting the most consistent answer) to finetune a 540B-parameter language model. Because math problem-solving has proven to be an exceptionally challenging test for language models, Ni et al. (2023) use self-sampled fully- and partially-correct solutions to finetune `GPT-Neo` models for math-reasoning tasks. Drawing inspiration from both Huang et al. (2023) and Ni et al. (2023), we devise an approach to finetune a small language model from an unlabeled dataset. Concretely, we propose a simple training framework using different solution-sampling techniques that allow the model to self-improve during training. We implement our approach using Microsoft's `Phi-2` as our SLM and unlabeled data from the `GSM8K` dataset.

Our contributions are summarized as follows:

1. We demonstrate the ability of SLMs to self-improve. We obtain a 6% increase in mean accuracy $(66\% \rightarrow 72\%)$ on the `GSM8K` benchmark, without the use of ground truth answers and without loss of accuracy on out-of-domain tasks.
2. We separately analyze a subset of hard problems, where the model has low confidence and accuracy. Self-improving on this subset by selecting self-consistent solutions wasn't effective, but we devise a method that encourages the model to generate diverse solutions and use an advanced solution filtering technique that is superior to self consistency. Using this technique, SLMs self-improve on hard problems, without any form of supervision. We achieve a 6% increase in mean accuracy $(36\% \rightarrow 42\%)$ on the hard problems.
3. We highlight the importance of diversity in self-sampled data for improving a language model's capabilities. We also propose methods for obtaining diverse and high-confidence solutions from self-sampled solutions.
4. We open-source our code and models on GitHub and HuggingFace.

3 Related Work

Huang et al. (2023) show that a large language model can self-improve using unlabeled data: they use data without ground-truth solutions and leverage CoT reasoning and self-consistency to achieve state-of-the-art results on the `ARC`, `GSM8K`, `OpenBookQA`, and `ANLI-A3` benchmarks. The model they use has 540B parameters, making it computationally hard to self-improve with more than one iteration of finetuning. Additionally, their evaluation framework uses 32 output paths for sampling and evaluation with self-consistency, while we use only 10. In our work, we use a 2.7B parameter model and train for multiple iterations. Using a model that is $1/200$th the size of theirs, $1/3$ as many output paths, and a very small fraction of GPU resources, we achieve a similar increase in performance on the `GSM8K` dataset.

Ni et al. (2023) finetune the `GPT-Neo` models using self-sampled partially- and fully-correct solutions. The paper uses the `PASS@k` evaluation technique, letting the language model generate k different solutions, and marking something as correct if any one of the k generations yields the correct answer. Their evaluations are also based on the 5.5K problems that they self-improve on, as opposed to evaluating on the unseen test set. In our work, we leverage self-consistency and evaluate on the test set, which our model does not observe during training. Additionally, their model uses ground-truth labels from the `GSM8K` dataset, which we do not.

4 Approach

In our research, we finetune Microsoft's `Phi-2` via a self-improvement process that uses the training split of the `GSM8K` dataset (Cobbe et al., 2021). Our study focuses on enabling `Phi-2` to iteratively self-improve, without access to ground truth labels, on math problem solving. Concretely, we propose a simple training framework together with 3 solution-sampling techniques, which allow `Phi-2` create it's own training set to further improve on it.

The input to the sampling step is a dataset consisting of problems, a set of prompts, and a model. These sampling techniques output a training dataset consisting of multiple (prompt, problem, solution)

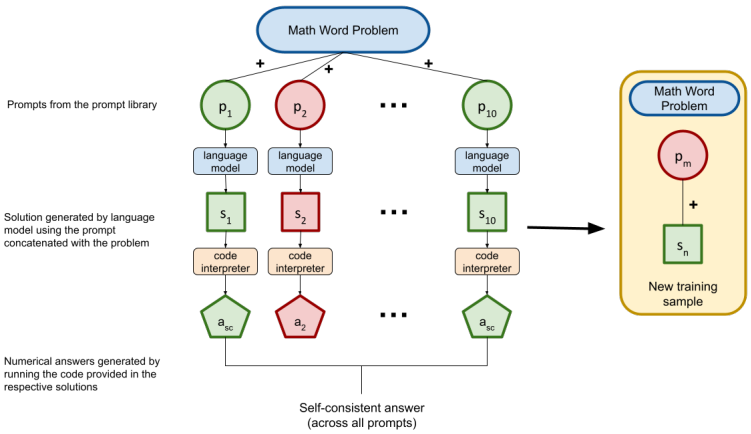

Figure 1: Cross Prompt Sampling

triplets, where the solution is self-sampled from the model we are improving. Using any of these sampling techniques in the training framework results in a self-improvement procedure, where sampling and training are done iteratively until convergence.

4.1 Training framework

Throughout this work, we used the same training framework for self-improvement while only varying the sampling stage of it. Our training framework is shown in Figure 4 and described as follows:

1. Let D'_{train} represent the training split of our data without labels, and P be the set of 10 prompts shown in Appendix E. We then take a subset of D'_{train} and all prompts from P and generate triplets of (prompt, problem, solution) for every problem in the subset. These triplets are generated using one of the sampling and filtering approaches described in Section 4.2. This yields a new dataset D_{train}, consisting of (prompt, problem, solution) triplets.
2. Then, we take the obtained D_{train} and train the current model in a supervised fashion (using the self-generated solutions) using Low-Rank Adaptation (LoRA) from Hu et al. (2021).
3. Finally, we repeat steps 1 and 2 for multiple iterations.

4.2 Solution sampling techniques

4.2.1 Cross-prompt sampling

Cross-prompt sampling, as shown in Figure 1, consists of creating multiple inputs for each math problem. Each input is created by concatenatenating a math problem with each of the 10 prompts shown in Appendix E. Then, for each of these 10 inputs, we use the model to generate a code solution, which we run through a code interpreter to generate a final numerical answer. We use self-consistency (Wang et al., 2023) to calculate the most common numerical answer among the 10 outputs. Then, we randomly choose a prompt that resulted in a non-self-consistent answer, and a solution that generates a self-consistent answer. We concatenate these, generating a new training sample in the format (problem, prompt, solution). This is repeated for every problem to create the training dataset. Intuitively, curating our dataset this way allows for prompts that result in bad solutions to catch up to the prompts that result in good solutions, effectively teaching the model to correct its mistakes.

Concretely, let D be the given dataset consisting of different problems, P be the given set of prompts and M be the given model. Then, cross-prompt sampling works as follows:

1. For each prompt in P and for each problem in D, we sample 1 solution using model M with $temperature = 0.3$ and $top_p = 0.3$. This yields $|P|$ solutions for every problem, each generated by one prompt.

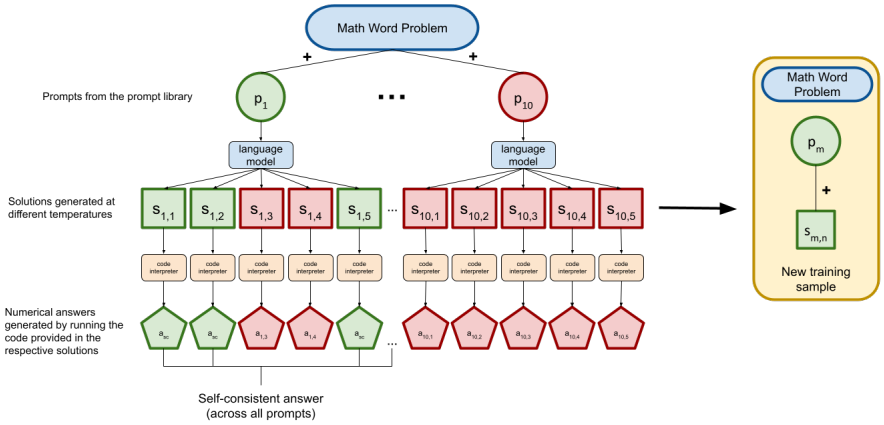

Figure 2: Single-prompt Sampling

2. Then, for a given problem, we randomly select a non-consistent prompt p_{nc} from the subset of prompts that resulted in a non-self-consistent solution for the problem at hand. Further, we pick a self-consistent solution s_{sc} from the subset of self-consistent solutions obtained by all prompts. We then build a triplet (problem, p_{nc}, s_{sc}).
3. After doing step 2 for all problems, we end up with a dataset consisting of $|D|$ triplets, which are further used for training.

Notably, when using this sampling approach in our training framework in Figure 4, we observed an increase in the average accuracy of the model. However, after 2 iterations of training, it also resulted in a diversity collapse in solutions (as observed in Table 1's `PASS@10` column) which quickly halted improvement.

4.2.2 Single-prompt temperature sampling

To address the shortcomings of cross-prompt sampling causing a diversity collapse in solutions, we devised the single-prompt temperature sampling approach (Figure 2). Simply put, single-prompt temperature sampling builds (problem, prompt, solution) triplets by pairing a prompt only with a self-consistent solution generated by itself, as opposed to a self-consistent solution generated by any other prompt.

The intuition behind this approach is that, since cross-prompt sampling has a diversity collapse given all prompts yield very similar solutions, it is critical to maintain the diversity of solutions provided by different prompts. It follows that training a prompt only on good solutions generated by itself would not affect the diversity of solutions obtained by using different prompts.

Concretely, let D be the given dataset consisting of different problems, P be the given set of prompts and M be the given model. Then single-prompt sampling works as follows:

1. For each prompt in P and for each problem in D, we sample 5 solutions using model M at 5 temperatures (0.3, 0.4, 0.5, 0.6 and 0.7). This yields $5 \cdot |P|$ solutions for every problem.
2. Then, we determine the self-consistent solution using self-consistency across all $5 \cdot |P|$ solutions.
3. For a given problem, we randomly select a prompt that results in the self-consistent answer, and any one of the prompt's self-consistent solutions. This results in the triplet (problem, p_{sc}, s_{sc}).
4. After doing step 3 for all problems, we end up with a dataset consisting of $|D|$ triplets, which are further used for training.

In this approach, the diversity of samples was largely maintained. However, the model hit a ceiling in accuracy. We observed that for a subset of hard problems, the model would only generate incorrect solutions, even at high temperatures.

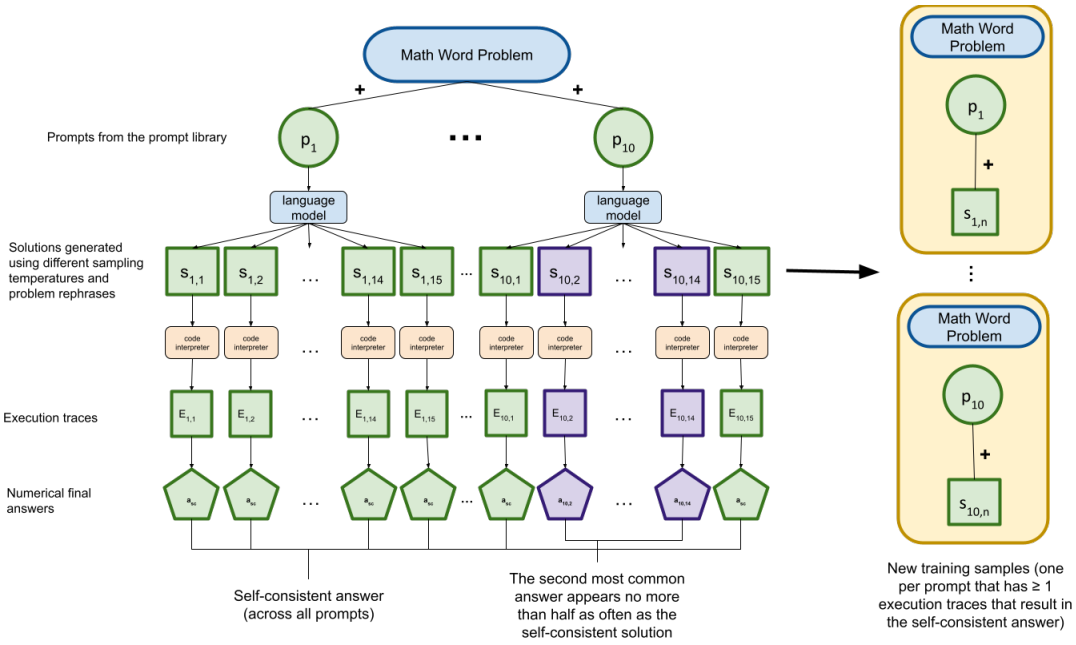

Figure 3: Execution Trace Sampling

4.2.3 Single-prompt sampling with rephrases and advanced filtering

To address the lack of improvement on hard problems discussed above, we created a sampling approach consisting of a combination of rephrase sampling and temperature sampling followed by a confidence filtering stage. This solution sampling method follows a similar principle to the one that `Phi-1.5` used to curate its synthetic training dataset (Gunasekar et al., 2023). The goal of this approach is to generate higher quality training samples for our unlabeled training set, where we are more certain that the solution provided by the model to the problem is correct.

To achieve higher quality solutions, we compute an Execution Trace (ET), as illustrated in Figure 6. An ET is a list of intermediate results of the Python code that solves the problem. We found that if the model can arrive at the same answer by generating code with several distinct ETs, the answer is more likely to be correct. Using ETs, we split our training set into high confidence (HC) and low confidence (LC) datasets. A problem belongs to the HC set if the number of solutions resulting in a self-consistent answer exceeds the number of solutions resulting in any other answer by a margin of 2, and if the number of distinct ETs for the self consistent solution exceeds the number of distinct ETs for the next most self consistent solution by a margin of 2. Subsequently, we refer to this process of filtering using ETs as advanced filtering.

Further, we slightly tweak the training procedure. At every iteration we only train on the HC set. However, note that as training progresses, the model becomes more confident and accurate about problems from the original LC set. Hence, at every iteration, the HC set grows as the model arrives at high-confidence solutions to problems from the LC set. i.e. at every training iteration, as the model self-improves, the HC set grows and the LC set shrinks.

This approach is illustrated in Figure 3. Concretely, let D be the given dataset consisting of different problems, P be the given set of prompts and M be the given model. Then the single-prompt sampling with rephrases and filtering works as follows:

1. We obtain 5 low-temperature rephrases for the problems in D using a separate fine-tuned rephrase-model, described in Section 4.3. Additionally, we obtain another 5 high-temperature rephrases for the problems in D using the rephrase-model.
2. We then sample 15 solutions for every (problem, prompt) tuple using model M. 5 of those solutions are obtained by solving a low-temperature rephrase of the problem and 5 of them

solving a high-temperature rephrase of the problem. The remaining 5 are obtained by solving the original problem with with varying temperatures like we did in Section 4.2.2.

3. We now have $15 \cdot |P|$ solutions at hand for every problem (15 solutions per prompt and $|P|$ prompts per problem). With them, we compute the ETs of each solution and the self-consistent solution for every problem. Then, using the procedure described above, we filter out our dataset D to the high confidence (HC) dataset.
4. Then, for every (problem, prompt) in the HC dataset, we randomly sample one of the solutions s_{sc} out of the self-consistent solutions obtained by that prompt (which is a subset of the 15 solutions it originally obtained). This yields at most one single triplet (problem, p_{sc}, s_{sc}) for any given (problem, prompt), and up to $|P|$ triplets (problem, p_{sc}, s_{sc}) for any problem in $|D|$.
5. Doing step 4 for all the problems in the HC dataset results in a dataset consisting of multiple high-confidence (problem, prompt, solution) triplets, which are then used for training.

4.3 Rephrase-model

To increase the diversity of the answers provided by our model during the solution sampling stage, we augment problems by rephrasing them. In our case, we rephrase problems by prompting the model with "Rephrase the following problem:" followed by the problem. Unfortunately, `Phi-2` fails at rephrasing problems. We observed that this model commonly just proceeds to solve the problems instead of providing a rephrase. To solve this, we finetune a separate version of the model, which we refer to as rephrase-model, for the rephrasing of problems. Due to the lack of rephrasing datasets of `GSM8K`, we create our own dataset, consisting of $3,000$ `GSM8K` problems, each with 3 highly-diverse rephrases. We generated the rephrasing dataset by sampling rephrases from gpt3.5-turbo from the OpenAI API (Brown et al., 2020).

5 Experiments

5.1 Data

The primary dataset we use is `GSM8K`. This is used for the baseline evaluation of our primary task, math problem solving. In addition, we use problems from the `GSM8K` train split (without ground truth solutions or answers) for self-improvement. We also use the datasets `HellaSwag`, `Piqa`, `Boolq`, and `Winogrande` for evaluation to ensure that training does not degrade performance on out-of-domain tasks (Zellers et al., 2019; Bisk et al., 2019; Clark et al., 2019; Sakaguchi et al., 2019).

5.2 Evaluation method

We evaluate our model based on the accuracy obtained at the task at hand. For the `GSM8K` dataset, we compare the final numerical answer obtained by our model with the original label (note, evaluation is the only stage where we access ground-truth labels). Concretely, we measure 3 types of accuracy to assess our model's performance with respect to the `GSM8K` dataset.

1. Mean `PASS@1` accuracy: the average accuracy of solutions generated across all prompts for all problems in the dataset.
2. Self-consistency accuracy (SC Acc.): we select the self-consistent answer (the numerical answer that is produced most often) as the only answer to a problem, and calculate accuracy across all problems in the dataset based on that.
3. `PASS@10`: the fraction of problems with at least one correct solution out of 10 solutions provided for that problem

In addition, we used 2 more measurements to evaluate the single-prompt sampling with rephrases and advanced filtering method. Namely, we use the advanced filtering accuracy (Adv Filter Acc.), which refers to the accuracy obtained by the model on the HC filtered training dataset. We also report the number of problems solved with high confidence. Finally, we also provide the accuracy on `HellaSwag`, `Piqa`, `Boolq`, and `Winogrande`. We leverage lm-evaluation-harness to run the evaluation for these datasets (Gao et al., 2023).

5.3 Experimental details

To test our methods, we train Microsoft's Phi-2 using Low-Rank Adaptation (LoRA) with the supervised finetuning trainer from HuggingFace. Namely, at each training iteration, the model was trained for 1 epoch with batch size of 1, gradient accumulation steps 8, learning rate 2×10^{-4} using the Adamw optimizer. We found that increasing the dropout from 0.05 to 0.1 results in a better generalization.

For the solution sampling procedure, we use VLLM (Kwon et al., 2023) , which permits us to sample approximately 400 problem solutions per minute on a single A5000 GPU. As mentioned in Section 4.2, we make use of a varied set of prompts during the sampling and training stages. Prompts are shown in Appendix E. Each prompt inserts the problem as a doc string into a header of a Python function, then the model generates the solution. We then run the solution through our multiprocessing python code interpreter with execution traces to obtain the final numerical answer.

5.4 Results

Table 1 shows results of training Phi-2 using each of the 3 sampling techniques and the GSM8K dataset. In addition, we show the accuracy on the low confidence LC set (consisting of hard problems) obtained by these approaches in Table 2. We further analyze our results in Section 6.

	Mean PASS@1	SC Acc.	PASS@10
Base Model Phi-2	66%	76%	88%
Cross-prompt training	72%	75%	82%
Single-prompt training	71%	78%	88%
Advanced filtering with rephrases	**72%**	**78%**	**88%**

Table 1: Results on GSM8K's test set

	Mean PASS@1	SC Acc.	PASS@10
Base Model Phi-2	36.0%	47.4%	**76.2%**
Cross-prompt training	40.9%	44.4%	61.3%
Single-prompt training	38.2%	43.4%	67.2%
Advanced filtering with rephrases	**41.7%**	**50.6%**	74.6%

Table 2: Results on the initial LC set, containing problems Phi-2 finds hard to solve.

	Mean PASS@1	SC Acc.	PASS@150	Adv Filter Acc.	Num probs solved with HC
Iter1	36%	47%	75%	77%	438
Iter2	41%	50%	75%	78%	538
Iter3	42%	51%	75%	79%	589

Table 3: Self improvement on the LC set using single-prompt sampling with rephrases and filtering

5.4.1 Rephrase-Phi

Finetuning Rephrase-Phi resulted in a network which was able to correctly provide diverse rephrases of problems, while preserving their meaning. We leave examples of rephrases of both Rephrase-Phi and Phi-2 for the same problem in D.1. Notably, while building the dataset using OpenAI's API, we qualitatively noticed a lack of diversity in rephrases using the default parameters of the API. Hence, the key to building a diverse rephrasing dataset was to add a penalization term to tokens that had occurred in previous rephrases. This penalization significantly minimized repetition and yielded a set of highly diverse rephrases for each of the problems, which we further trained Rephrase-Phi on.

6 Analysis

In this work, we explored the capabilities of self-improvement in SLMs. Concretely, we experimented with Phi-2 and the GSM8K dataset. We carried out our experiments by executing our self-learning

framework (shown in Section 4.1) using different solution sampling techniques, which are key to guiding the self-learning process of the model.

Cross-prompt sampling: We first noticed that using cross-prompt sampling limited self-improvement to 1-2 self iterations of training. This was caused by a diversity collapse in the solutions provided by the model, as can be observed in Table 1, where the `PASS@10` accuracy plummeted in just 2 iterations. Having a low `PASS@10` accuracy meant the model was generating very similar solutions for all prompts, which halted learning as the model got stuck at predicting the same wrong solutions for the problems it failed at.

Single-prompt sampling: We then observed that solution diversity was preserved if we used single-prompt sampling during training. Looking at Table 1, we can notice that the `PASS@10` accuracy is preserved even after 3 iterations. We think this is the case since diversity is preserved across prompts when doing single-prompt sampling, as prompts are not capable of observing solutions other prompts came up with. However, this solution sampling approach had limited performance growth as, after 3 iterations, it stopped improving in accuracy, effectively getting stuck at hard problems.

Single-prompt sampling with rephrases and filtering: We noticed that enhanced diversity through rephrasing problems combined with filtering out low-confidence solutions allowed our model to progressively improve on the hard-to-solve problems. In Table 3, we observe how the self-consistent solution accuracy on the hard problems remains close to 50% throughout training. This indicates that matching the hard problems with solutions obtained through simple self-consistency provides a poor training signal to the model on what constitutes a correct solution. However, after applying our advanced filtering technique to the hard problems, the accuracy on the obtained high confidence solution set is close to 79%. Evidently, training on this filtered, high-quality solution set provides much richer information to the model about correct solutions, which in turn allows it to self-improve on the hard problems.

Notably, we trained using single-prompt sampling with rephrases and filtering for only 3 iterations (due to the lack of resources and time) but didn't observe convergence during training. This approach also demonstrated to be the best for training the model to solve the hard problems that the base `Phi-2` model initially struggled with, as seen in Table 2. We attribute this to the capacity of this technique to produce very diverse solutions and to progressively increase the difficulty of the problems the model learned from. The substantial diversity of solutions prevented the model from becoming fixated on inferior solutions when tackling difficult problems. The filtering stage enabled the model to train only on a progressively growing and more accurate set of high-confidence solutions. This set of high confidence solutions grew as the model self-improved and became capable of confidently solving problems it previously failed at.

7 Conclusion

We achieve a 6% mean increase in the performance of the `Phi-2` model on the `GSM8K` dataset using unlabeled-data and self-improvement. For harder problems, we also found that the margin of self-improvement depended on the diversity of the data the model self-improved on. To aid this, we created an advanced filtering technique that let us select diverse, high-quality solutions for the model to self-improve on. Additionally, we finetuned `Phi-2` to create `Rephrase-Phi`, a model that generated linguistically-diverse rephrasings for problems that we used in training.

In our work, we often randomly selected a solution given multiple self-consistent ones. In the future, we plan to utilize the model itself to pairwise evaluate solutions to select higher quality samples for training. Additionally, a limitation of our work has been our suspicion that `Phi-2` was exposed to the `GSM8K` dataset during training. While we were able to show a significant self-improvement despite this suspicion, in the future, we will be testing our methods on other SLMs. Additionally, we had to curtail training for the single-prompt sampling with rephrases and filtering method due to resource constraints. We did not notice convergence in training, and believe the model will continue to improve over the next few iterations. We plan on continuing to train using this method as well. Given our findings about the importance of diversity in self-improvement, and the potential of multiple iterations of self-improvement, we would recommend research into more ways of diversifying self-sampled data, and more experiments into the self-improvement of small language models.

References

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. 2019. Piqa: Reasoning about physical commonsense in natural language.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2022. Palm: Scaling language modeling with pathways.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions.

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.

Ronen Eldan and Yuanzhi Li. 2023. Tinystories: How small can language models be and still speak coherent english?

Leo Gao, Jonathan Tow, Baber Abbasi, Stella Biderman, Sid Black, Anthony DiPofi, Charles Foster, Laurence Golding, Jeffrey Hsu, Alain Le Noac'h, Haonan Li, Kyle McDonell, Niklas Muennighoff, Chris Ociepa, Jason Phang, Laria Reynolds, Hailey Schoelkopf, Aviya Skowron, Lintang Sutawika, Eric Tang, Anish Thite, Ben Wang, Kevin Wang, and Andy Zou. 2023. A framework for few-shot language model evaluation.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks are all you need.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve.

Mojan Javaheripi and Sébastien Bubeck. 2023. Phi-2: The surprising power of small language models.

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

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.

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

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Oleksandr Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. 2023. Learning math reasoning from self-sampled correct and partially-correct solutions.

Sundar Pichai. 2023. Introducing gemini: Our largest and most capable ai model.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence?

A Figures

Figure 4 shows our training framework.

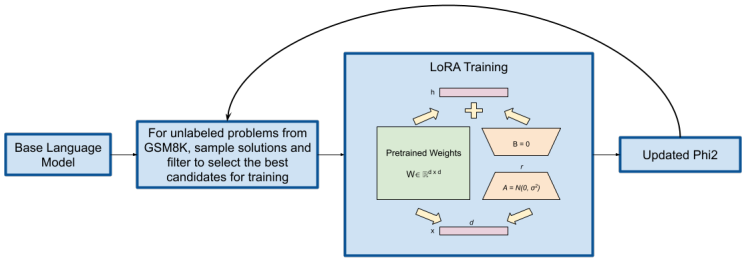

Figure 4: Self Improvement Process

B Result Tables

	HellaSwag	Piqa	Boolq	Winogrande
Base Model Phi-2	47.7%	78.8%	82.6%	75.4%
Cross-prompt sampling	47.7%	79.1%	82.4%	76.1%
Single-prompt sampling	47.8%	79.1%	82.6%	76.5%

Table 4: Results on Out-of-Domain Tasks

	Mean Acc.	Maj. Voting Acc.	PASS@10	ET Acc.	N LC Solved
Iter1	32%	37%	61%	81%	435
Iter2	33%	40%	61%	80%	475
Iter3	34%	40%	62%	83%	525

Table 5: Self improvement on LC problem set for rephrase sampling

	Mean Acc.	Maj. Voting Acc.	PASS@10	ET Acc.	N LC Solved
Iter1	26%	32%	54%	74%	327
Iter2	28%	33%	52%	77%	395
Iter3	28%	34%	53%	75%	420

Table 6: Self improvement on LC problem set for high-temperature rephrase sampling

C Models

Our self-improved models are available on Hugging Face under the MIT license:

1. Model trained using Cross-prompt sampling, 2 iterations:
`https://huggingface.co/deepbrain/phi2-gsm8k-cross-prompt-training`

2. Model trained using Single-prompt sampling, 3 iterations:
`https://huggingface.co/deepbrain/phi2-gsm8k-single-prompt-temperature-training`

3. Model trained using Single-prompt sampling with rephrase and filtering, 3 iterations:
`https://huggingface.co/deepbrain/phi2-gsm8k-rephrase-high-confidence-training`

D Examples

D.1 Rephrase-Phi v.s. Phi2

In Figure 5, we see how `Phi2`'s and `Rephrase-Phi`'s performances compare on the problem rephrasing task.

Example problem:

During the holiday season, Rebecca purchased two pies, with each pie being divided into eight equal parts. After consuming one slice from each pie herself, Rebecca and her guests consumed 50% of the leftover portions over the weekend. Subsequently, on Sunday evening, Rebecca and her spouse indulged in one more slice each. What is the total count of slices that are still available?

Rephrase-Phi's rephrase:

Rebecca bought 2 pies for the holiday weekend. Each pie was sliced into 8 slices. Rebecca ate 1 slice of each pie. Her family and friends ate 50% of the remaining pies over the weekend. On Sunday evening Rebecca and her husband each had another slice of pie. How many slices are remaining?

Phi2's rephrase:

Solution: First, we need to calculate the total number of slices of pie Rebecca bought. Since each pie was sliced into 8 slices and she bought 2 pies, she bought a total of 8 * 2 = «8*2=16»16 slices.
Next, we need . . . Therefore, there are 5 slices of pie remaining.

Figure 5: Example of a GSM8k problem, the rephrase given by Phi-Rephrase and the "rephrase" given by Phi-2, which is wrong as it's a solution instead of a rephrase. Note that both networks were prompted with "Rephrase the following problem:" followed by the problem.

D.2 Execution Traces

In Figure 6, we see an example of three execution traces. We consider two execution traces to be identical if they have the same number of lines of code, and if the resulting value on each line is the same between them.

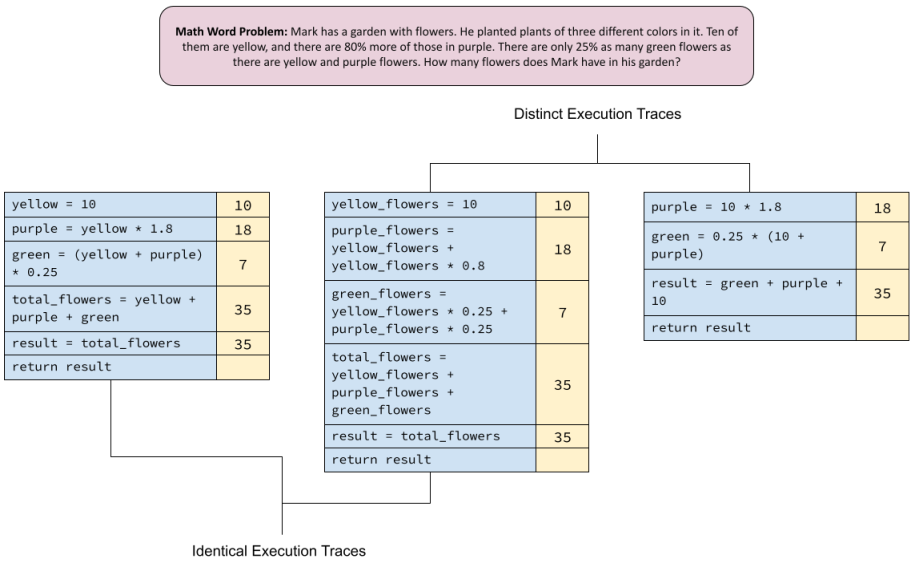

Figure 6: Example of an Execution Trace

E Prompts

Below, we have listed the ten prompts we used throughout our training and evaluation process. %s represents where the problem is inserted.

```
def problem() -> int:
"""%s
    Elaborate your thinking step by step in comments before each code
    line below
"""
def problem() -> int:
    """%s
       Add comments before each line
    """
def problem() -> int:
    """%s
       Be accurate and think step by step in comments before each code
       line below
    """
def problem() -> int:
    """%s
       Find unusual solution and comment before each of your line of
       code
    """
def problem() -> int:
    """%s
       In your comments write an algebraic formula based on the problem,
       solve it algebraically, then write code to calculate the result
    """
def problem() -> int:
    """%s
       Find the most elegant and correct solution
    """
def problem() -> int:
    """%s
       Think step by step in comments before each code line below
    """
def problem() -> int:
    """%s
       You must elaborate your thinking in comments below
    """

def problem() -> int:
    """%s
       Is this a simple math or algebra problem? For algebra problems, you
       must elaborate and solve it algebraically in the comments first,
       then write code to calculate the result. For simple math problems,
       you can write code to calculate the result directly
    """
def problem() -> int:
    """%s
       First, let's solve this problem using pure symbolic expressions.
       laborate with your algebraic skills below. Use x,y,z...to denote
       unknown variables. Use a,b,c... to denote given constants. Then
       write a python code to compute the result
    """
```