A Reinforcement Learning Technique for Large Language Model on Self-Verifiable Problems

Guanglei He
Tsinghua University
hgl22@mails.tsinghua.edu.cn

Yingxin Li
Tsinghua University
li-yx22@mails.tsinghua.edu.cn

Jiuyang Zhou
Central Conservatory of Music
24aw22@mail.ccom.edu.cn

Abstract

For certain types of self-verifiable problems, such as those encountered in programming competitions, game theory, and mathematical domains, an intriguing question arises: Can a large language model, guided solely by iterative attempts and performance feedback rather than human prior knowledge, incrementally refine its solutions to ultimately surpass human-level problem solving capabilities? By continuously recording both successful and unsuccessful attempts and employing these historical records as reinforcement signals, is it possible to train a model through iterative refinement and reinforcement learning to achieve expertise well beyond that of human practitioners?

Building on this concept, we leverage the Codegeex4-9B model as our foundational large language model and apply a reinforcement learning framework to the self-verifiable domain of programming challenges, such as those found in NOI/ACM competitions. Our preliminary experiments show that employing a feedback-driven problem-solving strategy can improve solution success rates by approximately 5–10% points over random trial attempts. Subsequently, we further enhance the model's capabilities through Direct Preference Optimization (DPO)-based reinforcement learning on the recorded solution histories.

Although time constraints have limited the extent of our current data, we plan to release additional results in the coming days as we continue to refine and evaluate our system.

1 Introduction

With the rapid development of large language models (LLMs)(1), code-specific language models have garnered significant attention in the community. Built upon pre-trained LLMs, code LLMs such as the StarCoder series, CodeLlama series , DeepSeekCoder series , CodeQwen1.5 , and CodeStral , have demonstrated superior performance in coding evaluations (Chen et al., 2021; Austin et al., 2021; Cassano et al., 2022; Jain et al., 2024; Liu et al., 2024a; Li et al., 2024b; Guo et al., 2024b; Wu et al., 2024b). However, in comparison with the recently state-of-the-art proprietary LLMs, Claude-3.5-Sonnet (Anthropic, 2024) and GPT-4o (OpenAI, 2024), the code LLMs are still falling behind, either open-source or proprietary models.

Problems in programming, mathematics, and game theory possess the characteristic of self-verification. Although current Large Language models have demonstrated strong capabilities in solving these problems, there is still a long way to go to reach or even surpass human levels. Taking programming problems as an example, the current strongest Large Language models—GPT-4o, GPT-o1-preview, and GPT-o1—have programming competition capabilities of 11.0%, 62%, and

Submitted to the Workshop on Instruction Tuning and Instruction Following at NeurIPS 2023. Do not distribute.

89%, respectively (24). While GPT-o1's programming ability is quite outstanding, there remains a certain gap compared to the best human performers.

GPT-o1 significantly improved the model's reasoning ability by introducing chain-of-thought (CoT) reinforcement learning. However, its training process relies on a large amount of high-quality human-labeled data and artificially designed search and reasoning processes (25; 26). These methods essentially incorporate human prior knowledge, allowing the model's reasoning ability to improve rapidly. However, human prior knowledge may also limit the further evolution of the model's thinking ability, and the human cognitive ceiling could become a bottleneck for the model, hindering it from surpassing human levels.

Reviewing the development history of AI technology (27), we find that some effective methods are often simple and straightforward. AI primarily solves problems through learning and search. If we incorporate too much prior knowledge or tricks into the AI's learning and search process, it may complicate the problem and limit AI's continuous evolution. Therefore, if we allow Large Language models to search and learn independently, is it possible for them to surpass human capabilities?

2 Relation Work

In recent years, research on self-verifiable tasks has advanced rapidly. Such tasks are characterized by the ability to verify solutions through automated testing or clearly defined evaluation criteria, thereby creating a closed-loop environment in which large language models (LLMs) can receive direct feedback on their correctness and quality without relying on human prior knowledge. For example, the SELF-REFINE (29) method demonstrates that, in the absence of external supervision, a model can independently produce an initial answer, provide iterative feedback on that answer, and subsequently refine it multiple times. Through this iterative self-improvement process, the model performance is significantly improved in domains such as dialogue, mathematical reasoning, and code optimization, indicating that even without human annotations, internal feedback loops can effectively improve the quality of generation for a variety of tasks.

Furthermore, approaches like Oracle-Guided Program Selection (28) employ carefully crafted test cases to guide the model in choosing correct code solutions from multiple candidates, thereby improving accuracy and trustworthiness. However, these methods still rely on manually designed test cases or selection steps, which can limit the capacity of the model for more widespread and autonomous development.

Our approach seeks to advance beyond these limitations by further embracing the principle of self-reinforcement. Without relying on externally curated test cases or human intervention, our framework enables the model to continually explore and summarize knowledge from its own historical attempts. This process allows the model to autonomously discover effective solution strategies and optimize its performance. Such a fully closed-loop, self-reinforcing paradigm holds the promise of enabling models to iteratively enhance their problem-solving capabilities in tasks such as programming competitions and mathematical problem-solving, ultimately surpassing human expert-level performance.

3 Research Plan

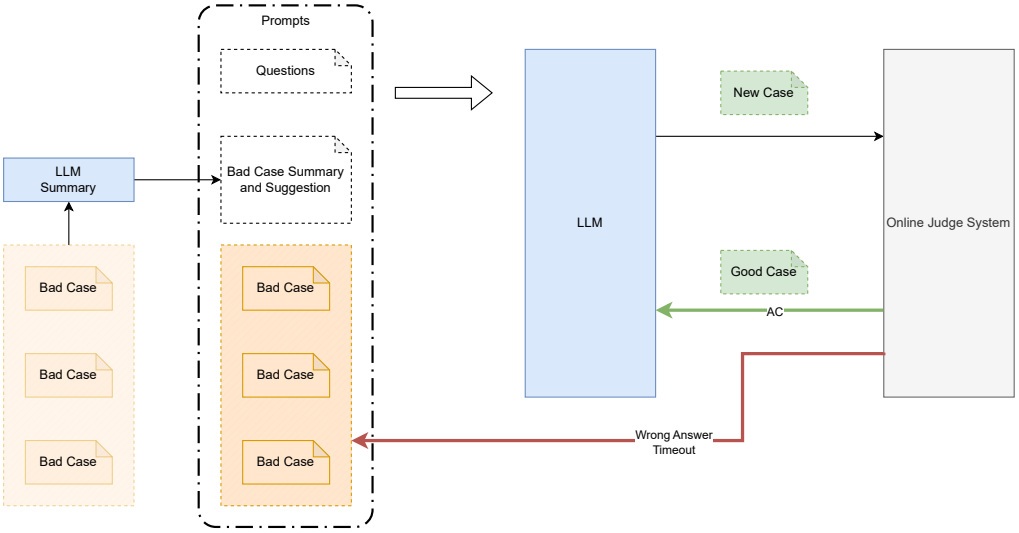
Based on the introduction considerations, we believe that for problems that can be self-verified, we can design a system that enables Large Language models to improve their capabilities through self-iteration. The core of the entire system is divided into two main parts:

Self-Feedback Problem Solving

We let the Large Language model solve programming problems of the NOI/ACM type, with the main steps as follows:

1. The model generates code based on the problem description and submits it to an online judge (OJ) system.
2. After execution, the system provides feedback on whether the problem is accepted (AC) or there are errors (Wrong Answer or Timeout).

Figure 1: Self-search system



3. Based on the result:
   - **Failure:** The model reflects and adjusts the code based on the feedback and historical records, then returns to step 1 to continue attempting. If it still cannot solve the problem after N attempts, it generates negative sample data.
   - **Success:** Ends the attempt and generates positive sample data.
4. When the historical record becomes too long, another model is used to compress and summarize it, providing more effective suggestions, ensuring that the model efficiently iterates and searches within limited input.

Table 1: Performance Comparison

Model	<Interview@any>	<Competition@any>
MapCoder APPS-150-cherrypicked (GPT-4)		**22**
GPT-J 6B (Finetuned)	13.15	13.51
code-davinci-002 175B(CodeT)	14.3	6.2
Codex 12B (Raw)	3.7	3.32
MoTCoder-15b	19.7	11.09
GPT-Neo 2.7B (Finetuned)	9.83	11.4
code-davinci-002 175B	**25.4**	14.5
GPT-Neo 2.7B	0.8	0
CodeChain+WizardCoder-15b	7.49	3.75
GPT2 1.5B (Finetuned)	9.27	8.8
WizardCoder-15b	4.21	0.81
AlphaCode 1B Filtered from 50000	9.66	7.75
CodeRL+CodeT5	14.33	15.7
AlphaCode 1B	5.24	7.06
CodeGeex4 9B	**29.63**	**22.63**

Iterative Improvement

1. In the above process, collect records of successful problem-solving after multiple attempts (positive samples) and records of multiple attempts still failing (negative samples).
2. Based on these data, fine-tune the model to reinforce its ability to solve programming problems.

3. After multiple iterations, the model's problem-solving ability is expected to continuously improve, ultimately reaching or even surpassing human levels.

The core keys of the system are:

- **No Introduction of Human Hints or Prior Knowledge:** All searches and reflections are completed by the model itself through iterative historical records. Humans do not provide any hints, ensuring that the model autonomously explores the optimal problem-solving strategies.

- **Iterative Reinforcement:** The key lies in utilizing the data generated during continuous attempts to fine-tune and reinforce the model, rather than relying solely on multiple attempts.

During the fine-tuning and iteration process, we believe that introducing GPT-o1's chain-of-thought technique into the problem-solving process is crucial for rapidly improving the model's reasoning ability. Therefore, when processing historical problem-solving data, we will focus on the model's thinking process, fine-tuning it, and encouraging it to engage in multi-step reasoning before providing results, further enhancing the model's reasoning capabilities.

Problem

You are a proficient competitive programmer.
You have attempted to solve the following problem:
Please write a sorting algorithm.

Input

- A single line containing $1 \leq n \leq 20$, the number of elements.
- A single line containing n integers, separated by spaces.

Output

- Print the sorted numbers on a single line, separated by spaces.

Example
Sample Input:

```
3
3 2 1
```

Sample Output:

```
1 2 3
```

Based on this, please write Python code to solve the problem. Ensure your code can execute with `__main__` as the entry point.

Accepted

```
def main():
  import sys
  input = sys.stdin.read
  d = input().split()

  n = int(d[0])
  ns = list(map(int, d[1:n+1]))
  ns.sort()
  print(" ".join(map(str, ns)))

if __name__ == "__main__":
    main()
```

Wrong Answer

```
score: -1
exitcode: 1 (execution failed)

Code output: Traceback
  (most recent call last):
  File "/tmp.py", line 1, in <sys>
    main()
  ...
  File "/tmp.py", line 6, in main
  for map in d[n]:
             ~~~~~~~~~~~~~~~
KeyError: 3
```

4 Methods

This research mainly consists of three parts. Firstly, based on a fixed dataset, a model is prompted continuously using the chain-of-thought (COT) technique with historical records. Through an Online Judge, a batch of correct and incorrect answers can be obtained. Then, these questions and correct answers are used for supervised fine-tuning (SFT), aligning the model to the capability of answering these correct answers. In the third stage, correct and incorrect answers are used for Direct Preference Optimization, further enhancing the model's answering ability. After one round of optimization, the model's capability will be higher than its initial correctness. This optimized model can be iteratively refined multiple times, leading to continuous improvement in the model's performance.

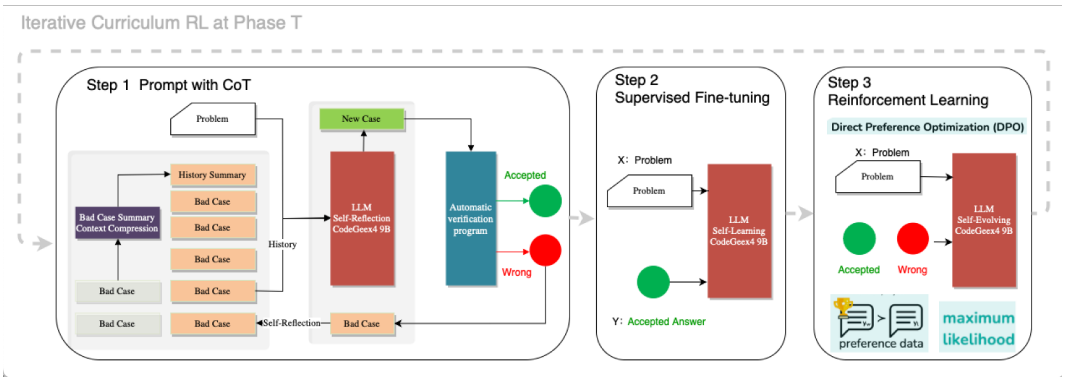


Figure 2: An overview of one phase, including prompt with CoT, Supervised Fine-Tuning and Reinforcement Learning. Such phase will iterate for multiple rounds which enable self-learning.

1. Application of Chain-of-Thought Technique:

- Referencing Codegeex4-9B's chain-of-thought, attempt to use historical data and rewards to conduct multiple rounds of prompting, stimulating the model's self-thinking and reasoning capabilities based on CoT to provide answers.
- Utilize an external Online Judge as a reward model, evaluating the model's answers and providing results such as Accepted, Wrong Answer, Time Limit Exceeded, and Memory Limit Exceeded.
- If the answer is correct, it is equivalent to collecting a correct sample. If the answer is incorrect, an incorrect sample is collected, and the negative reward is added to the historical data for the next round of prompting with CoT.

2. Supervised Fine-Tuning to Align Model Capabilities:

- Use the questions and correct answers as supervised data, and perform full-parameter Supervised Fine-Tuning on the model.
- Fine-tune until the loss converges. At this point, the model's capability should align with the first step, being able to answer questions correctly in one attempt that previously required multiple historical data prompts.

3. Multi-Round Iterative Reinforcement System:

- Use correct answers as preferred answers and incorrect answers and results as non-preferred answers, organizing them into corresponding data sets with the questions, and apply direct preference optimization to reinforce the model.
- Conduct multiple rounds of reinforcement learning until the loss converges. At this point, the model should perform better than after SFT.
- Return the model to the first step, using the first step's prompt with CoT method to solve the same dataset. The accuracy will be higher than the previous round. After multiple rounds, the model can become stronger based on self-rethinking and self-learning.

5 Results and Expected Outcomes

5.1 Results

In the experiment, the following key metrics are used to evaluate the effectiveness of one round of the SFT and DPO stages:

- **sft_loss_train:** The negative log-likelihood loss during the SFT stage. A smaller loss value indicates better model performance.
- **dpo_loss_train:** The training loss during the DPO stage.
- **rewards_train_accuracies:** Training accuracy.
- **rewards_train_margin:** The margin of the rewards.
- **logps_train_chosen:** The log probability of the chosen samples.
- **logps_train_rejected:** The log probability of the rejected samples.

The basic parameters of the first phase experiment are shown below:

Table 2: Basic parameters used in the experiment

Parameter	Value
Learning Rate	0.005
SFT Epoch	10
DPO Epoch	20
Batch Size	4
Question Nums	978
Chosen Nums	978
Rejected Nums	4623

The basic metrics of the current experiment are shown below:

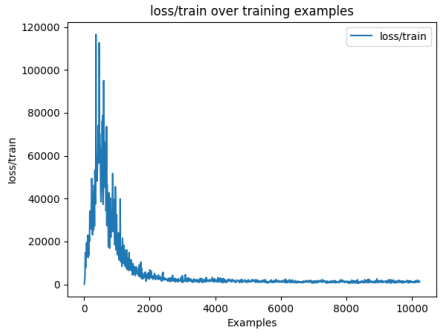


Figure 3: Loss of train in SFT

Figure 4: Loss of train in DPO

Figure 5: Some important metrics of phase 1

5.2 Expected Outcomes

It is expected that this method can achieve multi-round improvement effects similar to those demonstrated in WEBRL. In each phase, the model can align the ability after prompting with CoT so that it can solve the accepted question at pass@1, instead of attempting for so many times. Then it can enhance the ability to achieve higher success rate than pass@1. However, unlike WEBRL, which shows only one SFT result in the figure, our method includes multiple rounds of SFT and DPO results. The expectation is that each round will achieve a higher success rate than the previous one.

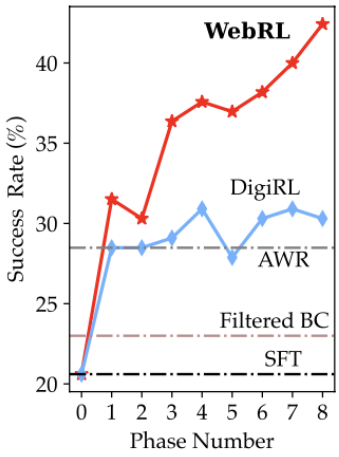


Figure 6: The enhancement of each phases of WEBRL. WEBRL is a self-evolving online curriculum reinforcement learn-ing framework for LLM-based web agents, yielding consistent continual improvements throughout the iterative self-evolution.

References

[1] Tom B Brown. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.

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

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

[4] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. arXiv preprint arXiv:2407.21783, 2024.

[5] AQ Jiang, A Sablayrolles, A Mensch, C Bamford, DS Chaplot, D de las Casas, F Bressand, G Lengyel, G Lample, L Saulnier, et al. Mistral 7b (2023). arXiv preprint arXiv:2310.06825, 2023.

[6] Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. Qwen technical report. arXiv preprint arXiv:2309.16609, 2023.

[7] An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. Qwen2 technical report. arXiv preprint arXiv:2407.10671, 2024.

[8] Anthropic. Claude 3.5 sonnet. https://www.anthropic.com/news/claude-3-5-sonnet, 2024. 2024.06.21.

[9] OpenAI. Gpt-4o. https://openai.com/index/hello-gpt-4o, 2024. 2024.05.13.

[10] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161, 2023.

[11] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. arXiv preprint arXiv:2402.19173, 2024.

[12] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. arXiv preprint arXiv:2308.12950, 2023.

[13] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. arXiv preprint arXiv:2401.14196, 2024a

[14] Qwen. Code with codeqwen1.5, April 2024. URL https://qwenlm.github.io/blog/ codeqwen1.5/.

[15] MistralAI. Codestral. https://mistral.ai/news/codestral, 2024. 2024.05.29.

[16] 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. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021.

[17] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. arXiv preprint arXiv:2108.07732, 2021.

[18] Federico Cassano, John Gouwar, Daniel Nguyen, Sydney Nguyen, Luna Phipps-Costin, Donald Pinckney, Ming-Ho Yee, Yangtian Zi, Carolyn Jane Anderson, Molly Q Feldman, et al. Multiple: A scalable and extensible approach to benchmarking neural code generation. arXiv preprint arXiv:2208.08227, 2022.

[19] Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. Livecodebench: Holistic and contamination free evaluation of large language models for code. arXiv preprint arXiv:2403.07974, 2024.

[20] Jiaheng Liu, Ken Deng, Congnan Liu, Jian Yang, Shukai Liu, He Zhu, Peng Zhao, Linzheng Chai, Yanan Wu, Ke Jin, et al. M2rc-eval: Massively multilingual repository-level code completion evaluation. arXiv preprint arXiv:2410.21157, 2024a.

[21] Ziming Li, Qianbo Zang, David Ma, Jiawei Guo, Tianyu Zheng, Xinyao Niu, Xiang Yue, Yue Wang, Jian Yang, Jiaheng Liu, et al. Autokaggle: A multi-agent framework for autonomous data science competitions. arXiv preprint arXiv:2410.20424, 2024b.

[22] Jiawei Guo, Ziming Li, Xueling Liu, Kaijing Ma, Tianyu Zheng, Zhouliang Yu, Ding Pan, Yizhi Li, Ruibo Liu, Yue Wang, et al. Codeeditorbench: Evaluating code editing capability of large language models. arXiv preprint arXiv:2404.03543, 2024b.

[23] Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xinrun Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, et al. Tablebench: A comprehensive and complex benchmark for table question answering. arXiv preprint arXiv:2408.09174, 2024b.

[24] OpenAI, "Learning to Reason with LLMs," Openai.com, 2024. https://openai.com/index/learning-to-reason-with-llms/

[25] T. R. McIntosh, T. Susnjak, T. Liu, P. Watters, and M. N. Halgamuge, "From Google Gemini to OpenAI Q* (Q-Star): A Survey of Reshaping the Generative Artificial Intelligence (AI) Research Landscape," arXiv.org, Dec. 17, 2023. https://arxiv.org/abs/2312.10868

[26] P. Putta et al., "Agent Q: Advanced Reasoning and Learning for Autonomous AI Agents," arXiv.org, 2024. https://arxiv.org/abs/2408.07199

[27] W. Hou and Z. Ji, "Comparing large language models and human programmers for generating programming code," arXiv.org, 2024. https://arxiv.org/abs/2403.00894 (accessed Oct. 08, 2024).

[28] Christakis M, Pradel M, Fan Z, Ruan H, Mechtaev S, Roychoudhury A (2024) Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2024. Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis:628–640. https://doi.org/10.1145/3650212.3680308

[29] Madaan A, Tandon N, Gupta P, Hallinan S, Gao L, Wiegreffe S, Alon U, Dziri N, Prabhumoye S, Yang Y, Gupta S, Majumder BP, Hermann K, Welleck S, Yazdanbakhsh A, Clark P (2023) Self-Refine: Iterative Refinement with Self-Feedback. arXiv. https://doi.org/10.48550/arxiv.2303.17651