Enhance Reasoning of Large Language Models via Macro-Micro Self-Training

Anonymous ACL submission

Abstract

Decomposing complex problems into smaller stages has proven to be highly effective in enhancing the reasoning capabilities of Large Language Models (LLMs). However, as the reasoning process becomes more intricate, uncertainties and errors tend to accumulate, making it challenging to achieve precise final outcomes. Overcoming this challenge and addressing uncertainty in multi-step reasoning necessitates innovative approaches. In this regard, we propose a novel macro-micro self-training method. Our approach leverages self-evaluation and self-modification to enable LLMs to continuously refine their outputs. Through self-evaluation, LLMs assess the accuracy of their generated outputs, while the critical aspect of self-modification allows for iterative refinement of these outputs. To ensure comprehensive refinement, we combine macro evaluation and modification of the entire code structure with micro analysis, where each line of code is individually assessed and refined in line with the problem statement. This dual approach ensures coherent handling of both syntax and semantics. Empirically, our results demonstrate the effectiveness of our approach, as it outperforms existing methods across all settings. Our method enables LLMs to achieve new levels of reasoning capability, providing superior performance in various tasks.

1 Introduction

Large Language Models (LLMs) (Radford et al., 2019; Brown et al., 2020; Chowdhery et al., 2022) have revolutionized Natural Language Processing (NLP), showcasing a broad spectrum of capabilities, such as text completion, translation, coding, intricate reasoning tasks (Zhang et al., 2021; Sivakumar and Moosavi, 2023; Mialon et al., 2023). Among them, the reasoning task, regarded as a representative task for evaluating LLM's intelligence, is widely studied. Specifically, research has delved into how LLMs reflect human-like content effects in common-sense reasoning, including abstract reasoning, understanding real-world knowledge (Dasgupta et al., 2023), coupling with logic programming (Yang et al., 2023), etc. Collectively, these studies highlight the evolving sophistication of LLMs in mimicking and potentially surpassing human-level reasoning in various contexts. However, mathematical reasoning remains a challenge for these models (Xu, 2023; Luo et al., 2023).

Various prompting approaches have been proposed to enhance the reasoning ability of LLMs. Chain-of-Thought (CoT) prompting has been a notable advancement in improving LLMs' mathematical reasoning capabilities, facilitating step-by-step problem-solving (Wei et al., 2022). Similarly, Least-to-Most prompting breaks down complex problems into simpler, more manageable subproblems (Zhou et al., 2022). Program of Thoughts (PoT) and Program-Aided Language models (PAL) represent further progress, combining neural LLMs with symbolic interpreters to enhance mathematical reasoning (Chen et al., 2022; Gao et al., 2022). However, these reasoning methods often make errors in various aspects, including logic organization and calculation details, which underscore the need for a robust self-enhancement mechanism. In this context, (Xie et al., 2023) showcases the potential of self-evaluation guided beam search as a means to navigate the vast reasoning space with improved accuracy. Additionally, (Jiao et al., 2023) utilizes self-supervision through in-context learning to enhance reasoning capabilities. Furthermore, (Gulcehre et al., 2023) adopts a reinforcement learning framework to facilitate self-training. However, these works can only solve step-by-step calculation errors and ignore errors in overall logic.

To overcome this challenge, we propose a novel methodology called Macro-Micro Self-Training (`M2ST`) to enhance the mathematical capabilities of LLMs. `M2ST` incorporates self-training at both the macro and micro levels, with the macro level

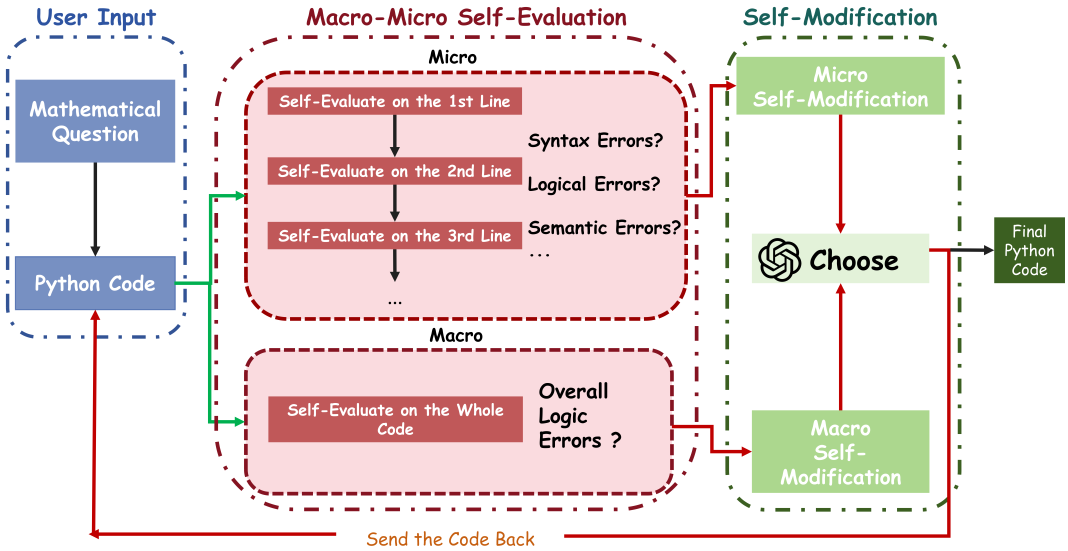

Figure 1: Overall Workflow of Macro-Micro Self-Training (M2SF).

prioritizing overall reasoning logic and the micro level concentrating on one-step calculation. The self-training process consists of two essential components: self-evaluation and self-modification. Through self-evaluation, LLMs can assess the accuracy of their generated outputs, while self-modification plays a critical role in iteratively refining the output. Figure 1 shows the detailed workflow of M2ST. Specifically, when inputting a math word question to LLM, we first get a Python code to solve it by PoT. Then M2ST will enhance this code from both macro and micro level. At the micro level, we utilize LLM for self-training purposes, enabling self-evaluation of individual lines to identify errors and subsequently self-modify them. Additionally, at the macro level, we employ LLM to evaluate the code as a whole, determining if any logical errors exist, and performing self-modifications if necessary. Moreover, in order to integrate the improved codes obtained from these two steps, we employ LLM itself to select the superior version. This selection process is accomplished through a zero-shot prompt approach inspired by (Kadavath et al., 2022). The self-training process will undergo multiple iterations until convergence is achieved in the training. Our approach has resulted in respectable improvements across various reasoning tasks. For instance, by implementing on Codex model (Chen et al., 2021), we achieve accuracies of 83.4%, 59.3%, and 89.8% on the GSM8K, AQuA, and SVAMP benchmarks, compared to the vanilla PoT reasoning-enhanced Codex performance of 71.6%, 47.3%, and 82.4%, respectively. Our further analysis on Llama-2 (Touvron et al., 2023) demonstrates the efficiency of our method in surpassing the self-training baseline under equivalent computational budgets.

2 Methods

2.1 Background: Reasoning via Code

Starting from CoT, reasoning in multi-steps is widely adopted for math word problems and common sense question-answering (Wei et al., 2023; Lyu et al., 2023; Yoran et al., 2023). Among them, PAL and PoT introduce solving math problems via code through prompting. Specifically, PoT utilizes LLMs, primarily Codex, to articulate reasoning steps that include both textual and programming language statements, culminating in an executable program. This program is then processed by an external interpreter, effectively decoupling the computational workload from the reasoning process. Such segregation allows PoT to circumvent the computational limitations of LLMs, leveraging the precision of program interpreters for mathematical evaluations and thereby enhancing the accuracy and efficiency of solving numerical reasoning tasks (Chen et al., 2023; Gao et al., 2023). In formal terms, when presented with a mathematical word problem P, PoT (Chen et al., 2023) utilizes prompts to guide the LLM in generating a Python code solution denoted as $C = LLM(P|\pi_{PoT})$. Here,

$C = \{C_1, \cdots, C_n\}$ represents the code, with each C_i representing a single line of code, and π_{PoT} represents the few-shots prompt of PoT.

Self-evaluation Mechanisms in LLMs The model's capacity to assess its own accuracy, utilizing metrics such as Expected Calibration Error (ECE), serves as the foundation for self-evaluation processes. In the rapidly evolving field of machine learning, LLMs have demonstrated inherent calibration properties, enabling them to introspectively evaluate their own outputs. This introspective capability proves particularly valuable in tasks involving multiple-choice questions, where calibration metrics like ECE effectively showcase the model's proficiency (Kadavath et al., 2022). The effectiveness of this mechanism depends not only on the model's architecture but also on the complexity of the task and the formulation of the prompt.

2.2 Macro-Micro Self-Training

M2ST integrates self-training at both the macro and micro levels, where the macro level emphasizes overall reasoning logic while the micro level focuses on one-step calculations. We proceed by providing a formal definition and explanation of these two self-training phases.

Macro Self-Training In macro self-training, our main focus lies in the comprehensive reasoning logic of the Python code. This encompasses understanding and rectifying errors such as miscalculating profits and improperly formulating equations to determine the number of bags sold. The subsequent example illustrates a specific flaw in the reasoning logic, where a correct equation is not established.

```python
# Math Problem: A grocer makes a 25% profit on the
# selling price for each bag of flour it sells.
# If he sells each bag for $100 and makes $3,000 in
# profit, how many bags did he sell?
# Question: How to solve the problem by Python code?
# Write Python Code to solve the math problem.

def calculate_bags_sold(selling_price, total_profit,
            profit_percentage):
    profit_per_bag = (profit_percentage / 100)
            * selling_price
    number_of_bags = total_profit / profit_per_bag
    return number_of_bags

selling_price = 100
total_profit = 3000
profit_percentage = 25

bags_sold = calculate_bags_sold(selling_price,
        total_profit, profit_percentage)
print(f"The number of bags sold is: {bags_sold}")
```

To evaluate the overall logic error, in t-th iteration of the training phase, where the Python code of the solution to the question is denoted as C_t, self-evaluation is achieved by

$$Macro\_SE = \text{LLM}(e_t | P, C_t, \pi_{MA-SE}), \quad (1)$$

where π_{MA-SE} is the self-evaluation prompt as shown in Table 1 and e_t is the evaluation at this iteration in the form of "Correct / Incorrect". For the aforementioned example, the format of e_t is as follows.

```python
# The Python code is: (B) Incorrect, because this
# problem requires establishing equation, but
# selling_price is 100 according to problem
# description and bag_sell should be unknown variable.
```

With the self-evaluation result, LLM is employed to achieve self-modification, denoted as

$$Macro\_SM = \text{LLM}(C_{t+1} | P, C_t, e_t, \pi_{MA-SM}), \quad (2)$$

where π_{MA-SM} is the self-modification prompt as shown in Table 1 and C_{t+1} is the new modified code. For the aforementioned example, C_{t+1} is as follows.

```python
# The Python code is: (B) Incorrect, because this
# problem requires establishing equation, but
# selling_price is 100 according to problem
# description and bag_sell should be unknown variable.
# Modify above Python code
def solution():
    bag_sell = Symbol('bag_sell', positive=True)
    selling_price = 100
    profit = 3000
    profit_percentage = 25
    solution = solve_it(bag_sell * selling_price
        * profit_percentage / 100 - profit, bag_sell)
    ans = solution[bag_sell]
    return ans
```

Micro Self-Training In micro self-training, our main focus lies on each calculation in the Python code. This includes understanding and rectifying errors, such as using incorrect numbers or performing incorrect calculations. The subsequent example illustrates a specific flaw in the incorrect calculation equation.

```python
# Math Problem: Meredith is a freelance blogger who
# writes about health topics and submits to
# clients each day as her permanent job. A blog
# article takes an average of 4 hours to research
# and write about. Last week, she wrote 5 articles
# on Monday and  2/5 times more articles on
# Tuesday than on Monday. On Wednesday, she wrote
# twice the number of articles she wrote on
# Tuesday. Calculate the total number of hours she
# spent writing articles in the three days.
# Question: How to answer the problem by Python code?
# Write Python Code to solve the math problem.
def solution():
    hours_to_write_one_article = 4
    num_articles_on_monday = 5
    num_articles_on_tuesday =
        num_articles_on_monday * 2 / 5
```

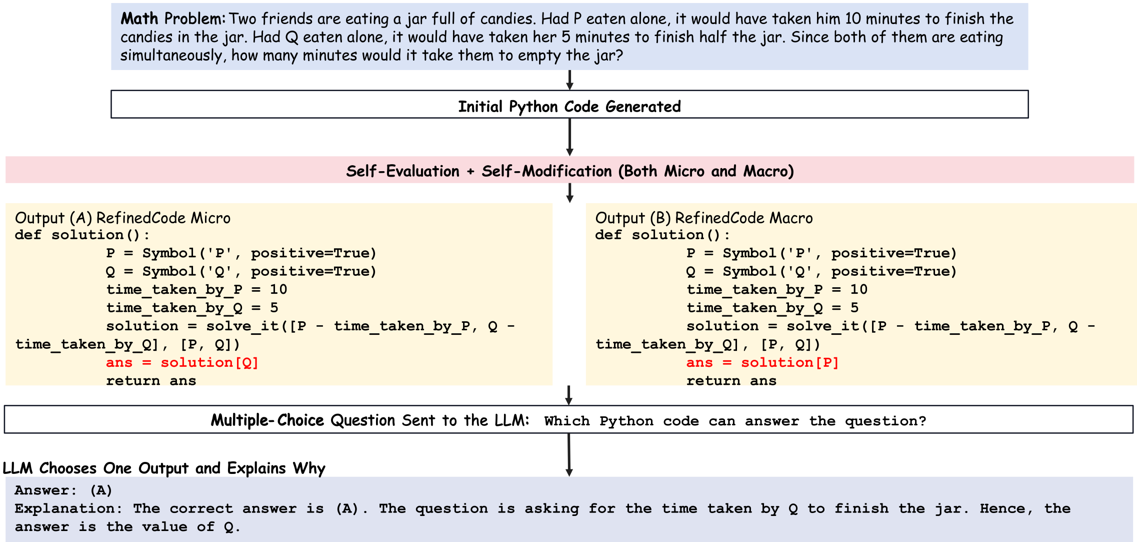

Figure 2: Illustration of the Combination Method.

```
    num_articles_on_wednesday =
        num_articles_on_tuesday * 2
    total_num_articles = num_articles_on_monday
        + num_articles_on_tuesday
        + num_articles_on_wednesday
    ans = total_num_articles
        * hours_to_write_one_article
    return ans
```

To evaluate the one-line calculation error, in t-th iteration of the training phase, where the Python code of the solution to the question is denoted as $C_t = \{C_1^{(t)}, \cdots, C_n^{(t)}\}$, where $C_i^{(t)}$ represents the i-th line code, self-evaluation is achieved by

$$Micro\_SE = \text{LLM}(e_i^{(t)} | P, C_t, \pi_{MI-SE}), \quad (3)$$

where π_{MI-SE} is the self-evaluation prompt as shown in Table 1 and $e_i^{(t)}$ is the evaluation at this iteration in the form of "Correct / Incorrect". For the aforementioned example, the format of e_t is.

```
# The calculation: num_articles_on_tuesday =
# num_articles_on_monday * 2 / 5 is
# (A) Correct
# (B) Incorrect
# The calculation is: (B) Incorrect, because
# num_articles_on_tuesday should be 2/5 more
# than num_articles_on_monday.
```

With the self-evaluation result, LLM is employed to achieve self-modification, denoted as

$$Micro\_SM = \text{LLM}(C_i^{(t+1)} | P, C_t, e_i^{(t)}, \pi_{MI-SM}), \quad (4)$$

where π_{MI-SM} is the self-modification prompt as shown in Table 1 and $C_i^{(t+1)}$ is the new modified code. For the aforementioned example, C_{t+1} is as follows.

```
# Fix the errors in above code
def solution():
    average_hour_to_write_article = 4
    num_articles_on_monday = 5
    num_articles_on_tuesday = num_articles_on_monday
        * 2 / 5 + num_articles_on_monday
    num_articles_on_wednesday
        = num_articles_on_tuesday * 2
    total_num_articles = num_articles_on_monday
        + num_articles_on_tuesday
        + num_articles_on_wednesday
    ans = total_num_articles
        * average_hour_to_write_article
    return ans
```

Combination With two answers modified at the macro and micro levels, it is necessary to propose a method for combining them into a single answer that represents the optimized result for this iteration. One intuitive approach is to merge these two steps sequentially, where the code is first passed through one stage and then pushed into another stage. However, experimental results have also confirmed that this approach leads to lower results due to the accumulation of errors at each stage. Therefore, we propose combining the two codes by selecting one with the assistance of LLM, which is done by

$$\text{LLM}(C_{t+1} | P, Macro\_SM, Micro\_SM, \pi_{Merge}), \quad (5)$$

where π_{Merge} represents the prompt used to combine two codes, as illustrated in Table 1. Furthermore, Figure 2 provides a concrete example for further clarification.

3 Experiments

3.1 Setup

Benchmarks. In our research, our primary objective was to enhance the reasoning abilities of LLMs

Task	Few-Shots Prompt
Micro Self-Evaluation	Math Problem: {question} # Python code, return ans {python code} The calculation: {one line code} is (A) Correct (B) Incorrect Answer: (A) Correct / (B) Incorrect, because {reason}
Micro Self-Modification	Math Problem: {question} # Python code, return ans {python code} # The {one line code} is incorrect, because {reason} # Modify above {one line code}
Macro Self-Evaluation	Math Problem: {question} # Python code, return ans {python code} Is the Python code (A) Correct (B) Incorrect Answer: (A) Correct / (B) Incorrect, because {reason}
Macro Self-Modification	Math Problem: {question} # Python code, return ans {python code} # The overall Python code is incorrect, because {reason} # Modify above Python code
Combine Macro and Micro	Math Problem: {question} (A) {python code} (B) {python code} Answer (A) / (B) Explanation: The correct answer is {correct answer} because

Table 1: Few-shots prompts for self-evaluation and self-modification in the micro and macro level.

specifically for solving math word problems. To accomplish this, we utilized a selection of diverse and challenging datasets, namely GSM8K (Cobbe et al., 2021), AQuA (Ling et al., 2017), SVAMP (Patel et al., 2021), and TabWMP (Lu et al., 2023). Each dataset possesses distinct characteristics and complexities, providing valuable opportunities to test and enhance various aspects of LLMs.

Baselines. We consider two types of baselines: (1) Chain-of-Thought (CoT) (Wei et al., 2022) prompting in free-text reasoning and (2) Program-Aided Language models (PAL) (Ling et al., 2017) and Program-of-Thought (PoT) (Chen et al., 2022) prompting in program-aided reasoning. In addition to these baselines, we employ self-training techniques, utilizing (4) Self-Evaluation Guided Beam Search (SEGBS) (Xie et al., 2023), which leverages self-evaluation as a signal during beam search, and (5) LogicLLM (Jiao et al., 2023), which employs self-supervision through in-context learning to enhance reasoning capabilities. To facilitate self-evaluation, we adopt a task formulation similar to multiple-choice question answering, following the approach outlined by Kadavath et al. (2022).

Prompt Detailed settings of few-shot prompts for self-evaluation and self-modification in the micro and macro level are shown in Table 1.

Backbone Models. We employed a thorough testing approach using both open-source and closed-source LLMs as the backbone models. For the open-source category, we utilized *Vicuna-13B v1.5* (Zheng et al., 2023), which is a chat model fine-tuned on the *LLaMa-2* (Touvron et al., 2023) framework. In the closed-source category, we employed *Code-Davinci-002* and *ChatGPT (gpt-3.5-turbo-0613)* (OpenAI, 2022), both built on the foundation of *GPT-3* (Brown et al., 2020).

Hyperparameters The only hyper-parameter requiring tuning in our approach is the number of training iterations. Since all four tasks we evaluate are reasoning tasks, we select the validation set of GSM8K to fine-tune this parameter for each model. Once optimized on the GSM8K validation set, we generalize the chosen number of training iterations across all tasks. Moreover, we select PoT as the foundational method for generating the initial Python code due to its superior stability and performance compared to PAL.

3.2 Main Results

Performance Across Different Methods and Models The empirical results outlined in Table 2 illustrates the performance enhancements brought about by the `M2ST` approach across various models. In the "code-davinci-002" model, `M2ST` sig-

Models	Method	GSM8K	AQuA	SVAMP	TabWMP
Code-Davinci-002	CoT	65.6	45.3	74.8	65.2
	PAL	72.0	-	79.4	-
	PoT	71.6	54.1	85.2	73.2
	SEGBS	80.2	55.9	89.6	79.1
	LogicLLM	76.2	47.3	82.4	69.7
	M2SF	**83.4**↑2.2	**59.3**↑3.4	**89.8**↑0.2	**81.9**↑2.8
Vicuna-13b v1.5	CoT	40.7	29.4	48.4	41.5
	PAL	49.1	-	53.1	-
	PoT	48.6	32.9	53.2	44.3
	SEGBS	52.3	33.2	56.8	46.2
	LogicLLM	45.2	30.4	50.6	42.7
	M2ST	**55.9**↑3.6	**34.2**↑2.0	**57.9**↑1.1	**49.4**↑3.2
ChatGPT	CoT	79.4	53.1	79.3	76.2
	PAL	81.6	-	85.8	-
	PoT	82.3	57.2	86.6	79.5
	SEGBS	84.3	59.5	88.4	82.7
	LogicLLM	80.7	55.9	83.3	78.4
	M2ST	**86.5**↑2.2	**62.6**↑3.1	**89.2**↑0.8	**84.8**↑2.1

Table 2: Main results of M2ST on reasoning tasks GSM8K, AQuA, SVAMP, TabWMP under Code-davinci-002, Vicuna-13b v1.5 and ChatGPT.

nificantly outperforms other methods, achieving remarkable improvements with an accuracy increase of 2.2% on GSM8K, 3.4% on AQuA, 0.2% on SVAMP, and 2.8% on TabWMP compared to the next best method, SEGBS. This underscores the efficacy of the M2ST methodology in refining LLMs' reasoning processes through its innovative evaluation and modification strategy.

Consistency Across Backbone Models Analysis of the M2SF method across different models, including "Vicuna-13b v1.5" and "ChatGPT", reveals consistent performance enhancement. For instance, within the Vicuna-13b v1.5 model, M2ST demonstrates significant accuracy improvements of 3.6% on GSM8K, 2.0% on AQuA, 1.1% on SVAMP, and 3.2% on TabWMP, over SEGBS. Similarly, in the ChatGPT model, M2ST leads with an accuracy increase of 2.2% on GSM8K, 3.1% on AQuA, 0.8% on SVAMP, and 2.1% on TabWMP. These results highlight the robustness and adaptability of the M2ST approach across various LLM architectures, marking a significant step forward in enhancing LLM reasoning capabilities.

3.3 Further Analysis

Micro and Macro Only. To demonstrate the effectiveness of combining macro and micro self-training, we conduct an ablation analysis by comparing its performance with that of single-phrase self-training. The detailed results of this analysis can be found in Table 3. In this comparison, we consider SEGBS as the baseline since it represents a special case of micro-only self-training without self-modification.

Our findings reveal that both micro-only and macro-only approaches consistently outperform PoT and PAL, providing evidence for the effectiveness of self-training. Specifically, in the case of code-davinci-002, micro-only achieves an average performance improvement of 3.1% across the four tasks compared to PoT, while macro-only demonstrates an average performance increase of 4.3% across the same tasks compared to PoT. Nevertheless, it is worth noting that both micro-only and macro-only approaches perform inferiorly to SEGBS. This outcome is expected since SEGBS utilizes beam search, which significantly expands the search space, allowing for more comprehensive exploration.

However, the performance of M2ST surpasses that of SEGBS, providing compelling evidence for the necessity and effectiveness of combining macro and micro self-training. Specifically, M2ST achieves an average improvement of 2.7% over the maximum performance between micro-only and macro-only self-training under code-davinci-002. Moreover, when considering the minimum performance between micro-only and macro-only self-training, M2ST exhibits an average improvement of 4.6%. These results further highlight the advantages of integrating macro and micro self-training within the M2ST framework.

Models	Method	GSM8K	AQuA	SVAMP	TabWMP
Code-Davinci-002	PAL	72.0	-	79.4	-
	PoT	71.6	54.1	85.2	73.2
	SEGBS	80.2	55.9	89.6	79.1
	Micro-Only	78.4↓1.8	52.6↓3.3	88.2↓1.4	77.8↓1.3
	Macro-Only	77.5↓2.7	56.5↑0.6	87.3↓2.3	79.4↑0.3
	`M2SF`	**83.4**↑2.2	**59.3**↑3.4	**89.8**↑0.2	**81.9**↑2.8
Vicuna-13b v1.5	PAL	49.1	-	53.1	-
	PoT	48.6	32.9	53.2	44.3
	SEGBS	52.3	33.2	56.8	46.2
	Micro-Only	51.2↓1.1	31.7↓1.5	55.6↓1.2	45.3↓0.9
	Macro-Only	50.8↓1.5	33.1↓0.1	54.5↓2.3	45.1↓1.1
	`M2ST`	**55.9**↑3.6	**34.2**↑2.0	**57.9**↑1.1	**49.4**↑3.2
ChatGPT	PAL	81.6	-	85.8	-
	PoT	82.3	57.2	86.6	79.5
	SEGBS	84.3	59.5	88.4	82.7
	Micro-Only	84.5↑0.2	58.9↓0.6	88.7↑0.3	82.1↓0.6
	Macro-Only	83.0↓1.3	57.3↓2.2	87.2↓1.2	82.3↓0.4
	`M2ST`	**86.5**↑2.2	**62.6**↑3.1	**89.2**↑0.8	**84.8**↑2.1

Table 3: Ablation analysis on micro-only and macro-only. PAL and PoT are listed to illustrate the effectiveness of self-training. ↑ represents accuracy increases compared to SEGBS while ↓ represents the opposite.

Macro and Micro One by One. Instead of selecting either macro or micro self-training with prompts, an intuitive method involves subjecting the Python code to both self-training methods sequentially. To validate the effectiveness of this selection approach over the sequential method, we perform a corresponding ablation analysis. This analysis aims to verify the validity of the selection process rather than the sequential application of the two self-training methods. Detailed results are shown in Table 4.

Our findings demonstrate that regardless of whether micro self-training is performed first or macro self-training is performed first, the sequential application of these two steps consistently yields inferior performance compared to performing both steps individually. Specifically, the micro-macro sequence leads to a performance decrease of 2.3%, while the macro-micro sequence results in a performance decrease of 2.7%. We attribute this decline in performance to the accumulation of errors at each step. Therefore, the selection approach, where one of the methods is chosen, avoids this problem and consistently delivers better performance.

Training Iteration. As the `M2ST` method is based on self-training, it is of utmost importance to establish the criteria for convergence. To accomplish this, we choose the validation set of GSM8K and fine-tune the convergence parameter for each model. After optimizing this parameter on the GSM8K validation set, we apply the selected number of training iterations to all tasks in a generalized manner. In order to examine the impact of this hyper-parameter on performance, we conduct a comprehensive ablation analysis, the results of which are depicted in the following Figure 3 and Figure 4. We choose the number of iterations in the range $[0, 10]$ and test on four reasoning tasks. We find that for the code-davinci-002 model, the optimal performance is achieved with 5 iterations, while for the ChatGPT model, the best performance occurs with 3 iterations. Beyond these optimal numbers, the performance starts to decline due to overfitting and the accumulation of errors. It is crucial to strike a balance between the number of iterations and performance to avoid such issues.

4 Related Work

Calibration and Self-Evaluation in Large Language Models LLMs have broad knowledge but face challenges with calibration—aligning predictions with actual outcomes—a critical concern in high-stakes fields like healthcare and finance (Ouyang et al., 2022). Studies indicate that even advanced LLMs struggle with calibration, emphasizing the need for effective solutions (Jiang et al., 2021; Liang et al., 2023). Research has been geared towards enhancing LLMs' self-assessment and calibration. (Kadavath et al., 2022) show that

Models	Method	GSM8K	AQuA	SVAMP	TabWMP
Code-Davinci-002	Micro-Only	78.4	52.6	88.2	77.8
	Macro-Only	77.5	56.5	87.3	79.4
	Micro-Macro	76.8↓1.6	51.3↓3.9	87.6↓0.6	76.5↓2.9
	Macro-Micro	75.3↓3.1	54.7↓1.8	86.1↓2.1	75.6↓3.8
	M2SF	**83.4**↑5.0	**59.3**↑2.8	**89.8**↑1.6	**81.9**↑2.5
Vicuna-13b v1.5	Micro-Only	51.2	31.7	55.6	45.3
	Macro-Only	50.8	33.1	54.5	45.1
	Micro-Macro	50.8↓0.4	30.3↓2.8	53.2↓2.4	44.7↓0.6
	Macro-Micro	50.3↓0.9	29.8↓3.3	53.4↓2.2	43.5↓1.6
	M2ST	**55.9**↑4.7	**34.2**↑1.1	**57.9**↑2.3	**49.4**↑4.1
ChatGPT	Micro-Only	84.5	58.9	88.7	82.1
	Macro-Only	83.0	57.3	87.2	82.3
	Micro-Macro	83.7↓0.8	56.7↓2.2	85.4↓3.3	81.6↓0.7
	Macro-Micro	82.3↓2.2	56.5↓2.4	86.1↓2.6	80.9↓1.4
	M2ST	**86.5**↑2.0	**62.6**↑3.7	**89.2**↑0.5	**84.8**↑2.5

Table 4: Ablation analysis on micro-macro and macro-micro. Micro-only and macro-only are listed for comparison. ↑ represents accuracy increases compared to the maximum between micro-only and macro-only while ↓ represents the opposite.

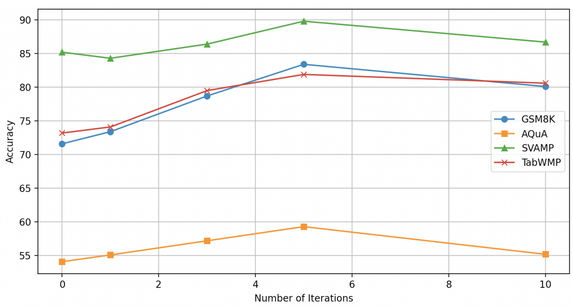

Figure 3: Accuracy of Code-Davinci-002 on four tasks with different numbers of iterations.

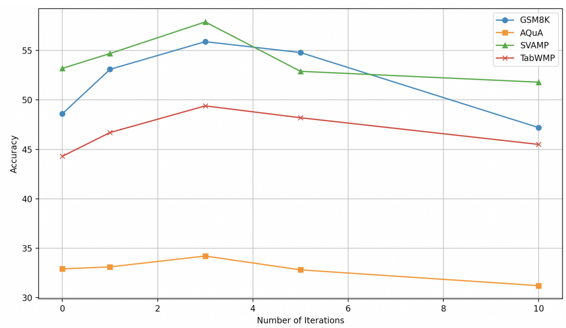

Figure 4: Accuracy of Vicuna-13b-v1.5 on two tasks with different numbers of iterations.

larger models can reliably evaluate their output across tasks. (Jain et al., 2023) introduce a self-supervised method for assessing LLM behavior on real-world data. (Zhu et al., 2023) explore how training influences model calibration, and (Zhao et al., 2023) propose a self-supervision framework for automatic LLM calibration and error correction, boosting LLM accuracy and reliability in sensitive applications without manual intervention.

Self Training Self-training, a semi-supervised learning paradigm, has been pivotal in advancing LLM capabilities. The key idea is to assign pseudo labels from a learned classifier to unlabeled data, and use these pseudo-labeled examples to further improve the original model training (He et al., 2020; Zhang et al., 2022). (Huang et al., 2022)'s study that demonstrated LLMs' ability to self-improve using only unlabeled datasets. In addition, the emergence of Reinforced Self-Training (ReST) showcases a novel stride in aligning LLMs with human preferences, particularly in the realm of language modeling (Gulcehre et al., 2023). Recent methodologies like Self-Instruct and CRITIC further enhance LLMs' autonomy in generating, critiquing, and refining outputs (Paul et al., 2024; Wang et al., 2022; Gou et al., 2023).

5 Conclusion

In conclusion, our proposed Macro-Micro Self-Training (M2ST) method marks a significant advancement in the domain of LLMs, particularly in enhancing their mathematical reasoning capabilities. By ingeniously integrating macro and micro levels of self-training, our methodology not only addresses errors in logic and calculation at their respective scales but also harmonizes them through a robust selection process, thereby mitigating the accumulation of inaccuracies. Empirical evaluations across diverse benchmarks and models underscore the superiority of M2ST over existing approaches, demonstrating its effectiveness in refining LLMs' reasoning processes for a variety of complex tasks.

Limitation

One limitation of the proposed method is its restricted applicability to scenarios that involve incremental reasoning, where problems can be broken down into smaller steps. This limits its usability in complex, non-linear problems that require holistic analysis or simultaneous consideration of multiple factors. Additionally, the method's sequential nature leads to a decrease in reasoning speed compared to parallel or concurrent reasoning approaches, making it less suitable for time-critical applications or situations that demand real-time decision-making. These limitations should be taken into account when considering the implementation of the method.

References

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks.

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.

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.

Ishita Dasgupta et al. 2023. Language models show human-like content effects on reasoning tasks. *arXiv preprint arXiv:2207.07051*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, Wolfgang Macherey, Arnaud Doucet, Orhan Firat, and Nando de Freitas. 2023. Reinforced self-training (rest) for language modeling.

Junxian He, Jiatao Gu, Jiajun Shen, and Marc'Aurelio Ranzato. 2020. Revisiting self-training for neural sequence generation.

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

Neel Jain, Khalid Saifullah, Yuxin Wen, John Kirchenbauer, Manli Shu, Aniruddha Saha, Micah Goldblum, Jonas Geiping, and Tom Goldstein. 2023. Bring your own data! self-supervised evaluation for large language models.

Zhengbao Jiang, Jun Araki, Haibo Ding, and Graham Neubig. 2021. How can we know when language models know? on the calibration of language models for question answering.

Fangkai Jiao, Zhiyang Teng, Shafiq Joty, Bosheng Ding, Aixin Sun, Zhengyuan Liu, and Nancy F Chen. 2023. Logicllm: Exploring self-supervised logic-enhanced training for large language models. *arXiv preprint arXiv:2305.13718*.

Saurav Kadavath, Tom Conerly, Amanda Askell, Tom Henighan, Dawn Drain, Ethan Perez, Nicholas Schiefer, Zac Hatfield-Dodds, Nova DasSarma, Eli Tran-Johnson, Scott Johnston, Sheer El Showk, Andy Jones, Nelson Elhage, Tristan Hume, Anna Chen, Yuntao Bai, Sam Bowman, Stanislav Fort, Deep Ganguli, Danny Hernandez, Josh Jacobson, Jackson Kernion, Shauna Kravec, Liane Lovitt, Kamal Ndousse, Catherine Olsson, Sam Ringer, Dario Amodei, Tom Brown, Jack Clark, Nicholas Joseph, Ben Mann, Sam McCandlish, Chris Olah, and Jared Kaplan. 2022. Language models (mostly) know what they know. *CoRR*, abs/2207.05221.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. 2023. Holistic evaluation of language models.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation : Learning to solve and explain algebraic word problems.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. 2023. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning.

H. Luo, Q. Sun, C. Xu, P. Zhao, J. Lou, C. Tao, X. Geng, Q. Lin, S. Chen, and D. Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct.

Qing Lyu, Shreya Havaldar, Adam Stein, Li Zhang, Delip Rao, Eric Wong, Marianna Apidianaki, and Chris Callison-Burch. 2023. Faithful chain-of-thought reasoning.

Grégoire Mialon, Roberto Dessì, Maria Lomeli, Christoforos Nalmpantis, Ram Pasunuru, Roberta Raileanu, Baptiste Rozière, Timo Schick, Jane Dwivedi-Yu, Asli Celikyilmaz, Edouard Grave, Yann LeCun, and Thomas Scialom. 2023. Augmented language models: a survey.

OpenAI. 2022. Introducing chatgpt.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems?

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2024. Refiner: Reasoning feedback on intermediate representations.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

J. A. Sivakumar and N. S. Moosavi. 2023. Fermat: An alternative to accuracy for numerical reasoning.

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*.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

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.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. Self-evaluation guided beam search for reasoning.

H. Xu. 2023. No train still gain. unleash mathematical reasoning of large language models with monte carlo tree search guided by energy function.

Zhun Yang et al. 2023. Coupling large language models with logic programming for robust and general reasoning from text. *arXiv preprint arXiv:2307.07696*.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering questions by meta-reasoning over multiple chains of thought.

Q. Zhang, L. Wang, S. Yu, S. Wang, Y. Wang, J. Jiang, and E.-P. Lim. 2021. Noahqa: Numerical reasoning with interpretable graph question answering dataset.

Shuai Zhang, Meng Wang, Sijia Liu, Pin-Yu Chen, and Jinjun Xiong. 2022. How does unlabeled data improve generalization in self-training? a one-hidden-layer theoretical analysis.

Theodore Zhao, Mu Wei, J. Samuel Preston, and Hoifung Poon. 2023. Automatic calibration and error correction for generative large language models via pareto optimal self-supervision.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

Chiwei Zhu, Benfeng Xu, Quan Wang, Yongdong Zhang, and Zhendong Mao. 2023. On the calibration of large language models and alignment.