# How Do Humans Write Code? Large Models Do It the Same Way Too

**Anonymous ACL submission**


## Abstract

Large Language Models (LLMs) often make errors when performing numerical calculations. In contrast to traditional chain-of-thought reasoning, the program-of-thoughts approach involves generating executable code to solve problems. By executing this code, it achieves more precise results. Using generated executable code instead of natural language can reduce computational errors. However, we observe that when LLMs solve mathematical problems using code, they tend to generate more incorrect reasoning than when using natural language. To address this issue, we propose Human-Think Language (HTL), a straightforward yet highly efficient approach inspired by human coding practices. The approach first generates problem-solving methods described in the natural language by the model, then converts them into code, mirroring the process where people think through the logic in natural language before writing it as code. Additionally, it utilizes the Proximal Policy Optimization (PPO) algorithm, enabling it to provide feedback to itself based on the correctness of mathematical answers, much like humans do. Finally, we introduce a focus-attention mechanism that masks the question segment, enhancing its reliance on natural language inference solutions during code generation. We conduct our experiments without introducing any additional information, and the results across five mathematical calculation datasets showcase the effectiveness of our approach. Notably, on the NumGLUE dataset, the LlaMA-2-7B-based model achieves a superior performance rate (75.1%) compared to the previous best performance with the LlaMA-2-70B model (74.4%).


## 1 Introduction

Mathematical reasoning problems are exercises presented as written or verbal scenarios, which require deciphering information, identifying relevant math concepts, and formulating equations to solve the problem, rather than straightforward equations in ARITHMETIC (Ahn et al., 2024). Mathematical reasoning problems represent a significant challenge for large models (Madaan et al., 2022; OpenAI et al., 2023).

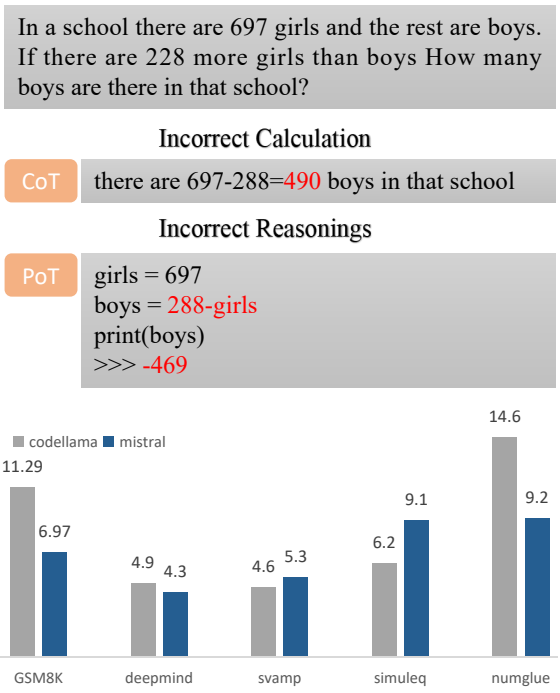


Figure 1: Above the figure is a simple example illustrating the drawbacks of CoT and PoT respectively, while below the figure is the investigation of LlaMA-base model and Mistral-base model using CoT correctly but PoT incorrectly across the five datasets.

This work focuses on mathematical reasoning within open-source models. In the realm of mathematical reasoning, the predominant challenges are **incorrect reasoning** and **incorrect calculations** (Lewkowycz et al., 2022). The chain-of-thought (CoT) methodology incrementally deduces steps, thereby yielding more robust reasoning outcomes (Wang et al., 2023b; Wei et al., 2022). But LLMs often make errors in numerical computations (Wei et al., 2023; Gao et al., 2023). (Chen et al., 2023) introduces program-of-thoughts (PoT), which di-

verges from generating natural language reasoning paths and instead focuses on producing executable Python code. PoT notably reduces calculation errors by delegating numerical computations to a Python executor (Chen et al., 2023). However, our findings indicate that while direct code generation improves computational precision, it also increases incorrect reasoning. We evaluate mathematical calculations using LlaMA-base model (Yue et al., 2023; Touvron et al., 2023; Rozière et al., 2023) and Mistral-base (Jiang et al., 2023) model. Inferences are generated separately using CoT and PoT, and are subsequently categorized based on correctness: instances where CoT is correct but PoT is not, and vice versa. The results illustrate in Figure 1.

The accuracy improvement of PoT compared to CoT is significant. However, the figure also clearly illustrates that in certain data points, PoT's accuracy lags behind that of CoT. The error rates are noteworthy, with the LlaMA-base model registering at 8.318%, and the Mistral-base model at 6.974%. We show some error examples in Figure 5. When manually examining the erroneous instances, we identify the primary sources of error as incorrect question comprehension and flawed reasoning. PoT consistently exhibits bias in question interpretation during code generation, leading to recurrent mistakes in variable initialization and problem-solving strategies. This limitation persists even with a 70B model. Within the scope of CodeLlaMA's pretraining data, which includes 500 billion code tokens, this constitutes a small fraction compared to the 2 trillion natural language tokens in the LlaMA-2 model. A noticeable gap in understanding mathematical problems between natural language and code in large language models still exists.

To address these discrepancies, we draw inspiration from human cognitive processes. When facing coding algorithm problems, humans typically first conceive a solution that can be described in natural language, and then translate it into code. Code, in this context, does not directly facilitate problem-solving but rather functions as an instrument for meticulous computation. In our presentation on the right side of Figure 2, we highlight the parallels between our integrated model and the human approach to solving mathematical problems. The concepts of CoT and PoT mirror the phases of human reasoning and code writing, respectively. We propose **Human-Think Language (HTL)**: a novel approach that merges the CoT and PoT methodologies. This method enables the model to initially generate answers using CoT, followed by the generation of PoT based on the CoT's inference steps. We incorporate the **error assessment function** based PPO (Schulman et al., 2017), leveraging the characteristics of obtaining answer correctness from mathematical questions (Luong et al., 2024). Meanwhile, we add **focus attention**, which precisely designates the information source during the generation of the k-th token. Focus-attention enhances PoT's capability to effectively harness CoT's reasoning steps.

In summary, our contributions are as follows:

(1) We propose Human-Think Language (HTL). We are the first to propose using the inference steps of CoT to enhance the reasoning ability of PoT. In our unique framework, we employ error assessment function which is a more precise reward function for reinforcement learning.

(2) We introduce focus-attention, a local attention that controls the generated information sources through mask settings.

(3) We evaluate our work on 5 mathematical reasoning datasets, and our results comprehensively outperform those of the baseline open-source models.

## 2 Method

In Human-Think Language (HTL), the model first generates problem-solving methods described in natural language by the model, then converts them into code, mirroring the common process where people think through the logic in language before writing it as code. We design two templates, tune-instruction and insert-instruction, to help the model adapt to new generation paths. To enable the model to learn the transition from natural language reasoning paths to programming languages, we design focus-attention for tune-instruction, which, during the training process to generate PoT, only allows the model to see information from CoT. After arriving at mathematical results, individuals can reinforce their math skills by checking if the answers are correct. In our model, the accuracy of the answers is also used as error assessment function to enhance the model's capabilities through the PPO algorithm.

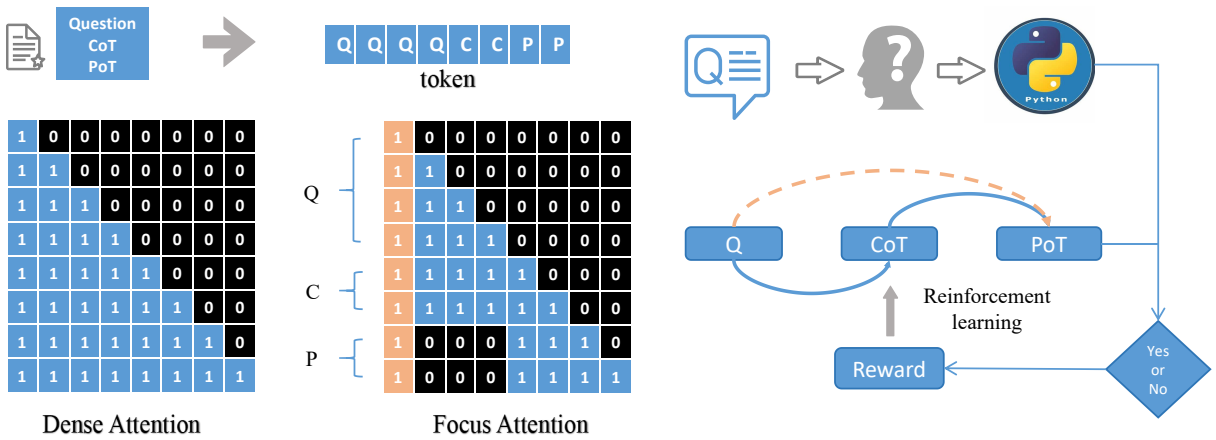


Figure 2: Dense Attention refers to traditional Attention, while focus attention is our approach. In the orange column on the left, the first four tokens share a consistent mask state of 1. On the right side of the figure, there is a comparison between human and LLMs in solving mathematical problems.

### 2.1 Instruction Construction

We design tune-instruction and insert-instruction. Tune-instruction instructs the model to generate CoT first and then PoT within a single stage, while insert-instruction takes the solution from CoT as input to the model and outputs the PoT answer. The templates show in the appendix B.

### 2.2 Focus Attention

**Attention Design** In our work, we use a local attention mechanism to control the mask matrix during training stage. Our method, as shown in Figure 2, divides the text of a mathematical problem into Q (question), C (CoT answer), and P (PoT answer). The objective in generating the PoT Answer is for the model to rely solely on information from the CoT Answer. However, local attention is limited in flexibility and cannot learn representations tailored to specific downstream tasks, as noted by (Beltagy et al., 2020). A recent study by (Xiao et al., 2023) introduced the concept of attention-sink, which forces a softmax on attention, leading to the initial tokens of a sequence containing a significant amount of information. Dependence only on a token's surroundings, as seen in window attention, restricts the model's effectiveness in downstream tasks. We utilize the attention-sink effect to overcome the limitations of local attention.

For the PoT information generation, the focus attention mechanism ensures the preservation of the initial tokens in the sequence. Echoing the findings of (Xiao et al., 2023), we include the first four tokens in the PoT information generation process.

**Training Strategy** While most local attention mechanisms are employed during the pre-training phase, as (Beltagy et al., 2020) suggests, we explore the usage of focus attention during the fine-tuning stage. It might lead to discrepancies in the information processed during inference.

The current formula for the causal mask matrix is as follows:

$$M_{ij} = \begin{cases} 1 & \text{if } j < i \text{ and } (j \in [0,3] \text{ or } j \in \text{CoT}) \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Ultimately, the contextualized representation $v_i^l \in \mathbb{R}^d$ of the $i$-th token at $l$-th attention layer can be formulated as:

$$A^l = \text{Softmax}\left(\frac{t^{l-1}W_Q^l \left(t^{l-1}W_K^l\right)^T}{\sqrt{d/N}} + M\right)$$
$$v^l = A^l\left(t^{l-1}W_V^l\right) \tag{2}$$

$W_Q$, $W_K$, and $W_V$ are intermediate matrix representations in the attention mechanism.

The conventional causal matrix is a lower triangular matrix, which leads to inconsistencies with downstream inference tasks.

To bridge this gap, we introduce an innovative training approach: during both the initial and final phases of training, we do not explicitly mask any tokens, thereby ensuring alignment with the inference stage. Throughout the training process, we incorporate a **mask coverage function**, which represents a quadratic equation and calculates the proportion of tokens being masked based on the

number of training steps:

$$\lambda_{masked} = \rho * min(1, -\alpha(\rho_{step} - \frac{1}{2})^2 + \beta) \quad (3)$$

$\lambda$ represents the total number of tokens for both the question and instruction, while $\rho_{step}$ denotes the current step's ratio in the total number of training steps. Then, we randomly select tokens from the question and instruct segments. The We provide the model with a buffer, allowing it to gradually learn local attention, and after training, we restore it to its autoregressive generation mode. The quadratic function accelerates its speed gradually both during ascent and descent, and at the peak, the derivative decreases, resulting in a longer duration of focused attention training.

### 2.3 Error Assessment Function Based PPO

In reinforcement-learning stage, the policy model which is the model after fine-tune stage learns by repeatedly sampling responses, evaluating the response's answer correctness, and updating its parameters. We employ PPO with a clipped objective algorithm for training. Following (Ziegler et al., 2020), the value model $V_\phi$ is constructed by appending a linear value head on top of the last hidden states of the policy model. For the reward model, we replace it with error assessment function. In non-terminal states, all action results are assigned a value of 0. At the terminal state, we use a reward function to score based on the correctness of the generated answer. Since we have access to both CoT and PoT, we can categorize the reasons for their errors and provide more fine-grained feedback scores for the model. The error assessment function is as follows:

$$f_r = \begin{cases} 1, & \text{CoT} = \boldsymbol{y}, \text{PoT} = \boldsymbol{y} \\ 0.6, & \text{CoT} \neq \boldsymbol{y}, \text{PoT} = \boldsymbol{y} \\ 0.3, & \text{CoT} = \boldsymbol{y}, \text{PoT} \neq \boldsymbol{y} \\ 0.1, & \text{CoT} = \text{null or PoT} = \text{null} \\ 0, & \text{CoT} = \text{null and PoT} \neq \text{null} \end{cases} \quad (4)$$

Regarding errors in PoT, we consider them to be inference errors and apply a significant penalty. Whereas, in cases where only CoT is incorrect while PoT is correct, we view it as solely a calculation error. Such a partial reward can help reduce the effect of learning from sparse reward. Furthermore, in line with (Zheng et al., 2023), our total reward encompasses both the reward function score and the Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) between the learned RL policy and the initial policy.

## 3 Experiment

### 3.1 Dataset

**Training Dataset** We select a subset of data from the training set of the MAmmoTH model. Then, we use the fine-tuned MAmmoTH model to generate CoT and PoT answers. We present the data to the model in the format of instruction tune. In other words, our experiments does not incorporate any additional information. In the end, we extract 36,000 examples from the training data of MAmmoTH, with 13,000 coming from GSM8K, 3,000 from NumGLUE, and 10,000 from Math (Hendrycks et al., 2021). In the work on training MAmmoTH, they take some extensions of these three datasets.

**Test Dataset** Our experiments test on five datasets: GSM8K (Cobbe et al., 2021), NumGLUE (Mishra et al., 2022), SimulEq (Koncel-Kedziorski et al., 2016), DeepMind (Saxton et al., 2019), and SVAMP (Patel et al., 2021). These five datasets encompass a wide range of mathematical computation problems, comprehensively reflecting the model's mathematical computational capabilities. Meanwhile, GSM8K, NumGLUE represent in-domain datasets, whereas SimulEq, DeepMind and SVAMP represent out-of-domain datasets, reflecting our model's mathematical transferability from two perspectives.

### 3.2 Experimental Setup and Baseline

Our work based on the MAmmoTH model, which achieves the state-of-the-art in open-source LLM mathematical reasoning by selecting existing datasets and generating a mixture of CoT and PoT for training (Yue et al., 2023).

We conduct experiments in MAmmoTH-Coder-7B (Yue et al., 2023) and MAmmoTH-Mistral-7B. In all experiments, the training is done using Deepspeed (Rasley et al., 2020) Zero stage3 and Huggingface Accelerate. The batch size is set to 32. The learning rates for LlaMA-base and Mistral-base are set at 5e-6 and 3e-7, respectively. The maximun length is set to 1024. For the PPO stage, we select 10% of the data for training using Lora over 40 epochs, with the KL coefficient set to 0.01. Our results compare the current models:

**Open-source Models** We compare existing open-source 7B LLMs. For LlaMA base: LlaMA-1, LlaMA-2 (Touvron et al., 2023), Code-LlaMA (Rozière et al., 2023) with PoT prompting. There

| Model | GSM8K | NumGLUE | SimulEq | DeepMind | SVAMP |
|---|---|---|---|---|---|
| Open-source 7B Parameter Model | | | | | |
| LlaMA-1 | 10.7 | 24.7 | 4.6 | 3.1 | 24.5 |
| LlaMA-2 | 14.6 | 29.9 | 6.0 | 7.2 | 34.5 |
| Code-LlaMA(PoT) | 25.2 | 26.8 | 3.5 | - | 49.4 |
| Galactica-6.7B | 10.2 | 25.8 | 4.2 | 17.1 | 25.6 |
| WizardMath | 54.9 | 36.1 | 12.8 | - | 36.1 |
| Mistral | 52.2 | 36.4 | 6.22 | 34.5 | 64.8 |
| Our Models(LlaMA-base) | | | | | |
| MAmmoTH-baseline(CoT) | 44.5 | 36.0 | 14.7 | 34.2 | 37.0 |
| HTL(CoT) | 44.1 | 36.2 | 15.03 | 33.8 | 36.9 |
| MAmmoTH-baseline(PoT) | 59.4 | 66.4 | 45.9 | 59.8 | 71.4 |
| HTL(Instruct) | 60.2 | 73.0 | 48.6 | 61.1 | 71.5 |
| HTL(two-stage) | 59.6 | 70.6 | 50.5 | 61.2 | 70.3 |
| HTL(Instruct+focus) | **62.4** | **74.1** | **50.8** | **63.1** | **72.3** |
| HTL(Instruct+focus+RL) | **65.7** | **75.1** | **50.9** | **62.8** | **74.4** |
| Our Models(Mistral-base) | | | | | |
| MAmmoTH-baseline | 75.0 | 73.9 | 50.3 | 61.1 | 80.6 |
| HTL(Instruct) | 74.7 | 76.3 | 51.6 | 62.9 | 81.2 |
| HTL(Instruct+focus) | **77.9** | **77.0** | **52.3** | **63.3** | **82.0** |
| HTL(Instruct+focus+RL) | **79.1** | **78.3** | **54.1** | **64.2** | **83.0** |

Table 1: HTL(Instruct) represents the results generated using only two specific instruction templates to create prompts, while "focus" indicates the inclusion of a focus attention mechanism. The "RL" is reinforcement learning.

are other models as well, such as Mistral (Jiang et al., 2023), which is an emerging model known for its outstanding performance in mathematical reasoning, WizardMath (Luo et al., 2023), a specialized model for mathematical inference, and Galactica (Taylor et al., 2022), which mainly to understand the performance of models specialized in STEM knowledge.

### 3.3 Main Result

The main result shows in Table 1. For closed-source models, the metrics on the GSM8K dataset for GPT-4 (OpenAI et al., 2023), GPT-4(Code Interpreter), PALM-2 (Anil et al., 2023), and Claude-2 (Bai et al., 2022) are respectively 92.0, 97.0, 80.7, and 85.2, far surpassing those of open-source models. Our results surpass other existing mathematical reasoning models on each dataset, achieving state-of-the-art (SOTA) across multiple datasets. Additionally, our approach demonstrates excellent performance in both in-domain and out-domain settings, evidencing its transferability. In our findings, the NumGLUE dataset showed a significant improvement, with an 11.5% increase. This aligns with the evaluations presented in Figure 1, where 14.5% of the data in the NumGLUE dataset is incorrectly solved by PoT but correctly by CoT. We extracte both the CoT outcomes and the subsequent CoT-then-PoT results generated by our model. The performance of our model's CoT answer is nearly identical to its original state. This outcome aligns with our expectations. We utilized the information from CoT to enhance the performance of PoT, while the generation of CoT should remain unchanged.

We evaluate our model using the methodology outlined in Figure 1 to determine how many data correctly predicted by CoT but our model wrong. The result shows in Figure 3. On models with different bases, the phenomenon of incorrect reasoning in PoT is significantly reduced.

**Ablation Study** We conducted ablation experiments on all datasets to investigate the contribution of each key component or strategy of our proposed method. The ablation tests include **w/o focus attention** and **w/o reinforcement learning**. We observe that focus attention and reinforcement learning significantly improve performance. Relying solely on instruction tuning leads to bottlenecks across the GSM8K, DeepMind, and SVAMP datasets.

**Two Stage** Inspired by the work of (Deng et al., 2023), it contrasts one-stage and two-stage approaches in the prompt guidance of LLMs. To explore which paradigm is more reasonable, we also conducted two-stage experiments. During train-

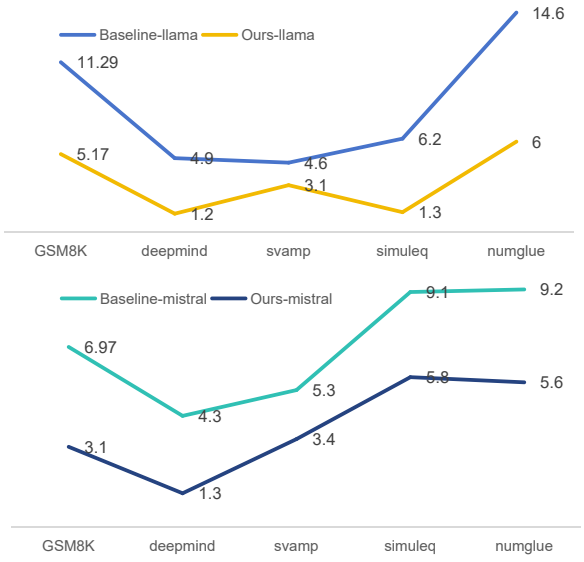


Figure 3: The upper line represents the proportion in the Baseline model where CoT is correct but PoT is wrong, while the lower line indicates the proportion in our model where CoT is correct but our answer is wrong.

ing, we use Insert tune for all data. We perform two generations to obtain the final results. In the first generation, we input the question to get the CoT answer. In the second generation, we input the question and the CoT answer to obtain the PoT answer. For different datasets, one-stage and two-stage approaches yield varying effects.

### 3.4 Influence of Instruction Length

| | G | N | Si | d | Sv |
|---|---|---|---|---|---|
| baseline | 59.4 | 66.4 | 45.9 | 54.8 | 71.4 |
| tune | 60.27 | 73.03 | 48.6 | 61.11 | 71.5 |
| padding | 59.14 | 63.36 | 44.16 | 52.13 | 69.9 |

Table 2: Padding represents a comparison experiment with the tune experiment, where the input sequence lengths are consistent but without introducing a comparison experiment with CoT answers.

In LLM generation, the length of the input sequence is a crucial factor that provides the model with more time for contemplation. Our experiments utilized a longer prompt template than before. In our test data, the length of most questions is only 20-30 tokens, while the length of our instructions has been extended to 80 tokens. We aim to verify whether the increased input length contributes to performance differences. We replacing all our newly added prompts with special characters such as comma, exclamation mark, and full stop. If all prompts and CoT answers replace with a large quantity of the same character, it impact the loss during training, hindering the model from learning information from other tokens. For instance, generating comma for all prompts could result in a satisfactory training loss. The experimental results show in Table 2.

We find that in mathematical reasoning tasks, simply adding meaningless characters to extend the model's thinking time is ineffective. This highlights the efficacy of the approach proposed by our method.

### 3.5 Influence of Subsets

Given training subsets with different distributions, this allows us to gain a clearer understanding of the importance of each section of data for the model. Meanwhile, we vary the quantity of different distributions to discover how much data volume enables the model to effectively assimilate knowledge. The result shows in Table 3. We discover an interesting phenomenon: when we use a specific dataset for downstream training, the model performs well on its corresponding test set but weakens its capabilities on other datasets. To allow for a clear comparison of our experimental results, we average the performance across the six datasets.

When we mix multiple datasets for training, the model's improvement in capabilities becomes more comprehensive. The combination of different datasets allows the model to focus more on the characteristics of mathematical problems rather than relying on specific patterns present in only one dataset.

### 3.6 The Effort of Mask Coverage Function

In order to explore how the Mask Coverage Function affects our performance, we conducte experiments in Table 4 by setting different function parameters. First, we test the effect of excluding the first four tokens to verify whether the attention sink has a significant impact. Remove the first four tokens, causing the model to lose a significant amount of information, making it quite challenging to make accurate predictions. We compared whether it is necessary to use Dense Attention during the training steps in the intervals [0, 0.1] and [0.9, 1]. The results from the two sets of experiments, with $\alpha$ = -8.0, $\beta$ = 2.0, and $\alpha$ = -12.0, $\beta$ = 3.0, actually show lower performance than the baseline. This demonstrates that after training, the model requires a period of adaptation to the inference paradigm.

| | GSM8K | NumGLUE | SimulEq | DeepMind | SVAMP | Avg |
|---|---|---|---|---|---|---|
| G | 64.3 | 65.7 | 47.4 | 57.9 | 71.6 | 61.38 |
| N | 59.2 | 74.6 | 46.6 | 55.5 | 70.7 | 61.32 |
| G/2+N | 61.2 | 72.3 | 47.1 | 60.9 | 70.4 | 62.38 |
| G/2+N+M/2 | 62.0 | 73.9 | 48.9 | 62.2 | 71.9 | 63.78 |
| G+N+M | **62.4** | **74.1** | **50.8** | **63.1** | **72.3** | **64.54** |

Table 3: G: GSM8K, M: MATH, N: NumGLUE. G/2 indicates that we only utilized half of the generated GSM8K data, with the aim of exploring whether optimal results can be achieved with a lower data volume.

| | GSM8K | NumGLUE | SimulEq | DeepMind | SVAMP |
|---|---|---|---|---|---|
| without inital tokens | 13.3 | 7.8 | 3.2 | 7.9 | 17.8 |
| $\alpha$= -8.0,$\beta$=1.28 | 61.3 | 73.5 | 47.2 | 62.1 | 71.4 |
| $\alpha$= -9.0,$\beta$=1.44 | 61.5 | 73.6 | **52.1** | 62.4 | 71.8 |
| $\alpha$= -11.0,$\beta$=1.76 | **62.4** | **74.1** | 50.8 | **63.1** | **72.3** |
| $\alpha$= -20.0,$\beta$=3.20 | 60.5 | 73.8 | 50.1 | 62.7 | 71.6 |
| $\alpha$= -8.0,$\beta$=2.0 | 58.7 | 64.9 | 43.1 | 57.6 | 69.9 |
| $\alpha$= -12.0,$\beta$=3.0 | 55.3 | 62.8 | 40.9 | 52.7 | 69.4 |

Table 4: Influence of different Coverage Function hyper-parameters on MAmmoTH-Coder without reinforcement learning. The corresponding function curves show in the Figure 4.

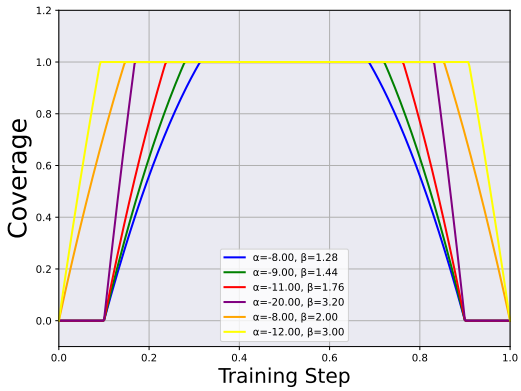


Figure 4: The relationship between Coverage and Training Step percentage under various parameters.

We attempt to set coverage all to 1. The model becomes unavailable during inference and generates a large amount of garbled text. For the remaining experiments, we compare the performance changes under different values of alpha and beta. A slow ascent rate on the curves causes the model to learn inadequately from CoT to PoT, while a too fast rate results in some adaptation issues for the model. Because we set a smaller batch size to give the model more training steps, we compared the performance of this batch size, which is excellent compared to the baseline.

## 4 Discussions

**Do Larger Models Have Issues with PoT?** Our current issue is that the model has some comprehension problems when generating PoT answers in the field of mathematical reasoning. Do these problems also exist in larger models? (Gao et al., 2023) also uses CoT to enhance the generation of PoT and inserts CoT reasoning directly into each line of PoT code. It achieved good results in testing on LaMDA-137B and PaLM-540B. At the same time, we conducted evaluations on MAmmoTH-Coder-13B and MAmmoTH-Coder-34B, calculating the proportion of generated code where CoT is correct but PoT is incorrect. On the five datasets, MAmmoTH-Coder-13B has an average error rate of 7.9%, while MAmmoTH-Coder-34B has an error rate of 7.2%. Even on larger models, the error rate remains high, and our approach is targeted at addressing this issue.

**Why We Use Local Attention** (Beltagy et al., 2020) propose sliding window attention, a mechanism where attention apply in a sliding window manner. It focuses on a fixed-size window around each token, aggregating information within this vicinity for the generation of the current token. In Windows Attention (Beltagy et al., 2020), the source of information generation can be controlled through the attention window. The experiments by (Han et al., 2021) and (Pan et al., 2023) show that

local attention mechanisms can focus information more effectively.

**The Application Domains of Focus Attention** In our work, we demonstrate that focus attention plays a crucial role in focusing information during the fine-tuning stage of downstream tasks, thereby addressing the shortcomings of previous local attention mechanisms that may lead to information loss. In any inference task, we can potentially enhance model performance by controlling specific intermediate steps. From a novel perspective, we can apply focus attention in the dimension of time (Goyal et al., 2023). Utilizing focus attention to create a special time token allows LLMs to transition over this token, thereby extending the deliberation time of large models. Currently, we make a slight attempt to add some special tokens between the question and PoT. However, due to the limited amount of data, the model does not adapt to this change.

**LLM's Digital Computation Abilities** The mathematical calculation errors of LLMs originate from two aspects. The first is due to the specificity of how language models understand text—they need to split each sentence into different tokens. For a multi-digit number, it cannot always be treated as a whole. Establishing more advanced token representations is one solution. The other aspect is the internal computational capability for numbers, where multiplication of multi-digit numbers is a significant challenge for LLMs currently. (Dziri et al., 2023) shows that GPT-4 achieves only 59% accuracy on three-digit multiplication problems, while its accuracy for four-digit and five-digit multiplication drops to 4% and 0%, respectively.

**Why No Math Dataset for Testing?** The Math dataset contains numerous highly challenging proof problems in Algebra and Number Theory that are difficult to solve through numerical computation. Even state-of-the-art models achieve low accuracy on these problems. As a result, we do not include the Math dataset in our testing, but we do use a portion of its training data for our purposes.

# 5 Related Work

## 5.1 CoT in Mathematical Reasoning

Current methods primarily rely on the CoT to address mathematical problems, generating reasoning answers in a step-by-step manner (Nye et al., 2021; Imani et al., 2023; Miao et al., 2023). The focus of recent research centers on data engineering and prompt design. In the realm of data engineering, the efforts aim to enhance the quality (Wang et al., 2023a; Liu et al., 2023) and increase the volume (Luo et al., 2023) of CoT data. However, another stream of research identifies several computational challenges associated with exclusively using the CoT approach. In response, (Chen et al., 2023) introduces the PoT, a method that employs Python programs for achieving more accurate results. (Yue et al., 2023) attempts to merge CoT and PoT data in their dataset, resulting in significant improvements. Similarly, (Gao et al., 2023) seeks to enhance PoT with CoT by weaving CoT snippets into code lines, employing few-shot prompting to guide the model.

## 5.2 LLM in Mathematical Reasoning

Both (Testolin, 2024) and (Lu et al., 2022) explore the application of Deep Learning within the realm of mathematical reasoning. With the development of LLMs, they have proven to be powerful assets in unraveling the intricacies of solving mathematical problems (Romera-Paredes et al., 2023; Imani et al., 2023). LlaMA-2 demonstrates powerful mathematical reasoning capabilities in open-source models. CodeLlaMA (Rozière et al., 2023) reduces a significant amount of computational errors using the PoT. WizardMath (Luo et al., 2023) introduces the Reinforcement Learning from Evol-Instruct Feedback (RLEIF) method into the field of mathematics. (Luong et al., 2024) innovatively enhances the PPO algorithm by devising a function to directly assess answer correctness, obviating the requirement for a reward model. Mistral is a model that uses Grouped-query attention for faster inference and Sliding Window Attention to handle longer sequences at a smaller cost. Mistral exhibits strong mathematical capabilities, with Mistral-7b outperforming LlaMA-2-13B on all benchmarks

# 6 Conclusion

In our paper, we propose a method for mathematical reasoning with large models that combines the advantages of CoT and PoT. We evaluate the shortcomings of existing models and find that without introducing any additional information, simply by instruction tuning, the model shows improved performance. Our approach is lightweight, automated, and does not require any additional manual data. Our method can easily transfer to other models.

## Limitations

**Lack of Equipment** Due to GPU limitations, we only use 36,000 data points for training and use Lora in the PPO stage. Simultaneously, our newly introduced attention mechanism also requires a large amount of data for learning. This amount of data is very small for the 7B model. After applying our method, the initialization issue still persists when the model generates PoT. Whether more data can completely eliminate this problem, we do not yet know.

**Exploration of Focus Attention** Regarding Focus Attention, we have not yet determined the specific reason for the need for a gradual increase in coverage to adapt to inference. If we extend this approach to other domains, such as incorporating it into the pre-training stage, it enables the model to better learn step-by-step generation, PoTentially leading to improved results.

**Experimental Limitations with Closed-source Models** We conduct experiments only on open-source large models. Due to the cost of running the models in the API, we do not attempt to explore whether this paradigm can enhance the inference capabilities of models like GPT-4 by constructing similar prompts. However, in our experiments, merely adjusting the prompts does not result in a significant performance improvement. Furthermore, because the training data for GPT-4 is unknown, its results are challenging to interpret.

## References


Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges.

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, and so on. 2023. Palm 2 technical report.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, Amanda Askell, Jackson Kernion, Andy Jones, Anna Chen, Anna Goldie, Azalia Mirhoseini, Cameron McKinnon, Carol Chen, Catherine Olsson, Christopher Olah, Danny Hernandez, Dawn Drain, Deep Ganguli, Dustin Li, Eli Tran-Johnson, Ethan Perez, Jamie Kerr, Jared Mueller, Jeffrey Ladish, Joshua Landau, Kamal Ndousse, Kamile Lukosuite, Liane Lovitt, Michael Sellitto, Nelson Elhage, Nicholas Schiefer, Noemi Mercado, Nova DasSarma, Robert Lasenby, Robin Larson, Sam Ringer, Scott Johnston, Shauna Kravec, Sheer El Showk, Stanislav Fort, Tamera Lanham, Timothy Telleen-Lawton, Tom Conerly, Tom Henighan, Tristan Hume, Samuel R. Bowman, Zac Hatfield-Dodds, Ben Mann, Dario Amodei, Nicholas Joseph, Sam McCandlish, Tom Brown, and Jared Kaplan. 2022. Constitutional ai: Harmlessness from ai feedback.

Iz Beltagy, Matthew E Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv preprint arXiv:2004.05150*.

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

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.

Yihe Deng, Weitong Zhang, Zixiang Chen, and Quanquan Gu. 2023. Rephrase and respond: Let large language models ask better questions for themselves.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang Lorraine Li, Liwei Jiang, Bill Yuchen Lin, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena D. Hwang, Soumya Sanyal, Sean Welleck, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality.

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

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2023. Think before you speak: Training language models with pause tokens.

Qi Han, Zejia Fan, Qi Dai, Lei Sun, Ming-Ming Cheng, Jiaying Liu, and Jingdong Wang. 2021. On the connection between local attention and dynamic depth-wise convolution. *arXiv preprint arXiv:2106.04263*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. Mathprompter: Mathematical reasoning using large language models.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Solomon Kullback and Richard A Leibler. 1951. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models.

Bingbin Liu, Sebastien Bubeck, Ronen Eldan, Janardhan Kulkarni, Yuanzhi Li, Anh Nguyen, Rachel Ward, and Yi Zhang. 2023.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2022. A survey of deep learning for mathematical reasoning. *arXiv preprint arXiv:2212.10535*.

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

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning.

Aman Madaan, Shuyan Zhou, Uri Alon, Yiming Yang, and Graham Neubig. 2022. Language models of code are few-shot commonsense learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1384–1403, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Ning Miao, Yee Whye Teh, and Tom Rainforth. 2023. Selfcheck: Using llms to zero-shot check their own step-by-step reasoning.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. 2022. NumGLUE: A suite of fundamental yet challenging mathematical reasoning tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3505–3523, Dublin, Ireland. Association for Computational Linguistics.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. 2021. Show your work: Scratchpads for intermediate computation with language models.

OpenAI, :, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, and Sam Altman. 2023. Gpt-4 technical report.

Xuran Pan, Tianzhu Ye, Zhuofan Xia, Shiji Song, and Gao Huang. 2023. Slide-transformer: Hierarchical vision transformer with local self-attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2082–2091.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 3505–3506, New York, NY, USA. Association for Computing Machinery.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2023. Mathematical discoveries from program search with large language models. *Nature*, pages 1–3.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science.

Alberto Testolin. 2024. Can neural networks do arithmetic? a survey on the elementary numerical skills of state-of-the-art deep learning models. *Applied Sciences*, 14(2):744.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023a. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning.

Lei Wang, Wanyu Xu, Yihuai Lan, Zhiqiang Hu, Yunshi Lan, Roy Ka-Wei Lee, and Ee-Peng Lim. 2023b. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2609–2634, Toronto, Canada. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

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.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. 2023. Efficient streaming language models with attention sinks. *arXiv preprint arXiv:2309.17453*.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. Mammoth: Building math generalist models through hybrid instruction tuning.

Rui Zheng, Shihan Dou, Songyang Gao, Yuan Hua, Wei Shen, Binghai Wang, Yan Liu, Senjie Jin, Qin Liu, Yuhao Zhou, Limao Xiong, Lu Chen, Zhiheng Xi, Nuo Xu, Wenbin Lai, Minghao Zhu, Cheng Chang, Zhangyue Yin, Rongxiang Weng, Wensen Cheng, Haoran Huang, Tianxiang Sun, Hang Yan, Tao Gui, Qi Zhang, Xipeng Qiu, and Xuanjing Huang. 2023. Secrets of RLHF in large language models part I: PPO. *CoRR*, abs/2307.04964.

Daniel M. Ziegler, Nisan Stiennon, Jeffrey Wu, Tom B. Brown, Alec Radford, Dario Amodei, Paul Christiano, and Geoffrey Irving. 2020. Fine-tuning language models from human preferences.

## A Error Case

We show some examples in Figure 5.

## B Instruction Template

**Tune Instruction**

Below is an instruction that describes a task.
Write a response that appropriately completes the request.
I'd like you to solve this problem in 3 steps:
1.Answer the question in plain language without writing any code.
2.Output one line of *.
3.Write program code based on the solution process in step 1 to solve the problem.
### Instruction:
<Question>
Let's write a program.
### Response:"
——Model Output——
<CoT Answer>
****
<PoT Answer>

**Insert Instruction**

Below is an instruction that describes a task.
Write a response that appropriately completes the request.
After the instruction, there is an existing solution. You need to write a corresponding solution program when referring to this solution.
### Instruction:
<Question>
### Solution:
<CoT Answer>
Let's write a program.
### Response:
——Model Output——
<PoT Answer>

| Question | CoT Answer | PoT Answer |
|---|---|---|
| Jerry had 2 action figures and 10 books on a shelf in his room. Later he added 4 more action figures to the shelf. How many more books than action figures were on his shelf? | Jerry had 2 + 4 = <<2+4=6>>6 action figures on his shelf.<br>He had 10 - 6 = <<10-6=4>>4 more books than action figures on his shelf.<br>#### 4 | action_figures = 2<br>books = 10<br>added_action_figures = 4<br>difference = (action_figures + added_action_figures) - books<br>print(difference)<br>>>> -4 |
| 21 children were riding on the bus. At the bus stop 10 children got off the bus while some more got on the bus. Then there were 16 children altogether on the bus. How many children got on the bus at the bus stop? | There are 21 - 10 = <<21-10=11>>11 children who got off the bus.<br>So there are 16 - 11 = <<16-11=5>>5 children who got on the bus.<br>#### 5 | children_before = 21<br>children_after = 16<br>children_off = 10<br>children_on = children_after - children_before - children_off<br>print(children_on)<br>>>> -15 |
| Rachel has 3 apple trees. She picked 8 apples from each of her trees. Now the trees have a total 9 apples still on them. How many apples were there on all trees initially? | Rachel picked 3 trees * 8 apples/tree = <<3*8=24>>24 apples.<br>There were 24 apples + 9 apples = <<24+9=33>>33 apples on all trees initially.<br>#### 33 | apples_picked_from_each_tree = 8<br>total_apples_on_trees = 9<br>number_of_trees = 3<br>total_apples_before = (total_apples_on_trees * number_of_trees) + (apples_picked_from_each_tree * number_of_trees)<br>print(total_apples_before)<br>>>> 51 |
| In the first half of a soccer match, team A scores 4 goals while team B scores 2 goals fewer than team A. In the second half, team A scores 1/4 of the number of goals scored by team B, which scores 4 times the number of goals it scored in the first half. What's the total number of goals scored in the match? Let's write a program. | If team A scored 4 goals in the first half, team B scored 4-2 = 2 goals.<br>In the second half, team B scored 4*2 = 8 goals.<br>The total number of goals scored by team A in the match is 4+1/4*8 = 4+2 = 6<br>The total number of goals scored by team B in the match is 2+8 = 10<br>The total number of goals scored in the match is 10+6 = 16<br>#### 16 | teamA_first_half = 4<br>teamB_first_half = teamA_first_half - 2<br><br># define the scores in the second half<br>teamA_second_half = teamB_first_half / 4<br>teamB_second_half = 4 * teamB_first_half<br><br># calculate the total score<br>total_score = teamA_first_half + teamB_first_half + teamA_second_half + teamB_second_half<br><br># print the total score<br>print(total_score)<br>>>> 14.5 |

Figure 5: The Figure consists of examples where CoT is correct and PoT is incorrect. The first three rows represent logical errors in PoT, which result in incorrect formulas when calculated. The last row represents errors in variable initialization. In PoT answers, such errors typically occur when the initial values need to be computed to obtain the correct result.