Parrot: A Training Pipeline Enhances Both Program CoT and Natural Language CoT for Reasoning

Senjie Jin[1*], Lu Chen[1*], Zhiheng Xi[1*], Yuhui Wang[1], Sirui Song[1], Yuhao Zhou[1], Xinbo Zhang[2†], Peng Sun[2], Hong Lu[1,4], Tao Gui[1,3,4†], Qi Zhang[1,4], Xuanjing Huang [1,4]

[1]College of Computer Science and Artificial Intelligence, Fudan University
[2]ByteDance Research [3]Shanghai Innovation Institute
[4]Shanghai Key Laboratory of Intelligent Information Processing
sjjin24@m.fudan.edu.cn
zhangxinbo.freya@bytedance.com, tgui@fudan.edu.cn

Abstract

Natural language chain-of-thought (N-CoT) and Program chain-of-thought (P-CoT) have emerged as two primary paradigms for large language models (LLMs) to solve mathematical reasoning problems. Current research typically endeavors to achieve unidirectional enhancement: P-CoT enhanced N-CoT or N-CoT enhanced P-CoT. In this paper, we seek to fully unleash the two paradigms' strengths for mutual enhancement and ultimately achieve simultaneous improvements. We conduct a detailed analysis of the error types across two paradigms, based on which we propose **Parrot**, a novel training pipeline for mathematical problems: 1) Three target-designed subtasks integrate sequential P-CoT and N-CoT generation. 2) A subtask hybrid training strategy to facilitate natural language semantic transferability. 3) The converted N-CoT auxiliary reward is designed to alleviate the sparse rewards in P-CoT optimization. Extensive experiments demonstrate that Parrot significantly enhances both the performance of N-CoT and P-CoT, especially on N-CoT. Using Parrot SFT, the LLaMA2's and CodeLLaMA's N-CoT performance achieve gains of +21.87 and +21.48 on MathQA over the RL baseline, which is resource-intensive[1].

1 Introduction

Large language models (LLMs) have exhibited an impressive success in multi-step mathematical reasoning (Wang et al., 2024; Shao et al., 2024; Wan et al., 2024). The existing work primarily concentrates on enabling models to generate natural language chain-of-thought (N-CoT) rationales (Wei et al., 2022) or leverage executable and verifiable code, such as Python (Chen et al., 2022; Gao et al., 2023; Luong et al., 2024; Xi et al., 2024), to generate program chain-of-thought (P-CoT) for offloading intensive calculations (Li et al., 2024c). These two paradigms exhibit distinct advantages. Specifically, N-CoT introduces more reasoning details by an explicit thinking process (Lin et al., 2024), which is more comprehensible and holds a broader applicability (Renze and Guven, 2024; Kumar et al., 2024), while P-CoT demonstrates high effectiveness (Gao et al., 2023) and enables easy process verification (Gou et al., 2023).

Current research typically endeavors to utilize one to facilitate the other: (1) N-CoT-enhanced P-CoT. Integrating an explicit natural language analysis prior to each code step or the entire code solution (Gao et al., 2023; Lin et al., 2024; Li et al., 2024b). (2) P-CoT-enhanced N-CoT. Presenting specific procedures as code and invoking them through an external verifier (Gou et al., 2023). Although (Yue et al., 2024) proposes a N-CoT&P-CoT rationale hybrid training strategy, which mainly aims at the solution diversity. The synergistic facilitation potential between these paradigms has not been sufficiently explored.

In this paper, we first conduct a comprehensive error analysis (Section 2) of these two paradigms and find that, on the one hand, in addition to intrinsic limitations in logical reasoning, the approach of directly generating P-CoT from problems struggles with accurate variable definition and problem comprehension (Yue et al., 2024; Li et al., 2024b). We integrate these capabilities suitable for natural language by constructing specialized subtasks and employing hybrid training. On the other hand, N-CoT mainly suffers from logical confusion (Xi et al., 2023; Wang et al., 2022) as well as calculation errors in intermediate steps (Gao et al., 2023). We enable N-CoT to refer to the concise P-CoT reasoning steps and incorporate the intermediate results of the latter as a simple yet effective form of process supervision (Lightman et al., 2023).

Based on the above, we propose **Parrot**, as

[*] Equal contribution.[†]Corresponding authors.
[1]https://github.com/Leonnnnnn929/ParrotTraining

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 26899–26916
November 4-9, 2025

illustrated in Figure 2, a novel training pipeline to promote both P-CoT and N-CoT performance on mathematical problems. The pipeline comprises three target-designed subtasks: **Information Retrieval** trains the model to concentrate on key information within problem. **P-CoT Reasoning** utilizes the information to generate variable well-defined code solutions. **Paradigm Conversion** enhances N-CoT with concise P-CoT and its intermediate outputs. This pipeline also aligns with the human problem-solving process (Krawec, 2014), which involves three stages: individuals examine the problem and identify key information, then utilize the formalized language for unambiguous declarations, thereby incorporating the characteristic problem context to generate interpretable and accessible resolutions. (Kazemi et al., 2012).

Regarding methodology, we initially adopt a hybrid Supervised Fine-Tuning (SFT) strategy, enabling the model to master subtasks while enhancing P-CoT through transferability across remaining subtasks (Yue et al., 2024). We will thoroughly discuss the impact of each sub-task in the analysis section 5.1. Furthermore, we introduce Reinforcement Learning (RL) to verify Parrot's applicability under different fine-tuning methods and data efficiency. During Online Self-Learning (On-SL) (Uesato et al., 2022; Anthony et al., 2017), we collect N-CoT solutions and use them in SFT to demonstrate their quality with the support of P-CoT. In the Proximal Policy Optimization (PPO) (Schulman et al., 2017) stage, we use the validity of the converted N-CoT as the auxiliary reward signal to mitigate the issue of sparse rewards (Zhong et al., 2017; Le et al., 2022) for P-CoT verification in mathematical reasoning.

In summary, we make the following contributions:

(1) We carry out a comprehensive analysis of limitations for coding-expertise (CodeLLaMA) and non-coding-expertise (LLaMA2) within P-CoT and N-CoT paradigms.

(2) We propose Parrot, a novel training pipeline enhancing both P-CoT and N-CoT mathematical reasoning performance. Additionally, we conduct extensive ablations to analyze the impact of each sub-task.

(3) We perform SFT on the collected N-CoT from On-SL to validate its quality with the aid of P-CoT, and we use the N-CoT auxiliary reward to mitigate the reward sparsity issue in the P-CoT RL phase.

(4) We conduct extensive experiments on three difficulty-level datasets and model families, which indicate that Parrot can effectively improve the model's P-CoT and N-CoT reasoning performance, especially on N-CoT.

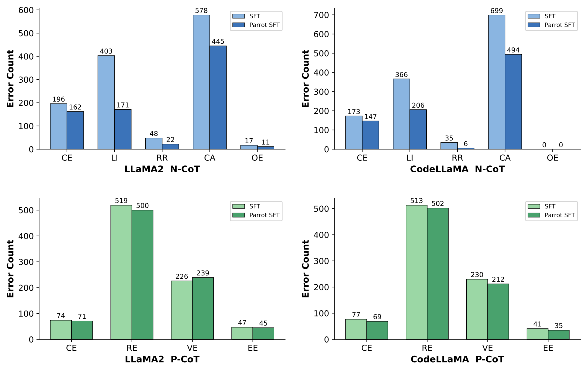

Figure 1: The histogram of error types. The labels on the x-axis are defined in section 2.1, while **OE** denotes Other Errors. Results from SFT are shaded in light colors, and Parrot SFT results are presented in dark colors.

2 Preliminary Analysis

Pre-training on different corpus compositions (Lu et al., 2024b) and reasoning paradigms collectively determines the model performance across math problems. We first perform a detailed error analysis on the coding-expertise models (CodeLLaMA) (Rozière et al., 2023) and non-coding-expertise (LLaMA2) models (Touvron et al., 2023) to investigate their intrinsic limitations in P-CoT and N-CoT. Following previous work (Luong et al., 2024), we perform Supervised Fine-Tuning (SFT) training on the MathQA (Amini et al., 2019) dataset and collect error samples. The error types and analysis are elaborated in the following sections.

2.1 Empirical Identification of Error Types

We first randomly sampled 50 error cases from each paradigm for manual examination. Our findings reveal that: (1) For N-CoT, except **calculation error** (Gao et al., 2023), the model also suffers from **logical inconsistency**, **problem comprehension**, **redundant and repetitive** information (Li et al., 2024b). (2) For P-CoT, besides the model's inherent **reasoning** limitations, generating P-CoT directly from problems has an issue

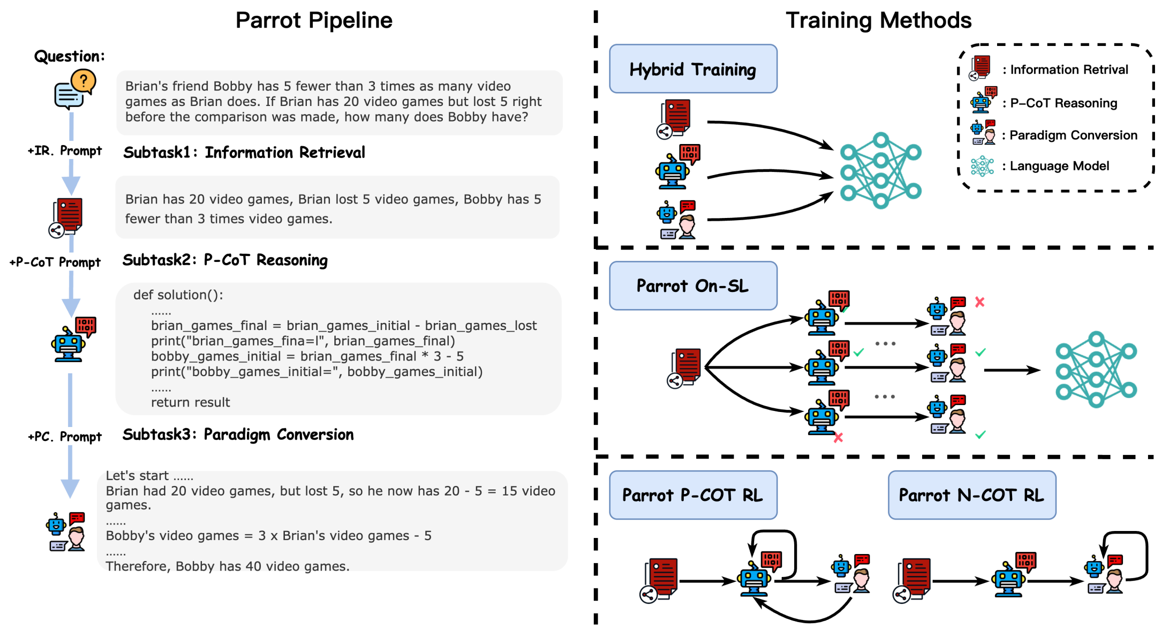

Figure 2: The training pipeline and methods of **Parrot**. On the left, the pipeline consists of three subtasks: **Information Retrieval**, **P-CoT Reasoning**, and **Paradigm Conversion**. By these subtasks, the model sequentially generates P-CoT and n-CoT. On the right, we use a Hybrid Supervised Fine-Tuning (SFT) strategy to enable semantic transfer and incorporate reinforced algorithms for further improvements. The detailed Parrot inference process and subtask prompts are provided in Appendix A.3.

in **problem comprehension**, **variable definition**, and **expression error**. We then utilize GPT-4 (OpenAI, 2023) to statistically analyze all error samples. Specific descriptions of error types and evaluation prompts are detailed in the Appendix A, and error examples in Appendix D.1.

2.2 Error Analysis

The statistical results are shown in Figure 1. We observe that: (1) For N-CoT, consistent with previous work (Gao et al., 2023), approximately half of the errors stem from calculation (**CA**), followed by logic inconsistency (**LI**), which we hypothesize are due to the absence of process supervision. In some cases, the model also exhibits the phenomenon of redundant and repetitive (**RR**). From the model perspective, incremental training in code enhances the model's logical capabilities and reasoning conciseness. However, this also results in insufficient semantic understanding, leading to more calculation errors. (2) For P-CoT, as proposed by (Li et al., 2024b), the code is inferior to natural language in semantic analysis and abstract reasoning. The primary issue arises from reasoning errors (**RE**). The model also fails short on variable definition errors (**VE**), accounting for one-quarter errors and Expression Errors (**EE**). Both paradigms struggle with problem comprehension errors (**CE**), with P-CoT fewer since its variable definition analysis.

3 Method

Motivation. From current cross-party facilitating works and the error types uncovered in section 2, we aim to explore the feasibility of leveraging strengths to mitigate counterpart drawbacks in P-CoT and N-CoT and ultimately achieve collective performance improvement. Hence, we propose **Parrot**, a novel training pipeline that focuses on key information to facilitate the variable well-defined P-CoT and generates N-CoT based on P-CoT and its intermediate outputs. We elaborate the details about pipeline subtasks in section 3.1 and training methods in section 3.2 and section 3.3, which is illustrated in Figure 2.

3.1 Pipeline Subtask Construction

We organically decompose the full training pipeline into three targeted distinct subtasks. 1) **Information Retrieval**. Although P-CoT enables precise calculations with the support of the external verifier, it often suffers from erroneous variable definitions (Jie et al., 2023). We first orient the model's attention on key numeric information within the problem to achieve variable well-defined in P-CoT. For a given problem x and information retrieval prompt p_1, key information d_1 is generated by:

$$d_1 \sim \Pi(\cdot|x \oplus p_1), \tag{1}$$

where $\oplus$ donates concatenation. **Note** in this phase, no extra knowledge is incorporated. 2) **P-CoT Reasoning**. Subsequently, utilizing the key information d_1 and code inference prompt p_2, the model generates a python snippet d_2:

$$d_2 \sim \Pi(\cdot|x \oplus p_1 \oplus d_1 \oplus p_2), \tag{2}$$

which is then validated by invoking the interpreter. 3) **Paradigms Conversion**. By harnessing the model's multilingual alignment capability (Xu et al., 2024a), we generate more understandable and widely accessible N-CoT based on the P-CoT, its intermediate results i and prompt p_3:

$$d_3 \sim \Pi(\cdot|x \oplus p_1 \oplus d_1 \oplus p_2 \oplus d_2 \oplus i \oplus p_3), \tag{3}$$

as analyzed in section 2, there are mainly two reasons for this N-CoT generation strategy: Firstly, the code reasoning features concise steps, which help alleviate repetition and redundancy errors. Secondly, the main errors with N-CoT are calculations and logical inconsistencies. Beyond precise calculations, incorporating P-CoT's intermediate results can serve as a simple and effective process supervision (Luo et al., 2024; Chen et al., 2025), and ablation analysis in section 5.3 validates our hypothesis.

3.2 Subtask Hybrid Training

Motivated by (Yue et al., 2024), we adopt a hybrid training strategy and structure all the subtasks into a unified input-output form to perform multi-task SFT training (Zhang and Yang, 2021) instead of training sequentially by subtasks, which often involves the challenge of knowledge degradation (Xu et al., 2024b; Su et al., 2024) and impairs the model's performance. This strategy is poised to facilitate problem comprehension due to the incorporation of solution diversity from P-CoT and N-CoT (Liang et al., 2024), and to transfer the explicit reasoning traces from N-CoT for semantic analysis (Lin et al., 2024), ultimately to enhance the logical reasoning ability of P-CoT. We conduct extensive ablation experiments to validate our hypotheses and thoroughly analyze the impact of each subtask within the pipeline in section 5.1.

3.3 Reinforcement Enhanced Reasoning

Upon completing model initialization through hybrid training, we incorporate reinforcement learning algorithms to further verify Parrot's applicability under different fine-tuning methods and data efficiency.

Online Self-learning. We implement the online self-training (On-SL) following (Luong et al., 2024). In our setup, the model sequentially generates P-CoT and N-CoT rollouts, using jointly correct samples to augment training with the original datasets.

Proximal Policy Optimization. We leverage proximal policy optimization (PPO) (Schulman et al., 2017) with a clipped objective as reinforcement learning (RL) algorithm. The final token before <eos> of the sampled sequence is assigned a reward score, while all remaining tokens receive 0 (Yu et al., 2023; Xi et al., 2024).

$$R(s_{t-1}, a_t) = \begin{cases} R_f(s_{t-1}, a_t), & t = T \\ 0, & t \neq T \end{cases}, \tag{4}$$

where $R_f(\cdot)$ is a rule-based reward function merely relies on the correctness of the answer. Despite its efficiency, it suffers reward sparsity. Inspired by partial reward design (Li et al., 2024b; Le et al., 2022), we use the validity of the converted N-CoT as the auxiliary reward signal to verify the P-CoT:

$$R_f(s_{T-1}, a_T) = \begin{cases} 1, & \text{Both answer correct} \\ 1-\gamma, & \text{P-CoT correct, N-CoT null} \\ \epsilon, & \text{P-CoT not, but numeric} \\ 0, & \text{P-CoT null} \end{cases} \tag{5}$$

when the converted N-CoT is incorrect but of numeric type, we consider it a calculation error. For cases with no answer, we give P-CoT reasoning a penalty γ for comprehension difficulty to enhance its effectiveness. The value model V_Φ is constructed by appending a linear value head on top of the last hidden states of the policy model π_θ^p. Consistent with (Luong et al., 2024), the final reward $R_f(s_{t-1}, a_t)$ integrates both the reward score and the token-level Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951). Based on the reward R and value model V_Φ, we estimated the generalized advantage esti-mate (GAE) (Schulman et al., 2017) $A(s_{t-1}, a_t)$, and the optimal objective is to maximize the return:

Training Method	Size	GSM8K		SVAMP		$\text{MathQA}_{\text{numeric}}$		Average	
		N-CoT	P-CoT	N-CoT	P-CoT	N-CoT	P-CoT	N-CoT	P-CoT
Tora + CodeLLaMA	7B	-	72.60*	-	70.40*	-	-	-	-
MathGenie + LlaMA2	7B	-	71.70*	-	78.50*	-	-	-	-
MathGenie + CodeLLaMA	7B	-	71.50*	-	80.20*	-	-	-	-
DotaMath + LlaMA2	7B	-	79.60*	-	-	-	-	-	-
MARIO + DeepSeek	7B	-	78.40*	-	-	-	-	-	-
HTL + CodeLLaMA	7B	-	65.70*	-	74.40*	-	-	-	-
HTL + Mistral	7B	-	78.10*	-	82.40*	-	-	-	-
GPT-4	-	92.72	97.00*	91.60	94.80*	83.17	66.29	89.16	86.03
LLaMA2 + SFT	7B	44.05	58.61	58.60	69.50	22.62	46.04	41.76	58.05
LLaMA2 + MAmmoTH SFT	7B	47.54	58.15	59.30	71.90	27.28	44.80	44.71	58.28
LLaMA2 + On-SL	7B	45.94	60.80	**60.70**	69.40	30.15	46.48	45.60	58.89
LLaMA2 + RL	7B	44.96	63.99	59.70	71.40	26.92	44.92	43.86	60.10
LLaMA2 + Parrot SFT	7B	60.81	59.74	59.60	71.60	48.79	46.73	56.40	59.42
LLaMA2 + Parrot On-SL	7B	60.96	59.21	59.40	69.60	49.22	45.92	56.53	58.24
LLaMA2 + Parrot RL	7B	**61.26**	**66.03**	60.00	**73.60**	**50.37**	**47.66**	**57.21**	**62.43**
CodeLLaMA + SFT	7B	44.88	65.05	56.70	75.50	22.37	47.04	41.32	62.53
CodeLLaMA + MAmmoTH SFT	7B	46.70	65.50	62.50	75.70	24.05	46.23	44.42	62.48
CodeLLaMA + On-SL	7B	45.19	65.43	59.70	76.30	26.17	48.10	43.69	63.28
CodeLLaMA + RL	7B	53.22	72.78	62.30	78.40	25.36	48.16	46.96	66.45
CodeLLaMA + Parrot SFT	7B	64.90	66.19	62.90	77.60	46.84	**49.03**	58.21	64.27
CodeLLaMA + Parrot On-SL	7B	64.82	65.73	61.70	75.40	47.04	48.29	57.85	63.14
CodeLLaMA + Parrot RL	7B	**65.04**	**74.53**	**64.30**	**79.60**	**48.35**	48.85	**59.23**	**67.66**

Table 1: The main experimental results on three benchmarks and two models. We simultaneously evaluated the model's performance on N-CoT and P-CoT. The results of Parrot-based methods are presented at the bottom of each block, with the overall performance outperforming those corresponding baselines. The best result is in **bold** while the second is marked with underline. $*$ indicates we report results from the corresponding paper. Some work in top block interleaves the natural language and code, which we classify as enhanced P-CoT or PAL (Gao et al., 2023), reporting P-CoT performance. **Note** that the SVAMP performance of MathGenie and HTL is evaluated in an OOD setting.

$$\mathbb{E}_{\tau\sim\pi_\theta^p}\left[\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta^p(a_t|s_{t-1})A(s_{t-1},a_t)\right], \quad (6)$$

where τ is the sampled sequence.

4 Experiments

4.1 Datasets and Models.

We conduct experiments on three widely used mathematical reasoning datasets spanning different difficulty levels: SVAMP (Patel et al., 2021), GSM8K (Cobbe et al., 2021), and MathQA (Amini et al., 2019). For MathQA, we convert the multiple-choice (i.e., ABCD) format into a numeric version to fit the unified input-output form. As for data sources, construction details, and train sizes, please refer to the Appendix C.

We choose LLaMA2-Base-7B (Touvron et al., 2023) and CodeLLaMA-7B (Rozière et al., 2023) as our foundation models due to their stability and widespread usage. Additionally, compared to LLaMA2, CodeLLaMA includes extra 500B code tokens, which help validate the differing performances of Parrot on code-expert and non-code-expert models. We also conduct experiments on LLaMA-3-8B (Grattafiori et al., 2024), LLaMA-3.2-3B[2], and Qwen-2.5-1.5B[3] with more complex MathQA to validate the applicability of Parrot in Section 5.4.

4.2 Baselines.

Our work aims to jointly enhance the performance of P-CoT and N-CoT through mutual promotion, primarily employing hybrid training and reinforcement learning methods. We use the following methods as baselines:

Standard SFT and RL methods. SFT quantifies the model's ability to learn from P-CoT and N-CoT demonstrations, validating the intrinsic advantages and drawbacks of these two paradigms. RL trains models by searching and learning (Kumar et al., 2025), with performance critically dependent on model initialization and reward design. Following (Luong et al., 2024),

[2] https://huggingface.co/meta-LLaMA/LlaMA-3.2-3B

[3] https://huggingface.co/Qwen/Qwen2.5-1.5B

we have implemented the Online Self-Learning (On-SL) (Hoi et al., 2021) and Proximal Policy Optimization (PPO) algorithms.

MAmmoth (Yue et al., 2024) trains the model using hybrid N-CoT and P-CoT rationales. For fair comparison, we re-implemented it on our P-CoT and N-CoT datasets and used different prompts for inference.

HTL (Li et al., 2024b) first generates CoT, which is used to guide the generation of P-CoT, and further uses error assessment-based PPO.

Tora (Gou et al., 2023) uses natural language reasoning interleaved with program-based tool use.

MathGenie (Lu et al., 2024a) employs solution back-translation to enhance the question diversity.

DotaMath (Li et al., 2024a) employs the decomposition of thoughts with code assistance and self-correction for mathematical reasoning.

MARIO (Liao et al., 2024) introduces a novel math dataset and enhanced with a capability to utilize a Python code interpreter.

Proprietary model. We also incorporate the closed-source model GPT-4 (OpenAI, 2023), which represents the advanced performance in mathematical reasoning.

4.3 Training details.

The specific training and implementation details can be found in Appendix B.

4.4 Experimental Results

The main experimental results are presented in Table 1. We primarily analyze the model performance on N-CoT reasoning and P-CoT reasoning.

Results on N-CoT reasoning. Compared to methods that directly generate N-CoT from problems, Parrot N-CoT refers to P-CoT and its intermediate results, which serve as a simple yet effective process supervision as discussed in section 5.3. Meanwhile, P-CoT's concise reasoning steps enable N-CoT to alleviate the issues of redundant information and logical incoherence. Overall, we found the following: 1) Generating N-CoT from P-CoT proves highly effective across all benchmarks, exceeding most of the baselines, and the performance improves with the enhancement of P-CoT's performance, which is relatively less challenge. 2) Parrot provides an efficient way for obtaining high N-CoT performance. After Parrot SFT, the model achieves significant improvements comparable to baseline RL. For example, LLaMA2-7B performers better, 12.54 on average, while RL requires considerable resources for searching and learning. 3) The benefit is more pronounced on the challenging MathQA dataset. While the model can't effectively learning using pure natural language, P-CoT compensates for this limitation. 4) The performance of N-CoT even outperforms P-CoT on LLaMA2-7B for MathQA dataset. We hypothesize this is due to natural language being more suited for semantic analysis and planning of complex problems with clear logic and process signals (Li et al., 2024b). The Parrot N-CoT example is provided in Appendix D.2.

Results on P-CoT reasoning. Similarly, the Parrot's average P-CoT performance is on par with or surpasses corresponding baselines, highlighting the significance of information retrieval and transferability afforded by hybrid training. A detailed subtask ablation will be provided in section 5.1. In addition, we found that: 1) Compared to the baseline RL, Parrot RL demonstrates clear improvements, with gains of 2.33 and 1.21 on LLaMA2 and CodeLLaMA. This indicates that models with proper initialization and reward design exhibit enhanced exploration capabilities. 2) The model's performance on Parrot On-SL has declined, likely as a result of overfitting stemming from the combination of hybrid training and the absence of negative examples during this phase.

5 Analysis and Discussion

Parrot primarily achieves mutual enhancement through three specially designed subtasks and hybrid training. We give a detail ablation analysis in section 5.1, and we further discuss: 1) The impact of the N-CoT penalty in P-CoT PPO in section 5.2, 2) With the aid of P-CoT, which errors are solved and how N-CoT's quality in section 5.3, 3) The applicability of Parrot training pipeline in section 5.4.

5.1 Ablations Analysis

Subtask Ablation. We analyze each subtask's role sequentially, and the results are in Table 2: 1) For **Information Retrieval**, which is designed

Table 2: The results of ablation experiments on Parrot subtasks. **IR.** refers to information retrieval and **PC. w/o im** is paradigms conversion without intermediate results while **PC. w/ im** is with intermediate results.

Model	**Subtask**	**GSM8K**		**SVAMP**		**MathQA$_{\text{numeric}}$**		**Average**	
		N-CoT	**P-CoT**	**N-CoT**	**P-CoT**	**N-CoT**	**P-CoT**	**N-CoT**	**P-CoT**
LLaMA2	N/P-CoT	44.05	58.61	58.60	69.50	22.62	46.04	41.76	58.05
	IR. + N/P-CoT	45.19	57.85	58.40	67.10	26.23	46.20	43.27	57.05
	P-CoT + PC. w/o im	49.43	59.06	59.60	71.60	27.85	45.98	45.63	58.88
	P-CoT + PC. w/ im	60.81	59.74	-	-	46.82	46.54	-	-
	Parrot SFT	60.81	59.74	59.60	71.60	48.79	46.73	56.40	59.36
CodeLLaMA	N/P-CoT	44.88	65.05	56.70	75.50	22.37	47.04	41.32	62.53
	IR. + N/P-CoT	45.34	65.96	56.20	75.60	22.55	47.23	41.36	62.93
	P-CoT + PC. w/o im	50.42	65.13	62.90	77.60	25.17	46.54	46.16	63.09
	P-CoT + PC. w/ im	64.90	66.19	-	-	42.73	46.86	-	-
	Parrot SFT	64.90	66.19	62.90	77.60	46.84	49.03	58.21	64.27

Method	LLaMA2	CodeLLaMA	Qwen-2.5	LLaMA-3.2
P-CoT SFT	46.04	47.04	48.29	41.56
IR. + P-CoT	46.20	47.23	48.91	41.87
P-CoT + PC.	46.54	46.86	48.04	42.43
Parrot SFT	46.73	49.03	50.53	44.42

Table 3: The results of **IR.** ablation experiments. We use Qwen-2.5-1.5B and LLaMA-3.2-3B. Compared with Parrot SFT, P-CoT + PC. omits the IR. subtask.

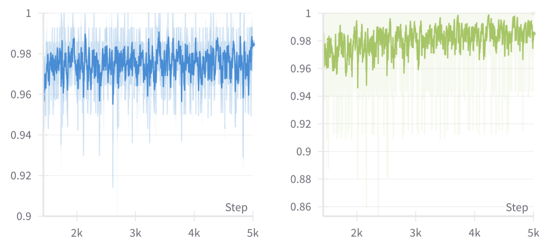

Figure 3: The training accuracy of LLaMA-2-7B on SVAMP for P-CoT RL. The left figure is without the converted penalty while the right is with the penalty.

to enable P-CoT's variable definitions. However, we find it has minimal impact on less challenging datasets such as SVAMP and GSM8K, where P-CoT has done well, information retrieval risks misleading for repetitive information, there by we only applied this subtask on MathQA.

Further, another phenomenon we observe is that although including IR. subtask achieves limited improvements compared to pure P-CoT SFT (47.04 to 47.23 on CodeLLaMA, 46.04 to 46.20 on LLaMA2) on the MathQA dataset, after Parrot training, the stronger model CodeLLaMA gains a notable improvement compared to without IR. (P-CoT + PC. w/ im 46.86 to Parrot SFT 49.03, 2.17). We envision that the model learns to identify whether the key information is accurate and how to utilize it from the transferability of subtask hybrid training, and ultimately generates better quality P-CoT. We also conduct experiments on advanced models Qwen-2.5-1.5B, LLaMA-3.2-3B, and reach similar conclusions. The results can be found in Table 3. 2) For **P-CoT Reasoning**, the results of removing this subtask is IR. + N-CoT for MathQA and N-CoT for GSM8K, SVAMP. Compared to Parrot, this causes a significant performance degradation on LlaMA2 with 22.56 on MathQA, 16.76 on GSM8K, and similar trends on CodeLLaMA. To further explore this notable degradation, we introduce two different settings in the 3) **Paradigms Conversion** ablation: with and without P-CoT's intermediate results. We are intrigued to find that intermediate results prove essential. The model learns concise reasoning from P-CoT intermediate steps, with the improvements of 11.38, 14.48, and 18.97, 17.56 on GSM8K and MathQA N-CoT for LLaMA2 and CodeLLaMA.

Hybrid Training. Inspired by (Yue et al., 2024), we apply hybrid training to expect semantic transfer from different linguistic solutions for mutual enhancement, especially on P-CoT with the explicit thinking process (Lin et al., 2024). Compared to baseline P-CoT, P-CoT + PC. consistently outperforms, achieving +1.1 on GSM8K and +2.1 on SVAMP, but shows a slight (-0.18) decline on MathQA. We hypothesize that while this enhances semantic diversity, it also interferes with P-CoT's precise variable definition. By integrating the information retrieval subtask, Parrot SFT yields a 1.99 improvement over P-CoT SFT on MathQA, validating our hypothesis.

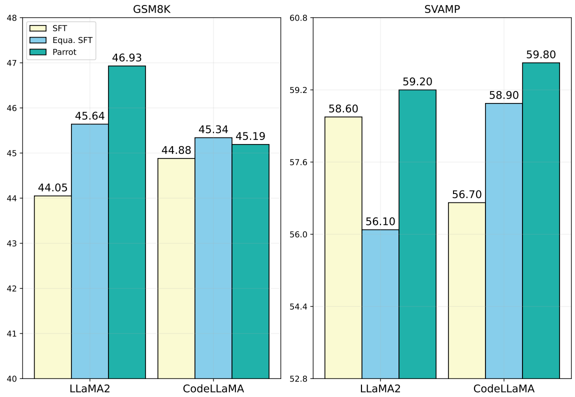

Figure 4: The results of performing SFT using the original N-CoT and the converted N-CoT data. In the left, SFT represents the original data size, while Equ. SFT refers to randomly expanding data to match the scale of the On-SL collections. Parrot denotes the collected data. We collect the correct N-CoT data from 3 epoch Parrot On-SL and perform supervised training after deduplicating.

5.2 The impact of the N-CoT penalty in P-CoT PPO.

Figure 3 shows the training accuracy of LLaMA-2-7B on SVAMP for P-CoT PPO. SVAMP is a relatively simple task, allowing the model to produce reasonably good samples and achieve high rewards during the early exploration stages. Without the penalty signal related to P-CoT quality, the model tends to fall into the trap of suboptimal overfitting. In contrast, as shown in the right figure, the model exhibits steady and continuous improvement.

5.3 Error Reduction and N-CoT Quality Gains.

P-CoT Alleviates Computational Errors and Logical Inconsistencies in N-CoT. As presented in section 2, we also statistically analyzed the error types of N-CoT after Parrot SFT. The comparison results are shown in the top of the Figure 1. Besides the significant reduction in computational errors (From 578 to 445 on LLaMA2, from 699 to 494 on CodeLLaMA), logical inconsistencies also significantly decreased, particularly on LLaMA2 (From 403 to 171), where P-CoT's intermediate results serve as a simple and effective process signal guiding the N-CoT reasoning. On one hand, we hope to use intermediate results to alleviate the calculation error of N-CoT. On the other hand, the intermediate variables in P-CoT are often linked to context, helping alleviate logical inconsistencies, and the inference process is provided in Appendix A.3. Due to incremental training on the code corpus, the decline in CodeLLaMA is relatively slight.

Better N-CoT training data obtained from P-CoT. We collect the converted N-CoT during 3 epoch Parrot On-SL training as the model ceases to merely generate high-quality data after several epochs due to the limited efficacy in exploration (Tao et al., 2024). We perform SFT and report the best epoch results. The results are in Figure 4. For fairness, we also randomly expand the original data to match the scale of the collected data. We observe two intriguing findings: 1) In the left, data expansion improves the efficacy on GSM8K, resulting in a performance gain of 1.59 on LLaMA2 and 0.46 on CodeLLaMA. However, for SVAMP, performance degrades with reductions of 2.5 on LLaMA2, which may be due to overfitting for its simplicity. 2) The performance of N-CoT obtained from P-CoT consistently exceeds the original data, even with no evidence of overfitting on SVAMP, which further demonstrates that with the aid of P-CoT, the model generates high-quality N-CoT.

Training Methods	N-CoT	P-CoT
LLaMA-3-8B + SFT	39.13	48.54
LLaMA-3-8B + Parrot SFT	52.03	50.28
LLaMA-3.2-3B + SFT	31.93	41.56
LLaMA-3.2-3B + Parrot SFT	41.00	44.42
Qwen-2.5-1.5B + SFT	32.15	48.29
Qwen-2.5-1.5B + Parrot SFT	47.58	50.53

Table 4: The Parrot results on the MathQA dataset with three different models.

5.4 The applicability of Parrot.

We additionally introduce some up-to-date works, MathGenie(Lu et al., 2024a), ToRA (Gou et al., 2023), DotaMath (Li et al., 2024a), MARIO (Liao et al., 2024). The performance of our model is generally close to or slightly lower than these results (74.53 vs 71.7, 72.6, 79.6, 78.4). Considering that they either used more data or a stronger base model, this gap is relatively acceptable and proves the data efficiency of Parrot.

To verify Parrot's versatility, we also apply the Parrot pipeline to LLaMA-3-8B, LLaMA-3.2-

3B, and Qwen-2.5-1.5B on the MathQA dataset, which is more challenging than GSM8K (Luong et al., 2024) and where we integrate all subtasks. The consistent improvements across different model sizes and families from Table 4 indicate Parrot has the broad applicability, and consistent with previous experiments, the improvement of N-CoT is significant.

6 Related Work

Mathematical Reasoning through CoT. Significant progress has been made in mathematical reasoning using large language models (LLMs) through chain-of-thought prompting (Wei et al., 2022) recently. Specifically, Fu et al. (2022) introduced the concept of complexity-based prompting, demonstrating that LLMs tend to favor long reasoning chains, which often lead to better performance. Recent works such as (Guo et al., 2025) and (Team et al., 2025) have also verified the contribution of long thought chains to reasoning ability. Despite these significant advancements, ensuring the correctness of the chain of thought remains a challenge.

Design of CoT in Mathematical Reasoning. Due to the difficulty in verifying the correctness of the CoT in natural language, a large number of studies have focused on the design of the CoT in mathematical reasoning. The determinacy of programming languages has made the program-assisted method a powerful tool for LLMs to solve mathematical problems. Chen et al. (2022) has developed a strategy to ensure the consistency of answers between program CoT and natural language CoT, aiming to enhance the reliability of the CoT. Similarly, Gao et al. (2023) executes tasks through a Python interpreter to mitigate calculation errors in the natural language CoT. Jie et al. (2023) has conducted a comprehensive analysis and comparison of the thought chains in natural language CoT and program CoT, revealing their unique characteristics and potential advantages.

Exploration of Mathematical Reasoning Paradigms. Specifically for solving math problems using LLMs, the main training paradigms revolve around Supervised Fine-Tuning (SFT), Reinforcement Learning (RL), and re-ranking.Uesato et al. (2022) and Lightman et al. (2023) trained an outcome-based or process-based reward model to perform reranking (Cobbe et al., 2021), attaining significantly superior performance compared to the methods of supervised fine-tuning (SFT) and majority voting (Wang et al., 2022). Luong et al. (2024) and Guo et al. (2025)further enhanced the generalization ability of LLMs in problem-solving through RL.

7 Conclusion and Future Work.

In this paper, we conduct a detailed analysis of error types of P-CoT and N-CoT paradigms and seek to merge the benefits of these two paradigms for mutual promotion, based on which we propose **Parrot**, a novel training pipeline that integrates three target-designed subtasks for the sequential P-CoT and N-CoT generations. We employ a hybrid training strategy to enhance transferability across pipeline subtasks and analyze the impact of each sub-task in detail. We further expand the pipeline with search and learning algorithms and introduce a converted N-CoT reward to alleviate the sparse issue in the P-CoT RL phase. Extensive experiments demonstrate that Parrot can simultaneously improve both P-CoT and N-CoT performance, especially on N-CoT. In the future, we plan to apply the Parrot pipeline to other reasoning domains such as math proving (Lin et al., 2024).

Limitations

This study has several limitations. First, the proposed sub-task hybrid training strategy demonstrates high sensitivity to data distribution, requiring carefully balanced datasets for optimal performance. Additionally, the resource-intensive search and learning algorithms necessitate substantial computational resources, with model initialization playing a critical yet potentially understudied role in multi-task scenarios. Second, our study focuses solely on mathematical reasoning, leaving other critical reasoning domains (e.g., logical, scientific, and ethical reasoning) unexplored, which limits the broader applicability of our methodology. Furthermore, we did not conduct experiments on the complex MATH dataset (Hendrycks et al., 2021) for several reasons: 1) Most MATH problems and solutions are written in LaTeX, which highlights the limitations of natural language in key information retrieval and resolution conversion. 2) The limited problems available for both P-CoT and N-CoT made it difficult for models to generate P-CoT. Instead, we conducted experiments on the MathQA dataset, which presents problems in natural language, using the LLaMA3 and Qwen-2.5 series models. Future research could investigate more stable training paradigms and expand the research framework to include additional cognitive tasks, enhancing the robustness and broader applicability of our approach.

Ethical Considerations

This research employs closed-source models for data synthesis and fine-tunes open-source models to enhance mathematical reasoning. We adhere to ACL's ethical policies and have rigorously checked the data to mitigate ethical and privacy concerns. Responsible use of LLMs is emphasized to avoid harmful or biased outputs. Reinforcement learning was also employed, which may lead to high resource consumption and environmental impacts.

Acknowledgements

The authors wish to thank the anonymous reviewers for their helpful comments. This work was partially funded by the Science and Technology Commission of Shanghai Municipality (No. 24511103100), National Natural Science Foundation of China (No.62476061,62206057), Shanghai Rising-Star Program (23QA1400200), and Natural Science Foundation of Shanghai (23ZR1403500).

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 2357–2367. Association for Computational Linguistics.

Thomas Anthony, Zheng Tian, and David Barber. 2017. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5360–5370.

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*.

Wenxiang Chen, Wei He, Zhiheng Xi, Honglin Guo, Boyang Hong, Jiazheng Zhang, Rui Zheng, Nijun Li, Tao Gui, Yun Li, et al. 2025. Better process supervision with bi-directional rewarding signals. *arXiv preprint arXiv:2503.04618*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *Cornell University - arXiv,Cornell University - arXiv*.

Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. 2022. Complexity-based prompting for multi-step reasoning. In *The Eleventh International Conference on Learning Representations*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Tora: A tool-integrated reasoning agent for mathematical problem solving. *arXiv preprint arXiv:2309.17452*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Sylvain Gugger, Lysandre Debut, Thomas Wolf, Philipp Schmid, Zachary Mueller, Sourab Mangrulkar, Marc Sun, and Benjamin Bossan. 2022. Accelerate: Training and inference at scale made simple, efficient and adaptable. https://github.com/huggingface/accelerate.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

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. *arXiv preprint arXiv:2103.03874*.

Steven CH Hoi, Doyen Sahoo, Jing Lu, and Peilin Zhao. 2021. Online learning: A comprehensive survey. *Neurocomputing*, 459:249–289.

Zhanming Jie, Trung Quoc Luong, Xinbo Zhang, Xiaoran Jin, and Hang Li. 2023. Design of chain-of-thought in math problem solving. *arXiv preprint arXiv:2309.11054*.

Farhad Kazemi, Mozafar Yektayar, and Ali Mohammadi Bolban Abad. 2012. Investigation the impact of chess play on developing meta-cognitive ability and math problem-solving power of students at different levels of education. *Procedia-Social and Behavioral Sciences*, 32:372–379.

Jennifer L Krawec. 2014. Problem representation and mathematical problem solving of students of varying math ability. *Journal of Learning Disabilities*, 47(2):103–115.

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

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D. Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M. Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal M. P. Behbahani, and Aleksandra Faust. 2024. Training language models to self-correct via reinforcement learning. *CoRR*, abs/2409.12917.

Komal Kumar, Tajamul Ashraf, Omkar Thawakar, Rao Muhammad Anwer, Hisham Cholakkal, Mubarak Shah, Ming-Hsuan Yang, Phillip HS Torr, Fahad Shahbaz Khan, and Salman Khan. 2025. Llm post-training: A deep dive into reasoning large language models. *arXiv preprint arXiv:2502.21321*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Chengpeng Li, Guanting Dong, Mingfeng Xue, Ru Peng, Xiang Wang, and Dayiheng Liu. 2024a. Dotamath: Decomposition of thought with code assistance and self-correction for mathematical reasoning. *arXiv preprint arXiv:2407.04078*.

Long Li, Xuzheng He, Haozhe Wang, Linlin Wang, and Liang He. 2024b. How do humans write code? large models do it the same way too. *arXiv preprint arXiv:2402.15729*.

Xiaoyuan Li, Wenjie Wang, Moxin Li, Junrong Guo, Yang Zhang, and Fuli Feng. 2024c. Evaluating mathematical reasoning of large language models: A focus on error identification and correction. *arXiv preprint arXiv:2406.00755*.

Zhenwen Liang, Ye Liu, Tong Niu, Xiangliang Zhang, Yingbo Zhou, and Semih Yavuz. 2024. Improving llm reasoning through scaling inference computation with collaborative verification. *arXiv preprint arXiv:2410.05318*.

Minpeng Liao, Wei Luo, Chengxi Li, Jing Wu, and Kai Fan. 2024. Mario: Math reasoning with code interpreter output–a reproducible pipeline. *arXiv preprint arXiv:2401.08190*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Haohan Lin, Zhiqing Sun, Sean Welleck, and Yiming Yang. 2024. Lean-star: Learning to interleave thinking and proving. *arXiv preprint arXiv:2407.10040*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Zimu Lu, Aojun Zhou, Houxing Ren, Ke Wang, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2024a. Mathgenie: Generating synthetic data with question back-translation for enhancing mathematical reasoning of llms. *arXiv preprint arXiv:2402.16352*.

Zimu Lu, Aojun Zhou, Ke Wang, Houxing Ren, Weikang Shi, Junting Pan, Mingjie Zhan, and Hongsheng Li. 2024b. Mathcoder2: Better math reasoning from continued pretraining on model-translated mathematical code. *arXiv preprint arXiv:2410.08196*.

Liangchen Luo, Yinxiao Liu, Rosanne Liu, Samrat Phatale, Meiqi Guo, Harsh Lara, Yunxuan Li, Lei Shu, Yun Zhu, Lei Meng, et al. 2024. Improve mathematical reasoning in language models by automated process supervision. *arXiv preprint arXiv:2406.06592*.

Trung Quoc Luong, Xinbo Zhang, Zhanming Jie, Peng Sun, Xiaoran Jin, and Hang Li. 2024. Reft: Reasoning with reinforced fine-tuning. *arXiv preprint arXiv:2401.08967*.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

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*.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: Memory optimizations toward training trillion parameter models. In *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–16.

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*, pages 3505–3506.

Matthew Renze and Erhan Guven. 2024. Self-reflection in LLM agents: Effects on problem-solving performance. *CoRR*, abs/2405.06682.

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. *CoRR*, abs/2308.12950.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Zhaochen Su, Jun Zhang, Xiaoye Qu, Tong Zhu, Yanshu Li, Jiashuo Sun, Juntao Li, Min Zhang, and Yu Cheng. 2024. Conflictbank: A benchmark for evaluating the influence of knowledge conflicts in llms. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. 2024. A survey on self-evolution of large language models. *CoRR*, abs/2404.14387.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. 2025. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*.

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, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Jonathan Uesato, Nate Kushman, Ramana Kumar, H. Francis Song, Noah Y. Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. 2022. Solving math word problems with process- and outcome-based feedback. *CoRR*, abs/2211.14275.

Ziyu Wan, Xidong Feng, Muning Wen, Stephen Marcus McAleer, Ying Wen, Weinan Zhang, and Jun Wang. 2024. Alphazero-like tree-search can guide large language model decoding and training. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

Peiyi Wang, Lei Li, Zhihong Shao, Runxin Xu, Damai Dai, Yifei Li, Deli Chen, Yu Wu, and Zhifang Sui. 2024. Math-shepherd: Verify and reinforce llms step-by-step without human annotations. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 9426–9439. Association for Computational Linguistics.

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

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Zhiheng Xi, Wenxiang Chen, Boyang Hong, Senjie Jin, Rui Zheng, Wei He, Yiwen Ding, Shichun Liu, Xin Guo, Junzhe Wang, et al. 2024. Training large language models for reasoning through reverse curriculum reinforcement learning. *arXiv preprint arXiv:2402.05808*.

Zhiheng Xi, Senjie Jin, Yuhao Zhou, Rui Zheng, Songyang Gao, Jia Liu, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Self-polish: Enhance reasoning in large language models via problem refinement. In *Findings of the Association for*

Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023, pages 11383–11406. Association for Computational Linguistics.

Haoran Xu, Young Jin Kim, Amr Sharaf, and Hany Hassan Awadalla. 2024a. A paradigm shift in machine translation: Boosting translation performance of large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Rongwu Xu, Zehan Qi, Zhijiang Guo, Cunxiang Wang, Hongru Wang, Yue Zhang, and Wei Xu. 2024b. Knowledge conflicts for llms: A survey. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 8541–8565. Association for Computational Linguistics.

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

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. Mammoth: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Yu Zhang and Qiang Yang. 2021. A survey on multi-task learning. *IEEE transactions on knowledge and data engineering*, 34(12):5586–5609.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A Preliminary Errors and Prompts

A.1 Preliminary Error Identifications

We randomly sampled 50 error cases from each paradigm and the empirical identifications of error types are as follows:

N-CoT Error Identifications:

- **Comprehension Error**: Misunderstanding of the problem, omission of conditions.
- **Logical Inconsistency**: Logic inconsistency between pre-and-post during the reasoning process.
- **Redundant and Repetitive**: Unnecessary information or overlapping functions, whereas repetitiveness refers to patterns that add little or no substantial value.
- **Calculation Error**: Basic arithmetic errors and improper application of formulas.
- **Other Errors**: Other reasoning errors fall outside the scope of the above.

P-CoT Error Identifications:

- **Comprehension Error**: Misunderstanding of the problem, omission of conditions.
- **Reasoning Error**: Inadequate reasoning, causal inversion, and circular arguments.
- **Variable Error**: Incorrect definition and assignment of variables.
- **Expression Error**: Violations of mathematical operation rules and non-standard Python output.

A.2 Error Evaluation Prompt

Based on manually verified error types, we use GPT-4 (OpenAI, 2023) to identify errors in the model's N/P-CoT rationales. The system prompt provided to GPT-4 is:

GPT-4 Evaluation System Prompt

System Prompt: You are a helpful assistant. Analyze the following answer reasoning process, identify the major error in it.
Types of errors: {N/P-CoT Error Identification Types}.
Please analyze the major type of error that may occur.
Don't output the explanation. Output the error type directly in the format: **The error type is: {}**.
User: {**Question**}'s ground truth answer is {**Answer Value**}.
Answer reasoning: {**Answer Reasoning Process**}.

The details of **N/P-CoT Error Identification Types** can be found in 2, while the **Answer Reasoning Process** refers to the model's inference outputs.

A.3 Subtask Prompts and Inference Process.

In this section, we provide the prompts we used and the inference process details of key information, P-CoT, and N-CoT, which align with the three subtasks.

Subtask Prompts:

- **System Prompt**: Question:
- **Information Retrieval Prompt**: Answer reasoning: To solve this question, we first find all the key information in the question:
- **P-CoT Reasoning Prompt**: Please refer to the key information to complete the Python-style solution:
- **Paradigm Conversion Prompt**: Please refer to the Python code style solution and the intermediate outputs to complete the natural language style solution. Therefore, the natural language style solution is:

Subtask Subtasks: We divide Parrot Pipeline into three sub-tasks: **Information Retrieval subtask**, **P-CoT Reasoning subtask**, **Paradigm Conversion subtask**.

Information Retrieval subtask

The input is:
{System Prompt}
Question
{Information Retrieval Prompt}

The output is:
Key information

P-CoT Reasoning subtask

The input is:
{System Prompt}
Question
{Information Retrieval Prompt}
Key information
{P-CoT Reasoning Prompt}

The output is:
P-CoT Reasoning Process

The P-CoT reasoning solution is executed with a Python interpreter, and we get the **P-CoT intermediate outputs**. The format of P-CoT intermediate outputs is: **variable_name1 = xxx, variable_name2 = xxx**, etc. **Note** during the inference phase, for unexecutable P-CoT or variables without specific values, we use their variable names as intermediate results. In this time, the format of P-CoT intermediate outputs is: **variable_name1 = xxx, variable_name2, variable_name3 = xxx**, etc.

Paradigm Conversion subtask

The input is:
{System Prompt}
Question
{Information Retrieval Prompt}
Key information
{P-CoT Reasoning Prompt}
P-CoT Reasoning Process
The python solution's intermediate outputs are: {P-CoT intermediate outputs}
{Paradigm Conversion Prompt}

The output is:
N-CoT Reasoning Process

B Training and implementation details

We conduct all experiments with eight A100-80GB GPUs, and using DeepSpeed Zero stage 2 (Rajbhandari et al., 2020; Rasley et al., 2020), Huggingface accelerate (Gugger et al., 2022) framework. We use AdamW (Loshchilov and Hutter, 2019) optimizer and set eps to 1e-8.

Hyper parameters The maximum input length is set to 1024, while the maximum output length is 700. In SFT, the learning rate is 1e-5, the train batch size is 32, and no warm-up stage. We train models for 5 epochs, except for MathQA, for 10 epochs, and report the best performance. In RL, we use the best-initialized models in the SFT stage, the train batch size is 24, and we employ LORA in P-CoT RL and On-SL experiments for efficient and set α 64, r 32. We set the policy and value learning rate 3e-7 for GSM8K and SVAMP and 1e-8 for MathQA. We set the partial reward ϵ to 0.1 and the convert penalty γ to 0.2. The KL constraint coefficient β is set to 0.05 for N-CoT experiments and 0.01 for P-CoT experiments. We train the model for 100 epochs and report the best performance. The discount factor and smoothing coefficient of GAE in PPO algorithm and the remaining details, we adhere to the settings in (Luong et al., 2024).

C Dataset construction and sizes

The main details of datasets we used in this paper are presented in Table 5. The P-CoT annotations are derived from (Luong et al., 2024). We use regular match (e.g., re module in Python) for key information annotations based on P-CoT annotations according to the following principles:

- Due to the authenticity of P-CoT annotations, the variable names in the reasoning process signify their importance, which we use to identify relevant sentences in the question.

- Sentences containing numbers and operators in the question.

- If none of the above is available, we will take the questions that do not contain the question part as the key information.

After completing these steps, we performed manual validation.

For N-CoT annotations, we execute P-CoT and print the intermediate results as described in A.3, then we use the GPT-4 (OpenAI, 2023) for generation with the following prompts:

N-CoT Annotation Generation Prompt

I will give you a math problem and a Python code:
{P-CoT Reasoning Process}
that solves this problem, along with the intermediate result information:
{Intermediate Results}
from this code. Please refer to the intermediate result information and python code to generate a natural language solution. The final answer should be given in the format **The answer is <answer>**.

For example, if the final answer is 10, you should output **The answer is 10**. You don't need to output anything else; just output the natural language solution.

We compared the final answer with the ground truth. For incorrect ones, after three attempts, those still unresolved were manually corrected. For all GPT-4-generated data, we used greedy decoding.

Category	Dataset	#Train Samples	#Test Samples
Math Reasoning	GSM8K (Cobbe et al., 2021)	7321	1319
	MathQA (Amini et al., 2019)	5706	1605
	SVAMP (Patel et al., 2021)	3052	1000

Table 5: Statistics of the experiments datasets.

D Examples

D.1 Examples of Error Types

In Section 2, we provide a detailed analysis of error types in N-CoT and P-CoT reasoning processes. For N-CoT, the predominant errors are comprehension errors (**CE**), calculation errors (**CA**), and logic inconsistencies (**LI**). For P-CoT, the main categories are reasoning errors (**RE**) and variable definition errors (**VE**).

N-CoT Reasoning Error Types

```
Comprehension Errors (CE)

Question: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts
in total does it take?
Ground true answer: 3

N-CoT SFT model output:
Let's start by finding out how many bolts of white fiber the robe takes.
The white robe takes: bolts of white fiber = 2 bolts of blue fiber.
Now, we can add the number of bolts of blue and white fiber together to find the total
number of bolts the robe takes 2 bolts of blue fiber + 4 bolts of white fiber = 6 bolts in
total.
Therefore, the answer is 6.
```

```
Calculation Error (CA)

Question: A new program had 60 downloads in the first month. The number of downloads in the
second month was three times as many as the downloads in the first month, but then reduced
by 30% in the third month. How many downloads did the program have total over the three
months?
Ground true answer: 366

N-CoT SFT model output:
Let's break down the problem into three parts: First month: 60 downloads. Second month: 3
times as many as the downloads in the first month, which is 3 x 60 = 180 downloads.
Third month: 30% reduction in the number of downloads,
which is 180 - (30/100 x 180) = 136 downloads.
To find the total number of downloads over the three months, we need to add up the number of
 downloads in each month: 60 + 180 + 136 = 376
Therefore, the answer is 376.
```

```
Logic Inconsistency (LI)

Question: a train 100 meters long completely crosses a 300 meters long bridge in 45 seconds.
 What is the speed of the train is?
Ground true answer: 32

N-CoT SFT model output:
To solve this problem, we need to use the formula:
Speed = Distance / Time
First, we need to convert the time from seconds to meters:
45 seconds = 45 x 100 meters = 4500 meters
300 meters long bridge = 300 meters
Now we can plug in the values:
Speed = 4500 meters / 300 meters = 15 meters per second
Therefore, the answer is 15.
```

Figure 5: The examples and analysis of Comprehension Error (**CE**), Calculation Error (**CA**) and Logic Inconsistency (**LI**) in N-CoT SFT model reasoning outputs.

P-CoT Reasoning Error Types

```
Reasoning Error (RE)

Question: a shopkeeper sold an article offering a discount of 5 % and earned a profit of
31.1 %. what would have been the percentage of profit earned if no discount had been offered
?
Ground true answer: 38

P-CoT SFT model output:
def solution():
    discount = 5
    profit = 31.1
    profit_without_discount = profit * (100 - discount) / 100
    result = profit_without_discount
    return result
```

```
Variable Definition Error (VE)

Question: an art gallery has only paintings and sculptures. currently, 1 / 3 of the pieces
of art are displayed, and 1 / 6 of the pieces on display are sculptures. if 1 / 3 of the
pieces not on display are paintings, and 1000 sculptures are not on display, how many pieces
 of art does the gallery have?
Ground true answer: 2250

P-CoT SFT model output:
def solution():
    pieces_of_art = 1 / 3
    piece_on_display = 1 / 6
    pieces_not_on_display = 1 - pieces_on_display
    sculptures_not_on_display = 1000
    pieces_total = pieces_of_art +  pieces_on_display + sculptures_not_on_display
    result = pieces_total
    return result
```

Figure 6: The examples and analysis of Reasoning Error (**RE**) and Variable Definition Error (**VE**) in P-CoT SFT model reasoning outputs.

D.2 Examples Showcasing Gains from Parrot Training

As shown in Section 2 and the Examples of Error Types D.1, N-CoT errors are primarily attributed to calculation errors and logical inconsistencies, while P-CoT errors largely arise from reasoning errors—issues that can be alleviated through Parrot hybrid training. In contrast, the intermediate results of P-CoT can mitigate the errors of N-CoT.

Examples Showcasing Gains from Parrot Training

```
Question: Brian's friend Bobby has 5 fewer than 3 times as many video games as Brian does.
If Brian has 20 video games but lost 5 right before the comparison was made, how many does
Bobby have?
Ground truth answer:40
```

N-CoT SFT Model output

```
N-CoT SFT Model output:
We know that Bobby has 5 fewer than 3 times as many video games as Brian does.
If Brian has 20 video games, then Bobby has:
3 x 20 = 60 video games, so Bobby has 60 video games.
Now we need to find out how many video games Brian had before he lost 5.
If Brian had 20 video games and lost 5, then he had:20 - 5 = 15 video games.
Therefore, Bobby has 60 - 5 = 55 video games.
The answer is: 55.
```

Parrot SFT Model output

```
P-CoT Parrot SFT Model output:
def solution():
    brian_games_initial = 20
    brian_games_lost = 5
    brian_games_final = brian_games_initial - brian_games_lost
    bobby_games_initial = brian_games_final * 3 - 5
    bobby_games_final = bobby_games_initial
    result = bobby_games_final
    return result

Parrot SFT P-CoT intermediate results: brian_games_final = 15, bobby_games_initial = 40.
```

```
N-CoT Parrot SFT Model output:
Let's start by finding out how many video games Brian has after losing 5.
Brian had 20 video games, but lost 5, so he now has 20 - 5 = 15 video games.
Next, we need to find out how many video games Bobby has. We know that Bobby has 5 fewer
than 3 times as many video games as Brian, so we can set up an equation:
Bobby's video games = 3 x Brian's video games - 5.
Substituting in the value we found for Brian's video games:
Bobby's video games = 45-5=40
Therefore, Bobby has 40 video games.
The answer is 40.
```

Figure 7: The example and improvement analysis of the Parrot training pipeline. While the N-CoT SFT model exhibits logical errors, the guidance of P-CoT and its intermediate results ultimately led the model to correct the N-CoT solution.