Training Setting	Training Tokens			w/o Tool Use			w/ Tool Use	
	General	Code	Math	GSM8K	MATH	CMATH	GSM8K+Python	MATH+Python
No Continual Training	–	–	–	2.9%	3.0%	12.3%	2.7%	2.3%
Two-Stage Training								
Stage 1: General Training	400B	–	–	2.9%	3.2%	14.8%	3.3%	2.3%
Stage 2: Math Training	–	–	150B	19.1%	14.4%	37.2%	14.3%	6.7%
Stage 1: Code Training	–	400B	–	5.9%	3.6%	19.9%	12.4%	10.0%
Stage 2: Math Training	–	–	150B	**21.9%**	**15.3%**	**39.7%**	17.4%	9.4%
One-Stage Training								
Math Training	–	–	150B	20.5%	13.1%	37.6%	11.4%	6.5%
Code & Math Mixed Training	–	400B	150B	17.6%	12.1%	36.3%	**19.7%**	**13.5%**

Table 6 | Investigation of how code affects mathematical reasoning under different training settings. We experiment with DeepSeek-LLM 1.3B, and evaluate its mathematical reasoning performance without and with tool use via few-shot chain-of-thought prompting and few-shot program-of-thought prompting, respectively.

improves models' ability to do mathematical reasoning both with and without tool use.

To study how code training affects mathematical reasoning, we experimented with the following two-stage training and one-stage training settings:

Two-Stage Training

- **Code Training for 400B Tokens → Math Training for 150B Tokens**: We train DeepSeek-LLM 1.3B for 400B code tokens followed by 150B math tokens;
- **General Training for 400B Tokens → Math Training for 150B Tokens**: As a control experiment, we also experiment with general tokens (sampled from a large-scale general corpus created by DeepSeek-AI) instead of code tokens in the first stage of training, in an attempt to investigate the advantages of code tokens over general tokens in improving mathematical reasoning.

One-Stage Training

- **Math Training for 150B Tokens**: We train DeepSeek-LLM 1.3B for 150B math tokens;
- **Training on a mixture of 400B Code Tokens and 150B Math Tokens**: Math training following code training degrades coding performance. We investigate whether code tokens, when mixed with math tokens for one-stage training, would still improve mathematical reasoning and also alleviate the problem of catastrophic forgetting.

Results Table 6 and Table 7 demonstrate the downstream performance under different training settings.

Code training benefits program-aided mathematical reasoning, both under the two-stage training and one-stage training settings. As shown in Table 6, under the two-stage training setting, code training alone already significantly enhances the ability to solve GSM8K and MATH problems using Python. Math training in the second stage yields further improvements. Interestingly, under the one-stage training setting, mixing code tokens and math tokens effectively mitigates the issue of catastrophic forgetting that arises from two-stage training, and also synergizes coding (Table 7) and program-aided mathematical reasoning (Table 6).

Training Setting	Training Tokens			MMLU	BBH	HumanEval (Pass@1)	MBPP (Pass@1)
	General	Code	Math				
No Continual Training	–	–	–	24.5%	28.1%	12.2%	13.0%
Two-Stage Training							
Stage 1: General Training	400B	–	–	25.9%	27.7%	15.2%	13.6%
Stage 2: Math Training	–	–	150B	33.1%	32.7%	12.8%	13.2%
Stage 1: Code Training	–	400B	–	25.0%	31.5%	25.0%	**40.0%**
Stage 2: Math Training	–	–	150B	**36.2%**	35.3%	12.2%	17.0%
One-Stage Training							
Math Training	–	–	150B	32.3%	32.5%	11.6%	13.2%
Code & Math Mixed Training	–	400B	150B	33.5%	**35.6%**	**29.3%**	39.4%

Table 7 | Investigation of how different settings of code and math training affect model performance of language understanding, reasoning, and coding. We experiment with DeepSeek-LLM 1.3B. We evaluate the models on MMLU and BBH using few-shot chain-of-thought prompting. On HumanEval and MBPP, we conduct zero-shot and few-shot evaluations, respectively.

Model	Size	ArXiv Corpus	English Benchmarks					Chinese Benchmarks		
			GSM8K	MATH	OCW	SAT	MMLU STEM	CMATH	Gaokao MathCloze	Gaokao MathQA
DeepSeek-LLM	1.3B	No Math Training	2.9%	3.0%	2.9%	15.6%	19.5%	12.3%	0.8%	17.9%
		MathPile	2.7%	3.3%	2.2%	12.5%	15.7%	1.2%	0.0%	2.8%
		ArXiv-RedPajama	3.3%	3.4%	4.0%	9.4%	9.0%	7.4%	0.8%	2.3%
DeepSeek-Coder-Base-v1.5	7B	No Math Training	29.0%	12.5%	6.6%	40.6%	38.1%	45.9%	5.9%	21.1%
		MathPile	23.6%	11.5%	7.0%	46.9%	35.8%	37.9%	4.2%	25.6%
		ArXiv-RedPajama	28.1%	11.1%	7.7%	50.0%	35.2%	42.6%	7.6%	24.8%

Table 8 | Effect of math training on different arXiv datasets. Model performance is evaluated with few-shot chain-of-thought prompting.

ArXiv Corpus	miniF2F-valid	miniF2F-test
No Math Training	20.1%	21.7%
MathPile	16.8%	16.4%
ArXiv-RedPajama	14.8%	11.9%

Table 9 | Effect of math training on different arXiv corpora, the base model being DeepSeek-Coder-Base-v1.5 7B. We evaluate informal-to-formal proving in Isabelle.

Code training also improves mathematical reasoning without tool use. Under the two-stage training setting, the initial stage of code training already results in moderate enhancements. It also boosts the efficiency of the subsequent math training, eventually leading to the best performance. However, combining code tokens and math tokens for one-stage training compromises mathematical reasoning without tool use. One conjecture is that DeepSeek-LLM 1.3B, due to its limited scale, lacks the capacity to fully assimilate both code and mathematical data simultaneously.

5.1.2. ArXiv Papers Seem Ineffective in Improving Mathematical Reasoning

ArXiv papers are commonly included as a component of math pre-training data (Azerbayev et al., 2023; Lewkowycz et al., 2022a; Polu and Sutskever, 2020; Wang et al., 2023c). However, detailed analysis regarding their impact on mathematical reasoning has not been extensively conducted. Perhaps counter-intuitively, according to our experiments, arXiv papers seem ineffective in improving mathematical reasoning. We experiment with models of different sizes, including DeepSeek-LLM 1.3B and DeepSeek-Coder-Base-v1.5 7B (Guo et al., 2024), using arXiv corpora that underwent varied processing pipelines:

- **MathPile** (Wang et al., 2023c): an 8.9B-token corpus developed with cleaning and filtering heuristic rules, over 85% of which are scientific arXiv papers;
- **ArXiv-RedPajama** (Computer, 2023): the entirety of arXiv LaTeX files with preambles, comments, macros, and bibliographies removed, totaling 28.0B tokens.

In our experiments, we separately train DeepSeek-LLM 1.3B for 150B tokens and DeepSeek-Coder-Base-v1.5 7B for 40B tokens on each arXiv corpus. It seems that arXiv papers are ineffective in improving mathematical reasoning. When trained on a arXiv-only corpus, both models display no notable improvements or even deterioration across various mathematical benchmarks of different complexities employed in this study. These benchmarks include quantitative reasoning datasets like GSM8K and MATH (Table 8), multiple-choice challenges like MMLU-STEM (Table 8), and formal mathematics like miniF2F (Table 9).

However, this conclusion has its limitations and should be taken with a grain of salt. We have not yet studied:

- The impact of arXiv tokens on specific math-related tasks not included in this research, such as informalization of theorems which is to convert formal statements or proofs to their informal versions;
- The effect of arXiv tokens when combined with other types of data;
- Whether the benefits of arXiv papers would manifest themselves at a larger model scale.

Thus, further exploration is required, which we leave for future studies.

5.2. Insights of Reinforcement Learning

5.2.1. Towards to a Unified Paradigm

In this section, we provide a unified paradigm to analyze different training methods, such as SFT, RFT, DPO, PPO, GRPO, and further conduct experiments to explore the factors of the unified paradigm. Generally, the gradient with respect to the parameter θ of a training method can be written as:

$$\nabla_\theta \mathcal{J}_{\mathcal{A}}(\theta) = \mathbb{E}[\underbrace{(q, o) \sim \mathcal{D}}_{Data\ Source}] \left(\frac{1}{|o|} \sum_{t=1}^{|o|} \underbrace{GC_{\mathcal{A}}(q, o, t, \pi_{rf})}_{Gradient\ Coefficient} \nabla_\theta \log \pi_\theta(o_t | q, o_{<t}) \right). \tag{5}$$

There exist three key components: 1) *Data Source* $\mathcal{D}$, which determines the training data; 2) *Reward Function* π_{rf}, which is the source of the training reward signal; 3) *Algorithm* $\mathcal{A}$: which processes the training data and the reward signal to the gradient coefficient GC that determines the magnitude of the penalty or reinforcement for the data. We analyze several representative methods based on such a unified paradigm:

- **Supervised Fine-tuning (SFT)**: SFT fine-tunes pretrained model on human selected SFT data.