CoinMath: Harnessing the Power of Coding Instruction for Math LLMs

Chengwei Wei[♢], **Bin Wang**[♢], **Jung-jae Kim** [♢], **Guimei Liu** [♢], **Nancy F. Chen**[♢,†]
[♢]Institute for Infocomm Research (I^{2}R), A*STAR, Singapore
[†]Centre for Frontier AI Research (CFAR), A*STAR, Singapore
wei_chengwei@i2r.a-star.edu.sg

Abstract

Large Language Models (LLMs) have shown strong performance in solving mathematical problems, with code-based solutions proving particularly effective. However, the best practice to leverage coding instruction data to enhance mathematical reasoning remains underexplored. This study investigates **three key questions**: (1) How do different coding styles of mathematical code-based rationales impact LLMs' learning performance? (2) Can general-domain coding instructions improve performance? (3) How does integrating textual rationales with code-based ones during training enhance mathematical reasoning abilities? **Our findings** reveal that code-based rationales with concise comments, descriptive naming, and hardcoded solutions are beneficial, while improvements from general-domain coding instructions and textual rationales are relatively minor. Based on these insights, we propose CoinMath, a learning strategy designed to enhance mathematical reasoning by diversifying the coding styles of code-based rationales. CoinMath generates a variety of code-based rationales incorporating concise comments, descriptive naming conventions, and hardcoded solutions. Experimental results demonstrate that CoinMath significantly outperforms its baseline model, MAmmoTH, one of the SOTA math LLMs.[1]

1 Introduction

Large Language Models (LLMs) have been extensively applied to solving mathematical problems (Lewkowycz et al., 2022; Azerbayev et al., 2023; Luo et al., 2023; Ahn et al., 2024; Shao et al., 2024). Recently, the approach of using code-based solutions has achieved significant success, as the calculations are handled by a program interpreter, while the reasoning steps are effectively represented in code format. Prior studies on prompting, such as Program-of-Thoughts (PoT) (Chen et al., 2022) and Program-Aided Language models (PAL) (Gao et al., 2023a), have demonstrated that generating solutions in a code format significantly enhances LLM performance on mathematical tasks compared to Chain-of-Thoughts (CoT) which uses a text format (Wei et al., 2022). Furthermore, LLMs trained on mathematical instruction data with code-based rationales, reasoning steps presented in executable code, have shown strong performance (Wang et al., 2023a; Yue et al., 2023; Zhang et al., 2024a; Toshniwal et al., 2024).

Given the demonstrated effectiveness of mathematical instruction data with code-based rationales, there is growing interest in synthesizing additional mathematical problems to expand mathematical coding instruction datasets (Zhang et al., 2024a; Toshniwal et al., 2024), aiming to enhance the performance further. Despite this progress, the question of how such coding instruction data can be best leveraged to maximize its impact remains underexplored. Recently, Zhang et al. (2024b) showed that model instruction-tuned on text or a mix of text and code outperforms models trained solely on code in mathematical reasoning. However, the conclusions are drawn from evaluating the models by generating solutions in a text format instead of a code format. Bi et al. (2024) found that code-based rationales of medium complexity yield the best reasoning performance.

Taking this further, our study delves deeper into the effective utilization of existing coding instruction data to enhance mathematical reasoning. Specifically, we first conduct a comprehensive study on coding instruction to address three critical questions: 1) How do different coding styles (e.g., having a detailed comment or not) of mathematical code-based rationales impact LLMs' learning performance in mathematical reasoning? 2) Can coding instructions from general domains, beyond math, provide meaningful benefits under PoT rea-

[1]Our model, code, and datasets are open-sourced at https://github.com/amao0o0/CoinMath

Findings of the Association for Computational Linguistics: ACL 2025, pages 786–797
July 27 - August 1, 2025

soning? and 3) How does integrating textual rationales with code-based ones during instruction tuning enhance mathematical reasoning abilities?

Our findings can be summarized as follows:

- Mathematical code-based rationales that include concise comments and descriptive naming conventions are the most effective. Hardcoded rationales offer straightforward solutions that facilitate the model's learning.
- General domain coding instructions provide limited performance improvement compared to those from the mathematical domain.
- Adding textual rationales of math questions slightly enhances the performance of general-purpose models. In contrast, textual rationales do not benefit code-specialized models.

As the impact of coding instructions from general domains and textual rationales is minimal, while models exhibit varying behaviors toward code-based rationales with different coding styles, we propose **CoinMath**, short for **Co**ding **In**struction for **Math**, a learning strategy to effectively enhance LLMs' mathematical reasoning capabilities. CoinMath diversifies the coding styles of code-based rationales by incorporating advantageous coding attributes, including concise comments, descriptive naming, and hardcoded solutions. Experimental results demonstrate that CoinMath significantly outperforms the previous SOTA Math LLMs.

Our major contributions are as follows:

- We present a systematic study that investigates what makes diverse coding instructions effective, how they enhance the mathematical reasoning of LLMs, and why they work, uncovering the key factors behind their impact.
- Based on the above findings, we propose CoinMath, a learning strategy that improves LLMs' mathematical reasoning by incorporating code-based rationales with concise comments, descriptive naming conventions, and hardcoded solutions. Experimental results show that it outperforms the SOTA model by an average improvement of 5.9% in accuracy.
- We release the CoinMath model, curated datasets, and training and evaluation pipelines, enabling reproducibility and advancing research in mathematical reasoning for LLMs.

2 Related Work

Solving Math with Code. Chain-of-Thought (CoT) prompting (Nye et al., 2021; Wei et al., 2022), which guides LLMs to generate intermediate steps in textual rationales, has proven effective on reasoning tasks. Expanding on this foundation, Program-of-Thoughts (PoT) and Program-Aided Language models (PAL) (Chen et al., 2022; Gao et al., 2023a) have used code for reasoning. These methods prompt LLMs to generate reasoning steps as code and assign computations to external program interpreters, which has achieved significant success, particularly in mathematical reasoning tasks that involve complex calculations and logical operations. Subsequently, recent works (Toshniwal et al., 2024; Wang et al., 2023a; Yue et al., 2023; Gou et al., 2024) have constructed mathematical instruction datasets with code-based rationales and fine-tuned LLMs on them, further improving model performance on solving math problems.

Effect of Coding Instruction on Mathematical Reasoning. Several studies have explored coding instruction's impact on LLMs' mathematical reasoning abilities. Wang et al. (2023b) and Ma et al. (2024) indicated that coding instruction only enhances task-specific reasoning capabilities, which suggests non-math-specific data have minimal or even negative effects on mathematical reasoning. Zhang et al. (2024b) found that models trained on pure textual instructions or a combination of textual and coding instructions perform better in mathematical reasoning than those trained solely on coding instructions. However, the evaluations of these studies focused on CoT reasoning and generating textual solutions rather than code-based ones, which may limit the potential benefits of coding instructions. By contrast, Bi et al. (2024) evaluated models in PoT reasoning and revealed that mathematical code-based rationales with medium complexity lead to the most significant improvements. However, they did not comprehensively examine the effects of different coding styles, code domains, or the interplay between textual and code-based rationales in enhancing mathematical reasoning.

3 Study on Coding Instruction

This study investigates the effective utilization of coding instructions to enhance the mathematical reasoning abilities of LLMs. It focuses on three key questions. 1) Coding Style: Identifying the effect of different coding styles of mathematical code-

based rationales on LLMs' mathematical reasoning learning. 2) Code Domain: Exploring whether coding instructions from general (non-mathematical) domains can enhance performance in solving mathematical problems under PoT reasoning. 3) Integration with Textual Rationales: Evaluating the impact of combining mathematical textual rationales with code-based ones during training to enhance mathematical performance.

To explore the above questions, we collect instruction tuning datasets for three types: **Math-Text**, **Math-Code** and **General Code**. The Math-Text dataset, derived from MathInstruct (Yue et al., 2023), contains 26k math questions annotated with textual rationales. Similarly, Math-Code is another subset of MathInstruct, consisting of the same 26k math questions as Math-Text but annotated with code-based rationales. Finally, General Code comprises 21k Python coding instructions for non-math-specific tasks, curated from Code Alpaca (Chaudhary, 2023) and code generation tasks[2]. Detailed statistics for these datasets can be found in Appendix Table 7.

For evaluation datasets, we use four math Q&A datasets representing distinct mathematical perspectives: (1) **Arithmetic**: A dataset of arithmetic questions from MathBench (Liu et al., 2024). (2) **SVAMP** (Patel et al., 2021): A primary-school-level algebraic word problem dataset. (3) **GSM** (Cobbe et al., 2021): A middle-school-level algebraic word problem dataset. (4) **MATH** (Hendrycks et al., 2021): A challenging dataset that extends beyond the high school level and covers diverse topics, including algebra, precalculus, and number theory. Examples from each dataset are provided in Appendix Figure 5. While Arithmetic focuses solely on basic arithmetic, the difficulty of mathematical problems increases progressively from SVAMP to GSM, and finally to MATH. By default, we use PoT reasoning during evaluation unless otherwise specified as CoT reasoning.

Both general-purpose and code-specialized models are evaluated. The general-purpose models include Llama-3.1-8B (Dubey et al., 2024) and Gemma-2B (Gemma-Team, 2024), while the code-specialized models include CodeLlama-Python-7B (Roziere et al., 2023) and CodeGemma (CodeGemma-Team, 2024). The instruction tuning details are provided in Appendix D.

[2]https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca

3.1 Coding Style

What coding styles in code-based rationales are most effective for model learning? To address this question, we examine three key attributes of coding styles that may influence a model's mathematical reasoning capabilities: Comment Usage, Naming Convention, and Solution Generality. Previous studies have discussed the role of variable naming and comment usage in prompting design (Chen et al., 2022; Gao et al., 2023a). In contrast, we focus on the influence of the instruction tuning phase with three attributes including a newly introduced one called, Solution Generality. Figure 1 illustrates examples of different coding styles across these attributes. To generate rationales with diverse coding styles, we employ GPT-4o (OpenAI, 2024) to transform the original code-based rationales in the Math-Code dataset into variations reflecting diverse coding styles. Detailed instructions for generating these rationales are provided in Appendix C.

We focus on evaluating models in a zero-shot setting, as instruction-tuned models trained with datasets featuring different coding styles typically require few-shot examples that align with those styles, but a few-shot setting with different examples for different models may lead to biased evaluation results. Zero-shot evaluation allows us to isolate the effects of the instruction tuning data as the sole variable.

Comment Usage. We analyze the role of comment density in code-based rationales. Comments explain the overall logic of the rationales and the reasoning behind each step. We create three types of code-based rationales with varying comment usages: No Comment, Concise Comment, and Detailed Comment. In No Comment, the code stands alone, requiring the model to learn the code's func-

Model	Comment	A.	S.	G.	M.	Avg.
Llama-3.1-8B	No	61.7	30.4	32.4	11.5	34.0
	Concise	83.3	80.3	73.5	36.4	**68.4**
	Detailed	80.7	83.4	73.6	35.5	68.3
CodeLlama-Python-7B	No	82.7	64.4	52.2	27.3	56.7
	Concise	83.0	68.7	59.7	28.1	**59.9**
	Detailed	83.3	69.6	55.4	27.2	58.9
Gemma-2-2B	No	73.0	65.3	53.6	25.8	54.4
	Concise	77.3	67.6	54.7	26.6	**56.6**
	Detailed	79.0	67.2	54.0	24.4	56.2
CodeGemma-2B	No	79.0	53.1	40.9	21.4	48.6
	Concise	80.0	52.0	41.0	21.7	48.7
	Detailed	81.0	55.2	39.7	21.4	**49.3**

Table 1: Results with different Comment Usages. The best average performance for each model is highlighted in bold. A., S., G., and M. refer to the Arithmetic, SVAMP, GSM, and MATH datasets, respectively.

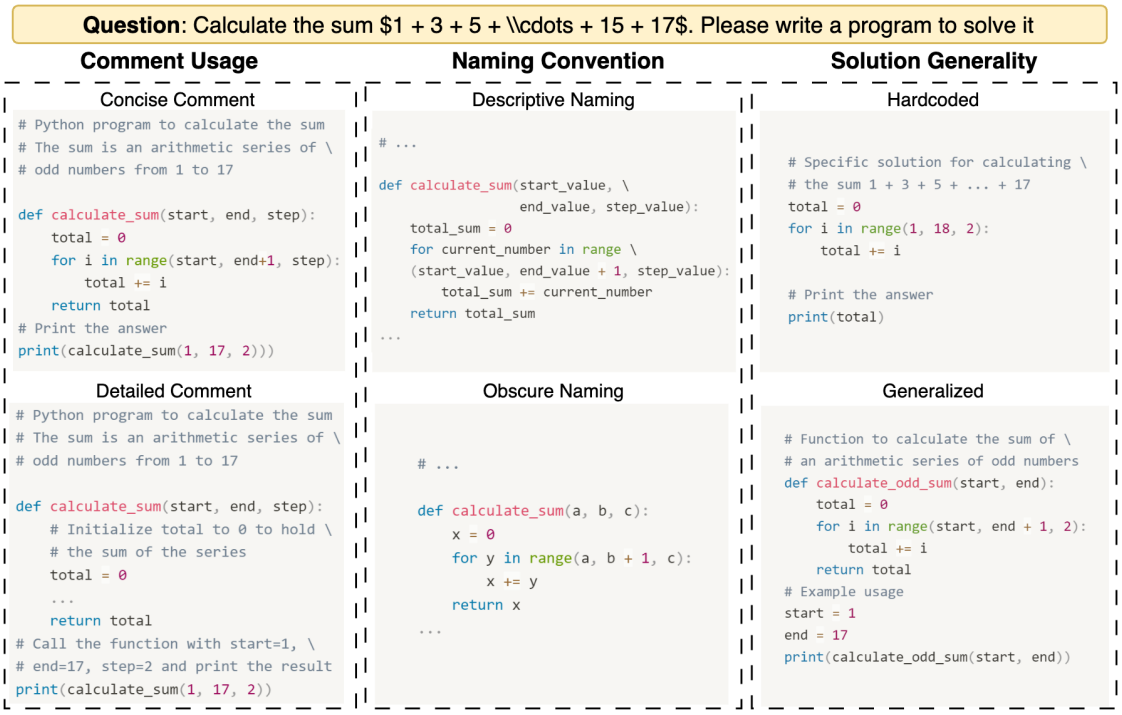


Figure 1: Exemplary code-based rationales in various coding styles (excluding No Comment in Code Notation). Certain lines are folded or omitted for improved visualization.

tionality and mathematical logic solely from the code. Concise Comment includes essential annotations that clarify key steps in math problem-solving. Detailed Comment rationales offer comprehensive annotations, providing line-by-line explanations of the code's functionality and the underlying mathematical logic. By examining these three types of comment usage, we can assess how the presence and density of comments in instruction data influence the model's mathematical reasoning capabilities.

The experimental results are summarized in Table 1. Compared with No Comment, both Concise Comment and Detailed Comment styles exhibit good overall performance, with Concise Comment slightly outperforming Detailed Comment. Additionally, the performance improvement of Concise Comment and Detailed Comment over No Comment is larger for general-purpose models than for the code-specialized models. This suggests that while both general-purpose and code-specific models benefit from comments, code-specialized models inherently have a strong ability to understand code, making comments less critical for them. In contrast, comments are relatively crucial for general-purpose models to better grasp the mathematical reasoning underlying the code.

Naming Convention. We identify naming conventions that best support models' mathematical reasoning learning. We design two types of naming conventions, namely, Descriptive Naming and Obscure Naming, while keeping other aspects of the code, such as comments and logic, unchanged. Descriptive Naming uses clear and meaningful names for all variables, whereas obscure naming employs short, non-descriptive names (e.g., single letters or random abbreviations).

Model	Naming	A.	S.	G.	M.	Avg.
Llama-3.1-8B	Descriptive	82.3	81.8	72.9	35.6	**68.2**
	Obscure	78.0	80.1	72.3	35.7	66.5
CodeLlama-Python-7B	Descriptive	80.3	69.8	58.3	28.0	**59.1**
	Obscure	79.0	66.1	55.5	28.0	57.2
Gemma-2-2B	Descriptive	74.7	65.3	57.8	26.5	**56.1**
	Obscure	73.7	62.3	55.0	25.9	54.2
CodeGemma-2B	Descriptive	74.0	57.3	40.9	21.6	**48.4**
	Obscure	72.0	54.8	39.7	21.4	47.0

Table 2: Results with different Naming Conventions.

As shown in Table 2, the results indicate that Descriptive Naming generally outperforms Obscure Naming. These findings are consistent with previous studies on naming conventions in prompting (Chen et al., 2022; Gao et al., 2023a). This further supports the idea that Descriptive Naming provides semantic context, helping models associate variable and function names with their intended purposes, which in turn enhances their understanding of mathematical problems.

Model	Generality	A.	S.	G.	M.	Avg.
Llama-3.1-8B	Hardcoded	83.3	78.9	73.5	34.8	**67.6**
	Generalized	78.0	75.1	72.7	29.0	63.7
CodeLlama-Python-7B	Hardcoded	82.0	67.9	58.5	29.3	**59.4**
	Generalized	71.3	67.4	54.8	21.7	53.9
Gemma-2-2B	Hardcoded	78.3	62.5	55.6	26.4	**55.7**
	Generalized	68.7	60.3	51.4	18.9	49.8
CodeGemma-2B	Hardcoded	76.0	52.8	40.3	21.4	**47.6**
	Generalized	68.7	49.0	39.2	16.9	43.5

Table 3: Results with different Solution Generalities.

Solution Generality. We explore the benefits of generalized versus hardcoded solutions. In a generalized solution, a reusable function is written to capture the logic underlying the math problem, allowing the code to handle similar mathematical problems by accepting different inputs. Such reusable functions help models acquire generalizable knowledge of mathematical problems, thereby improving their ability to generalize. In contrast, hardcoded solutions are tailored to specific problem instances, making them straightforward and easier for the model to learn.

As shown in Table 3, hardcoded solutions consistently outperform generalized solutions, suggesting that models benefit more from the explicit specificity of hardcoded logic, which simplifies learning. However, these results do not imply that hardcoded solutions are always better for mathematical instruction tuning. For example, a retrieval-augmented generation (RAG) model (Gao et al., 2023b) may perform better with generalized solutions if it retrieves relevant functions effectively. We do not explore this area, as our experiments focus on a zero-shot setting for coding styles.

Summary. Based on our analysis of coding styles, we draw the following conclusions: 1) Comments on code-based rationales consistently improve performance as they help LLMs understand the logic behind mathematical problems. 2) Descriptive naming conventions should be adopted to enhance the model's understanding of mathematical problems. 3) The hardcoded style of code-based rationales provides a simpler and more effective approach for model learning.

3.2 General Coding Instruction is Limited for Math Reasoning

The second question is whether coding instructions from general domains, i.e. non-mathematical domains, can improve mathematical ability under PoT reasoning. To investigate this, we train models on three instruction tuning datasets: General Code, Math-Code, and a combination of General Code and Math-Code.

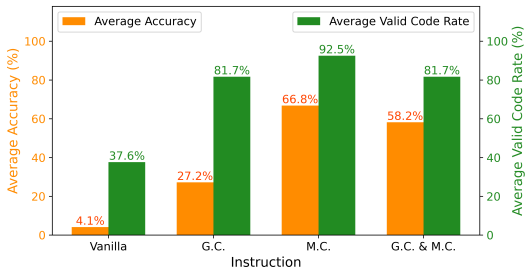
IT Data	Arith	SVAMP	GSM	MATH	Avg.
Zero-Shot					
Vanilla Llama	13.0	1.4	1.0	1.0	4.1
+ G.C.	35.0	36.6	25.3	11.8	27.2
+ M.C.	80.3	80.9	73.8	32.1	66.8
+ Mix	65.3	75.9	71.1	20.3	58.2
Vanilla CodeLlama	19.3	7.2	1.2	2.4	7.5
+ G.C.	70.3	50.9	23.4	14.4	39.8
+ M.C.	83.7	69.0	58.7	29.2	60.2
+ Mix	81.7	70.6	58.1	23.9	58.6
Few-Shot					
Vanilla Llama	78.0	73.1	50.0	18.0	54.8
+ G.C.	74.7	76.2	51.3	23.0	56.3
+ M.C.	83.7	80.4	71.6	33.8	67.4
+ Mix	78.7	78.0	71.6	34.4	65.7
Vanilla CodeLlama	79.7	52.9	21.1	14.1	42.0
+ G.C.	75.3	54.0	27.3	16.6	43.3
+ M.C.	80.0	65.8	54.4	25.9	56.5
+ Mix	80.7	65.4	51.8	24.7	55.6

Table 4: Performance of coding instruction from different domains. G.C., and M.C. represent General Code, and Math-Code, respectively. Vanilla Llama and Vanilla CodeLlama represent Llama-3.1-8B and CodeLlama-Python-7B, respectively. Cells are green if the instruction tuning boosts the vanilla models' performance, and red if the instruction tuning hurts the performance.

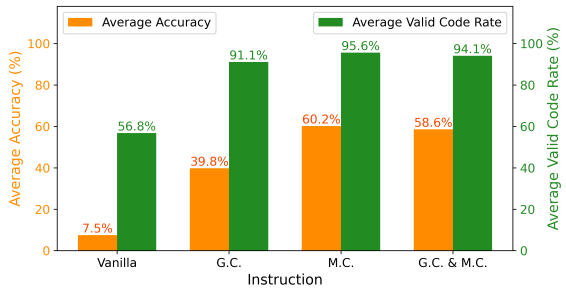
Prior studies (Wang et al., 2023b; Zhang et al., 2024b) indicate that, under CoT reasoning, instruction tuning on general code, i.e., non-math-specific data can have trivial or even negative effects on mathematical reasoning abilities. Under PoT reasoning, our experiments reveal a different trend, which is presented in Table 4.

In both zero-shot and few-shot settings, training solely on General Code mostly enhances the models' performance in solving math problems. This improvement is primarily attributed to the models' enhanced ability to generate valid code and follow instructions, making the models better suited for PoT reasoning compared to the base model. We define the valid code rate as the percentage of executable code generated by a model. Figure 2 presents the accuracy and valid code rate under zero-shot settings. The results demonstrate that training on General Code significantly improves the models' ability to generate valid code and adhere to instructions, leading to higher accuracy in solving mathematical questions.

However, models solely trained on Math-Code achieve the best performance, and incorporating General Code alongside Math-Code reduces its effectiveness. This implies that non-task-relevant coding instructions, i.e., General Code, distract the models and hinder their mathematical reasoning capabilities. These findings suggest that coding instructions from general domains offer limited



(a) LLama-3.1-8B



(b) CodeLlama-Python-7B

Figure 2: Average accuracy and average valid code rate across the evaluation datasets under zero-shot.

benefits for enhancing mathematical reasoning.

3.3 Math-Text is Not Always a Supplement for Math-Code

Recent works (Yue et al., 2023; Wang et al., 2023a) commonly employ hybrid training on both textual and code-based rationales to enhance mathematical reasoning. According to Yue et al. (2023), textual rationales contribute to general language-based reasoning, particularly for scenarios where PoT reasoning struggles, such as abstract reasoning in multiple-choice questions.

In this study, we delve deeper by focusing on math questions that require concrete calculations, excluding abstract reasoning, and investigate whether Math-Text consistently serves as a supplement to Math-Code. To ensure a fair comparison, we trained models on three instruction tuning datasets: Math-Text, Math-Code, which contain identical mathematical questions, differing only in the rationales provided, and a combination of both.

The results presented in Table 5 show different trends between general-purpose and code-specialized models. For the general-purpose model, Llama-3.1-8B, incorporating Math-Text alongside Math-Code slightly improves performance by enhancing the model's ability to comprehend mathematical problems in a language-based manner. However, for code-specialized models, this addition negatively impacts performance, as these models are inherently optimized for code generation. The inclusion of textual rationales may interfere with their specialized capabilities. This finding suggests that when mathematical questions include code-based rationales, augmenting them with textual explanations is beneficial only for models that are proficient in textual reasoning.

Model	IT Data	Arith	SVAMP	GSM	MATH	Avg
Zero-Shot						
Llama-3.1-8B	M.T.	38.7	62.2	54.6	20.6	44.0
	M.C.	80.3	80.9	73.8	32.1	66.8
	Mix	80.3	81.1	76.0	34.4	**67.9**
CodeLlama-7B	M.T.	22.7	44.8	32.4	7.9	26.9
	M.C.	83.7	69.0	58.7	29.2	**60.2**
	Mix	79.3	67.8	59.1	28.1	58.6
Few-Shot						
Llama-3.1-8B	M.T.	48.3	67.5	62.7	21.8	50.1
	M.C.	83.7	80.4	71.6	33.8	67.4
	Mix	83.0	79.8	73.0	35.4	**67.8**
CodeLlama-7B	M.T.	31.0	48.4	26.7	7.9	28.5
	M.C.	80.0	65.8	54.4	25.9	**56.5**
	Mix	78.3	65.1	53.4	24.5	55.3

Table 5: Performance of coding instruction from textual and code-based rationales. M.T, and M.C. represent Math-Code, and Math-Code, respectively. M.T. uses CoT prompting inference while others use PoT prompting inference.

4 CoinMath

Building on the insights from our systematic study on coding instruction, we introduce CoinMath, a strategy that enhances coding instructions by incorporating beneficial coding styles to improve the mathematical reasoning capabilities of LLMs.

4.1 Method

The CoinMath framework is depicted in Figure 3. As discussed in the previous section, LLMs demonstrate varying performance depending on the coding styles of mathematical rationales. Among the evaluated coding styles, Concise Comment performs best in the Comment Usage category, Descriptive Naming outperforms in Naming Conventions, and Hardcoded Solution outperforms in Solution Generalities. However, while these styles outperform their counterparts within their respective attributes, no single coding style proves universally optimal for all mathematical questions. For instance, as shown in Section 3.1, Hardcoded Solution, while better than Generalized Solution in the Solution Generality attribute, performs poorly on the SVAMP dataset for Llama-3.1-8B. This suggests that a fixed coding style may not be ideal for all mathematical problems.

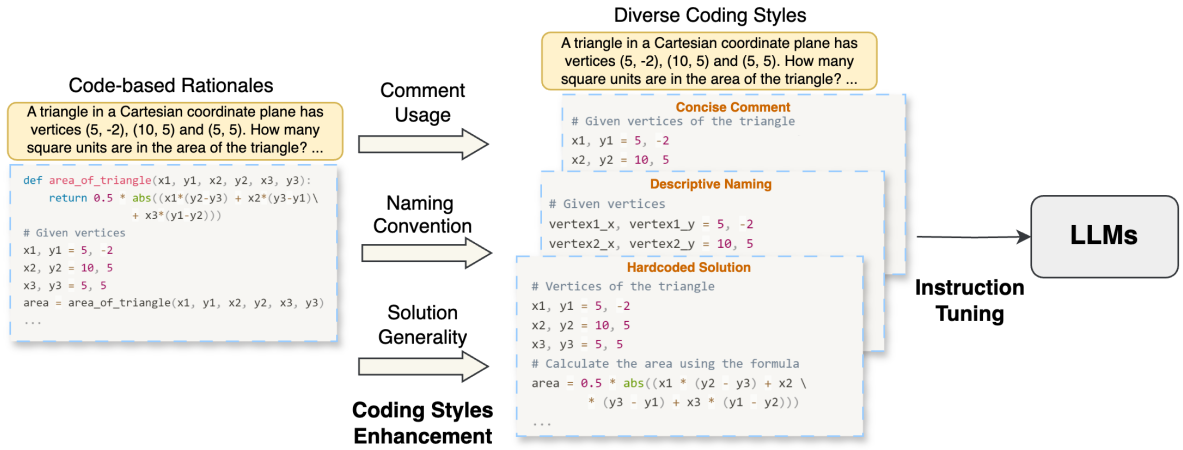


Figure 3: Overview of CoinMath framework. CoinMath generates three distinct variations of code-based rationales with advantageous coding attributes—Concise Comment, Descriptive Naming, and Hardcoded Solution—and ensembles them for LLM instruction tuning.

Instead of creating only one code-based rationale for a math question with fixed coding attributes, CoinMath generates three variations of rationales with diverse beneficial coding styles, using the same strategy outlined in Section 3.1. Each variation emphasizes one specific coding style attribute: concise comments, descriptive naming conventions, or hardcoded solutions. CoinMath then ensembles these three variations of rationales with beneficial coding attributes to maximize the enhancement of models' mathematical reasoning.

Additionally, CoinMath excludes coding instructions from general domains and mathematical textual rationales, as their contributions to improving mathematical reasoning are minimal or even negative.

4.2 Experimental Setup

We scale up the base instruction tuning dataset using the complete set of code-based rationales from MathInstruct (Yue et al., 2023), which contains 73k math questions with code-based rationales. Subsequently, we leverage this scaled dataset to generate code-based rationales with diverse coding styles using GPT-4o. We use the same evaluation datasets as in Section 3.

For comparison, we select representative models from the following three categories: **Base models**: We consider LLama-3.1 (Dubey et al., 2024) and CodeLlama-Python-7B (Roziere et al., 2023) as our base models. **Instruct models**: We include CodeLlama-3.1-8B-Instruct (Dubey et al., 2024) as a representative instruct-tuned model. **Math-specific models**: These models are fine-tuned specifically for solving math problems, including WizardMath (Luo et al., 2023), MathCoder (Wang et al., 2023a), and MAmmoTH (Yue et al., 2023). Since our instruction-tuning dataset with diverse coding styles is derived from MathInstruct which MAmmoTH utilizes, we further fine-tuned LLama-3.1-8B and CodeLlama-Python-7B on MathInstruct to ensure consistency in base models used for comparison. This approach establishes a strong baseline for assessing the improvements introduced by CoinMath. Additionally, we include Qwen2-Math-7B and Qwen2-Math-7B-Instruct (Yang et al., 2024), which are heavily optimized for mathematics and demonstrate superior performance, rivaling even closed-source models (e.g., GPT-4).

We evaluate the models' zero-shot performance and select the best result from two prompting approaches: CoT prompting and hybrid prompting. The hybrid prompting approach first uses PoT prompting and resorts to CoT prompting only if PoT fails to generate a result.

4.3 Results and Analysis

The experimental results are presented in Table 6. CoinMath significantly outperforms MAmmoTH after learning from code-based rationales that incorporate concise comments, descriptive naming conventions, and hardcoded solutions. Additionally, both CoinMath and Llama-3.1-Instruct are built on the Llama-3.1 base model and are instruct-tuned for mathematical reasoning, with Llama-3.1-Instruct further fine-tuned on other instruction-tuning topics. CoinMath surpasses Llama-3.1-Instruct except

Model	Base	Prompt	Arithmetic	SVAMP	GSM	MATH	Avg
Non-Math-Specific Model (<8B)							
Llama-3.1	-	CoT	36.7	45.7	17.9	8.6	27.3
Llama-3.1-Instruct	-	CoT	51.7	81.1	75.1	45.7	63.4
CodeLlama-Python	-	Hybrid	24.3	21.9	3.7	4.9	13.7
Math-specific Model (<8B)							
Qwen2-Math	Qwen2	CoT	76.3	91.4	78.8	71.6	79.5
Qwen2-Math-Instruct	Qwen2	CoT	65.0	87.0	79.2	70.2	75.4
WizardMath-V1.1	Llama-2	CoT	38.3	72.5	67.4	33.5	52.9
MathCoder-L†	Llama-2	-	-	71.5	64.2	23.3	-
MathCoder-CL†	CodeLlama	-	-	70.7	67.8	30.2	-
MAmmoTH	Llama-2	Hybrid	13.0	65.3	51.9	31.5	40.4
MAmmoTH-Coder	CodeLlama	Hybrid	31.0	54.2	31.8	20.5	34.4
MAmmoTH	Llama-3.1	Hybrid	71.4	81.3	73.0	39.1	66.2
MAmmoTH-Coder	CodeLlama-Python	Hybrid	72.7	67.0	52.9	21.8	53.6
CoinMath	Llama-3.1	Hybrid	77.0 (+5.6)	83.1 (+1.8)	76.4 (+3.4)	40.3 (+1.2)	69.2 (+3.0)
CoinMath	CodeLlama-Python	Hybrid	80.7 (+8.0)	75.1 (+8.1)	62.0 (+9.1)	31.8 (+10.0)	62.4 (+8.8)

Table 6: Zero-shot performance on mathematical evaluation datasets. Red numbers highlight the improvement compared with the same base models trained on MathInstruct, i.e., MAmmoTH models. † means the results are from the corresponding papers.

for the MATH dataset, demonstrating the effectiveness of our approach.

It is worth noting that the performance on the Arithmetic dataset, an out-of-domain evaluation dataset, decreases compared to the results in Section 3 as the training set is scaled up. In addition, it is reasonable that our models fall behind the Qwen2-Math models, as those models undergo extensive training for mathematics across pre-training, instruction tuning, and reinforcement learning stages while CoinMath focuses solely on the instruction tuning stage.

Ablation Study. We investigate the impact of combining different coding styles on model performance. Figure 4 presents the average accuracy across our evaluation datasets for various combinations of coding styles. Detailed performance metrics for each individual evaluation dataset are provided in Appendix E. Our results show that incorporating concise comments, descriptive naming conventions, and hardcoded solutions consistently enhances the model's performance in solving math problems. In contrast, their counterparts—no comment, obscure naming conventions, and generalized solutions—yield relatively inferior performance, achieving results similar to training solely with concise comments. Finally, combining all coding styles does not achieve performance as high as the combination of concise comments, descriptive naming conventions, and hardcoded solutions, further proving that these three attributes are particularly effective for code-based rationales in improving the model's mathematical reasoning ability.

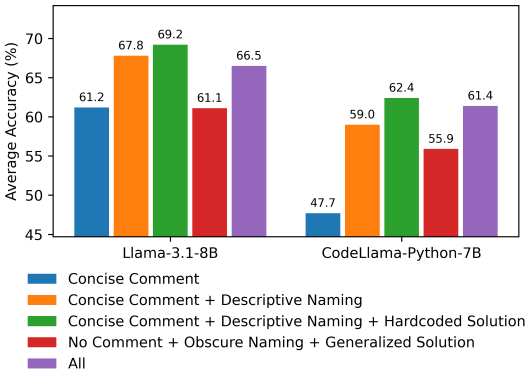


Figure 4: Average accuracy of the models using code-based rationales with different combinations of styles. The average accuracy is calculated across the Arithmetic, SVAMP, GSM, and MATH datasets.

5 Conclusion

We conduct a systematic study to investigate how coding instructions can effectively enhance the mathematical reasoning abilities of LLMs. Based on these findings, we propose CoinMath, which focuses specifically on mathematical code-based rationales and combines mathematical coding instructions with concise comments, descriptive naming, and hardcoded solutions to boost the mathematical reasoning capabilities of LLMs. Experimental results demonstrate that CoinMath outperforms the baseline model, MAmmoTH, one of the SOTA models, by an average improvement of 5.9% in

accuracy.

Limitations

We focus on evaluating LLMs' mathematical reasoning abilities on datasets requiring concrete calculations, including Arithmetic, SVAMP, GSM, and MATH. However, for other types of math questions involving abstract reasoning or mathematical knowledge, such as abstract algebra, and theorem comprehension. code-based solutions may be less effective. This underscores the importance of extending our work to these tasks to gain insights across a broader range of mathematical question types.

Acknowledgments

This research is supported by the Ministry of Education, Singapore, under its Science of Learning Grant (Award ID MOE-MOESOL2021-0006). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

The authors acknowledge the use of AI Assistants for grammar and language refinement during the preparation of the manuscript.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. In *The 18th Conference of the European Chapter of the Association for Computational Linguistics*, page 225.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen McAleer, Albert Q Jiang, Jia Deng, Stella Biderman, and Sean Welleck. 2023. Llemma: An open language model for mathematics. *arXiv preprint arXiv:2310.10631*.

Zhen Bi, Ningyu Zhang, Yinuo Jiang, Shumin Deng, Guozhou Zheng, and Huajun Chen. 2024. When do program-of-thought works for reasoning? In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17691–17699.

Sahil Chaudhary. 2023. Code alpaca: An instruction-following llama model for code generation. https://github.com/sahil280114/codealpaca.

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*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

CodeGemma-Team. 2024. Codegemma: Open code models based on gemma. *arXiv preprint arXiv:2406.11409*.

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

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

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023b. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Gemma-Team. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, yelong shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2024. ToRA: A tool-integrated reasoning agent for mathematical problem solving. In *The Twelfth International Conference on Learning Representations*.

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. *NeurIPS*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Hamish Ivison, Yizhong Wang, Valentina Pyatkin, Nathan Lambert, Matthew Peters, Pradeep Dasigi, Joel Jang, David Wadden, Noah A Smith, Iz Beltagy, et al. 2023. Camels in a changing climate: Enhancing lm adaptation with tulu 2. *arXiv preprint arXiv:2311.10702*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, 35:3843–3857.

Hongwei Liu, Zilong Zheng, Yuxuan Qiao, Haodong Duan, Zhiwei Fei, Fengzhe Zhou, Wenwei Zhang, Songyang Zhang, Dahua Lin, and Kai Chen. 2024. Mathbench: Evaluating the theory and application proficiency of llms with a hierarchical mathematics benchmark.

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. *arXiv preprint arXiv:2308.09583*.

YINGWEI Ma, Yue Liu, Yue Yu, Yuanliang Zhang, Yu Jiang, Changjian Wang, and Shanshan Li. 2024. At which training stage does code data help LLMs reasoning? In *The Twelfth International Conference on Learning Representations*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

OpenAI. 2024. Hello gpt-4o. Accessed: 2024-05-26.

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.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. 2024. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *arXiv preprint arXiv:2402.10176*.

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. *arXiv preprint arXiv:2310.03731*.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. 2023b. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36:74764–74786.

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.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, et al. 2024. Qwen2. 5-math technical report: Toward mathematical expert model via self-improvement. *arXiv preprint arXiv:2409.12122*.

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. *arXiv preprint arXiv:2309.05653*.

Bo-Wen Zhang, Yan Yan, Lin Li, and Guang Liu. 2024a. Infinitymath: A scalable instruction tuning dataset in programmatic mathematical reasoning. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 5405–5409.

Xinlu Zhang, Zhiyu Zoey Chen, Xi Ye, Xianjun Yang, Lichang Chen, William Yang Wang, and Linda Ruth Petzold. 2024b. Unveiling the impact of coding data instruction fine-tuning on large language models reasoning. *arXiv preprint arXiv:2405.20535*.

A Statistics of Instruction Tuning Datasets

Table 7 presents the statistics of the three base instruction-tuning datasets used in our study on coding instruction. Math-Code and Math-Text share the same set of mathematical questions but differ in their rationales, with Math-Code providing code-based rationales and Math-Text offering textual ones.

IT Dataset	# Sample	Characteristics	Annotation
Math-Code	26k	Math questions with code-based rationales	GPT4
Math-Text	26k	Math questions with texutal rationales	Human & Llama
General Code	21k	Coding instruction for general tasks	Llama

Table 7: Statistics of Math-Text, Math-Code and General Code

B Sample Questions from Evaluation Datasets

Figure 5 shows the samples from Arithmetic, SVAMP, GSM and MATH.

Question: What is (-1)/(-1)*((4563/(-15))/13 - -23)*-125?
Answer: 50

Question: Calculate 1000804 divided by -500402.
Answer: -2

(a) Arithmetic

Question: Paco had 26 salty cookies and 17 sweet cookies. He ate 14 sweet cookies and 9 salty cookies. How many salty cookies did Paco have left?
Answer: 17.0

(b) SVAMP

Question: Raymond and Samantha are cousins. Raymond was born 6 years before Samantha. Raymond had a son at the age of 23. If Samantha is now 31, how many years ago was Raymond's son born?
Solution: When Raymond's son was born Samantha was 23 - 6 = <<23-6=17>>17 years old.
Thus it has been 31 - 17 = <<31-17=14>>14 years since Raymond's son was born.
14

(c) GSM

Question: What is the distance between the two intersections of $y=x^2$ and $x+y=1$?
Solution: To find the x-coordinates of the intersections, substitute x^2 for y in $x+y=1$ and solve for x, resulting in ...
Answer: ["\\sqrt{10}", 3.1622776601683795]

(d) MATH

Figure 5: Sample questions from the evaluation datasets

C Prompts for Generating Diverse Coding Style Rationales

Figure 6 shows the instruction we used for generating code-based rationales with different coding styles.

D Instruction Tuning Details

We perform instruction tuning of the LLMs following the implementation of TULU (Wang et al., 2023b; Ivison et al., 2023). The models are fine-tuned using LoRA (Hu et al., 2021) with a rank of 64 and a total batch size of 128.

E Results of Mixing Coding Styles

Table 8 demonstrates the performance of different combinations of coding styles for each evaluation dataset.

```
instruction = """
1) Please add a comment at the beginning of the code, explaining the purpose of the code.
2) Add comments to each line of the code in the solution, explaining its functionality where comments are missing.
3) Do not modify the code itself; only add comments.
4) Output only the code and comments. Exclude any additional text from the response.
"""
```

(a) Instruction for generating detailed comments

```
instruction = """
1) Given a math question and its Program-of-Thought (PoT) solution, generate two versions of the solution code:
    a) **Descriptive Variable Names**: Use clear and descriptive names for all variables and functions.
    b) **Obscure Variable Names**: Use short and non-descriptive names (e.g., single letters or random abbreviations).

2) Keep the code's structure, comments, and logic unchanged; only replace the variable names.

3) Avoid using reserved or unsafe names such as {'sum', 'len', 'float', 'int', 'str', 'list', 'dict'} as variable names.

4) Output only the code and comments in the specified format. Exclude any additional text.
"""
```

(b) Instruction for applying different naming conventions

```
instruction = """
1) Given a math question and its Program-of-Thought (PoT) solution, generate two versions of the solution code:
    a) **Generalized Programming**:
        - Create a reusable function that abstracts the logic and can handle similar math problems with different inputs or parameters.
    b) **Hardcoded Programming**:
        - Write a specific solution tailored to the given math question, with fixed values and logic.

2) Output only the code and comments in the specified format. Exclude any additional text outside the provided template.
"""
```

(c) Instruction for generating generalized and hardcoded solutions

Figure 6: Instruction for generation diverse coding style rationales

Model	IT Data	Arithmetic	SVAMP	GSM	MATH	Average
Llama-3.1-8B	Concise Comment	58.0	75.5	72.8	38.6	61.2
	Concise Comment + Descriptive Naming	81.7	80.5	71.3	37.6	67.8
	Concise Comment + Descriptive Naming + Hardcoded Solution	77.0	83.1	76.4	40.3	69.2
	No Comment + Obscure Naming + General Solution	48.3	81.4	75.2	39.4	61.1
	All	72.3	83.0	75.0	35.8	66.5
CodeLlama-7B	Concise Comment	44.3	68.6	56.6	21.4	47.7
	Concise Comment + Descriptive Naming	80.3	69.6	56.6	29.5	59.0
	Concise Comment + Descriptive Naming + Hardcoded Solution	80.7	75.1	62.0	31.8	62.4
	No Comment + Obscure Naming + General Solution	66.0	69.9	60.2	27.4	55.9
	All	79.3	73.7	60.7	31.7	61.4

Table 8: Ablation Study of Coding Styles. Evaluation is under zero-shot setting