Towards Better Understanding of Program-of-Thought Reasoning in Cross-Lingual and Multilingual Environments

Patomporn Payoungkhamdee[1], Pume Tuchinda[1], Jinheon Baek[2]
Samuel Cahyawijaya[3], Can Udomcharoenchaikit[1], Potsawee Manakul[4],
Peerat Limkonchotiwat[5], Ekapol Chuangsuwanich[6], Sarana Nutanong[1]
[1]School of Information Science and Technology, VISTEC [2]KAIST [3]Cohere
[4]SCB 10X [5]AI Singapore [6]Department of Computer Engineering, Chulalongkorn University
{patomporn.p_s21, pumet_pro,canu_pro,snutanon}@vistec.ac.th
jinheon.baek@kaist.ac.kr, samuelcahyawijaya@cohere.com, potsawee@scb10x.com
peerat@aisingapore.org, ekapolc@cp.eng.chula.ac.th

Abstract

Multi-step reasoning is essential for large language models (LLMs), yet multilingual performance remains challenging. While Chain-of-Thought (CoT) prompting improves reasoning, it struggles with non-English languages due to the entanglement of reasoning and execution. Program-of-Thought (PoT) prompting separates reasoning from execution, offering a promising alternative but shifting the challenge to generating programs from non-English questions. We propose a framework to evaluate PoT by separating multilingual reasoning from code execution to examine (i) the impact of fine-tuning on question-reasoning alignment and (ii) how reasoning quality affects answer correctness. Our findings demonstrate that PoT fine-tuning substantially enhances multilingual reasoning, outperforming CoT fine-tuned models. We further demonstrate a strong correlation between reasoning quality (measured through code quality) and answer accuracy, highlighting its potential as a test-time performance improvement heuristic.[1]

1 Introduction

Multi-step reasoning is crucial for large language models (LLMs), enabling them to effectively solve complex tasks, including logical, mathematical, and symbolic problems (Wei et al., 2022; Yao et al., 2023). Extending this capability to multilingual settings can greatly expand accessibility, allowing diverse multilingual communities to benefit from these advances. However, Shi et al. (2023); Chen et al. (2024) showed that LLMs perform worse in non-English languages due to differences in linguistic structure and training data. This finding highlights the need for approaches that tackle both multi-step reasoning and multilingual challenges.

1.1 Research Gap

Traditionally, multi-step reasoning has been handled through chain-of-thought (CoT) (Wei et al., 2022), which allows LLMs to tackle mathematical problem-solving by breaking down problems into sequential reasoning steps. However, CoT requires models to handle both reasoning and computation, often leading to errors, particularly in multilingual contexts where linguistic disparity exacerbates the challenge. Program-of-thought (PoT) prompting (Chen et al., 2023; Gao et al., 2023) addresses these limitations by *decoupling reasoning from computation*. By shifting execution to an external interpreter, PoT ensures that the reasoning stage focuses solely on code generation, reducing reliance on the model's linguistic fluency in executing computational steps. This separation makes PoT advantageous in multilingual settings, where disparity among languages can greatly affect the model's performance.

Despite its potential, multilingualism in PoT remains underexplored. Compared to the rich literature on non-English CoT, especially regarding multilingual fine-tuning (Chen et al., 2024; Lai and Nissim, 2024; She et al., 2024; Zhu et al., 2024), PoT is limited to a single cross-lingual prompting study (Ranaldi et al., 2024a). Although recent work by Li et al. (2024a) proposed multilingual structural reasoning, the overall research landscape on PoT fine-tuning across languages remains sparse. This disparity emphasizes the necessity for research into PoT fine-tuning to fully exploit its potential for enhanced generalization to unseen languages and improved performance in multilingual environments.

1.2 Problem Formulation

This study examines the feasibility of decoupling natural language reasoning from computation in non-English languages. We formalize multilingual PoT within a two-stage framework as shown in

[1]https://github.com/calzonelover/xpot

Findings of the Association for Computational Linguistics: ACL 2025, pages 15810–15828

Figure 1: (i) $Q \rightarrow R$, where the model generates reasoning steps R from questions Q; (ii) $R \rightarrow A$, where an external interpreter executes R to obtain the final answer A. Our research is organized into two problems: P1 and P2, as follows.

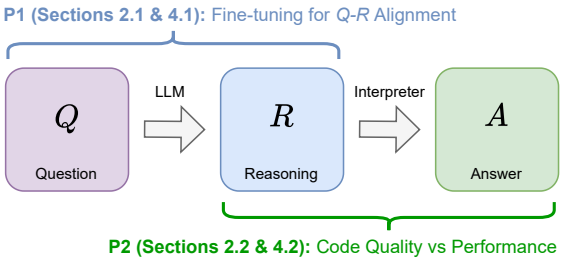

Figure 1: Proposed experimental framework under the PoT workflow $Q \rightarrow R \rightarrow A$. **P1**: Aligning multilingual questions (Q) with reasoning steps (R) through fine-tuning and inline comments. **P2**: Assessing the correlation between reasoning steps (R) and final answers (A) through code quality and test-time inference.

(P1) Fine-tuning for Q-R Alignment. This study attempts to answer the question: *How can we **align questions** Q posed in different languages with effective **reasoning steps** R in PoT, and how do **fine-tuning decisions** influence cross-lingual and multilingual reasoning performance?*

We evaluate different fine-tuning decisions under two fine-tuning scenarios.

- *Cross-lingual:* The model is fine-tuned only in English and evaluated cross-lingual zero-shot.
- *Multilingual:* Training data includes samples in target languages, allowing direct Q-R alignment in target languages.

These examinations analyze the impact of fine-tuning choices on reasoning alignment, providing insights into how language availability influences cross-lingual and multilingual PoT performance.

We explore multiple decisions regarding the use of inline comments in PoT reasoning, evaluating their impact in both cross-lingual and multilingual settings. For the cross-lingual setting, since the model is fine-tuned only in English, we compare keeping English comments versus removing them entirely. Our results show that removing comments leads to better generalization in unseen languages. For the multilingual setting, with access to target-language training data, we evaluate keeping English comments versus translating them into the target language. Our findings indicate that translating comments improves reasoning alignment, reflecting overall performance.

(P2) Code Quality vs Performance. This study attempts to answer the question: *To what extent does the **code quality** of reasoning steps R **affect the correctness** of final answers A, and how can we use this knowledge to* improve PoT performance*?*

We investigate the relationship between PoT performance and code quality measured through `ICE-Score` (Zhuo, 2024), which quantifies the correctness of intermediate steps within the code. Our analysis reveals a strong correlation between them. Building on this insight, we employ the ICE score as a heuristic for test-time scaling within the Soft Self-Consistency (Wang et al., 2024) method, implementing it as a form of soft voting. Experimental results show that this simple adjustment outperforms the standard Self-Consistency (Wang et al., 2023) baseline, where models generate multiple candidates and apply hard voting. In particular, this approach improves the overall accuracy across languages, in cross-lingual settings, increasing the performance from 31.6% to 56.6%.

1.3 Contributions

The contributions of our work are as follows:

- **Experimental Framework for Multilingual PoT —** We exploit the reasoning-execution disentanglement in PoT to break down the problem into two key challenges: Q-R alignment (how multilingual questions map to reasoning steps) and R-A association (how reasoning quality translates into correct answers).
- **Systematic Evaluation of Fine-Tuning for Q-R Alignment —** We investigate how fine-tuning impacts multilingual PoT performance under cross-lingual and multilingual settings, analyzing the role of inline comments.
- **Correlation Between Code Quality and Answer Accuracy —** We assess how the quality of generated reasoning steps R influences the correctness of the final answers A and leverage this insight to improve test-time inference.

2 Proposed Studies

2.1 Fine-tuning for Q-R Alignment (P1)

To fairly compare PoT and CoT, we use the *Grade School Math* (`GSM8K`) dataset (Cobbe et al., 2021) and explore three prompting strategies for generating PoT with an Oracle LLM: (i) zero-shot PoT, (ii) few-shot PoT, and (iii) the proposed few-shot PoT with CoT guidance, as shown in Figure 2. Zero-shot PoT generates Python solutions without examples. Few-shot PoT improves this with two

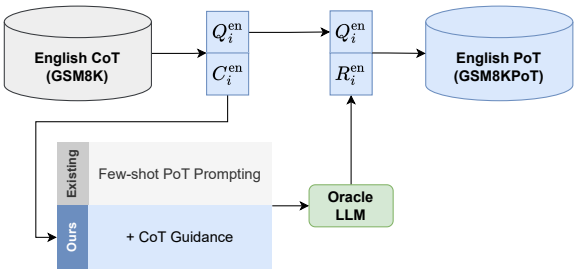

Figure 2: The generation pipeline for GSM8KPoT, in which a PoT answer ($\boldsymbol{R}_i^{en}$) is synthesized using the Oracle LLM, with additional natural language reasoning ($\boldsymbol{C}_i^{en}$) provided as guidance.

solved examples while adding CoT guidance further enhances program generation. As reported in Table 1, incorporating CoT guidance further enhances the correctness of PoT answer, yielding a correctness rate of 96.1% in PoT outputs, leading to the development of the GSM8KPoT dataset (details in Appendix B).

Generation Strategies for PoT Answer	Correctness (%)
Zero-shot PoT Prompting	58.7
Few-shot PoT Prompting	94.5
Few-shot PoT Prompting + CoT guidance	**96.1**

Table 1: Evaluation of the correctness of synthesized PoT answers under three distinct generation strategies.

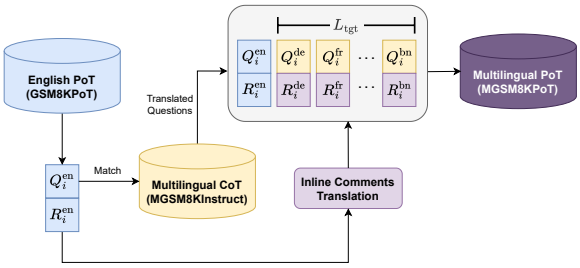

Figure 3: The generation pipeline for MGSM8KPoT involves the sequential construction of translated questions and code inline comments through mapping with an existing multilingual CoT dataset, followed by machine translation.

Examining PoT in multilingual settings is challenging due to the scarcity of datasets that align questions across multiple languages with structured reasoning steps. To address this challenge, we construct a standardized multilingual PoT dataset using the pipeline illustrated in Figure 3, yielding several dataset variants (detailed in Table 2) for evaluating both cross-lingual and multilingual fine-tuning. We control for language effects by varying the languages of questions and inline comments, allowing us to assess the impact of each fine-tuning strategy.

Setup	Lang. of Q	Lang. of Comm. in R	Dataset	Eq.
Cross	En	En	$\mathcal{D}_{\text{en}}^{\text{GSM8KPoT}}$	1
	En	*nc*	$\mathcal{D}_{\text{nc}}^{\text{GSM8KPoT}}$	2
Multi	En	Multi	$\mathcal{D}_{\text{cross-comment}}^{\text{MGSM8KPoT}}$	3
	Multi	En	$\mathcal{D}_{\text{cross-question}}^{\text{MGSM8KPoT}}$	4
	Multi	Multi	$\mathcal{D}_{\text{parallel}}^{\text{MGSM8KPoT}}$	5
	Multi	*nc*	$\mathcal{D}_{\text{nc}}^{\text{MGSM8KPoT}}$	6

Table 2: Our proposed study employs multiple approaches, leveraging the question-comment characteristics within the dataset to compare different best fine-tuning strategies. *NC* stands for "no comment".

In the context of cross-lingual and multilingual PoT, inline comments can potentially play a crucial role. As established by Shi et al. (2023), the language used in multi-step reasoning processes, such as those in CoT reasoning, is a key design consideration. We hypothesize that the design choices for inline comments in PoT function similarly to the language considerations in CoT. Thus, we analyze its implications comprehensively.

2.1.1 Cross-lingual Setup

We follow the zero-shot cross-lingual transfer approach described in Hu et al. (2020), where a large language model is trained on a source language, typically English, and evaluated on a set of unseen target languages. The cross-lingual setup includes two datasets: one that incorporates inline comments within the reasoning steps and another that excludes them, as defined below.

En-En — We employ GSM8KPoT as the foundational dataset, which can be formally represented by the following equation.

$$\mathcal{D}_{\text{en}}^{\text{GSM8KPoT}} = \{(\boldsymbol{Q}_i^{en}, \boldsymbol{R}_i^{en})\}_{i=1}^{N}, \quad (1)$$

where the questions $\boldsymbol{Q}_i^{en}$ are obtained from English GSM8K, and the synthesized intermediate reasoning in the programming language ($\boldsymbol{R}_i^{en}$) include inline comments in English. Note that the superscript en in $\boldsymbol{R}_i^{en}$ denotes the language of code comments.

En-nc — We also include a variant with all comments removed.

$$\mathcal{D}_{\text{nc}}^{\text{GSM8KPoT}} = \{(\boldsymbol{Q}_i^{en}, \boldsymbol{R}_i^{nc})\}_{i=1}^{N}, \quad (2)$$

where the superscript "nc" in the reasoning steps $\boldsymbol{R}_i^{\text{nc}}$ stands for "no comment".

2.1.2 Multilingual Setup

The multilingual setup comprises four datasets. Following the concept proposed in MGSM8KInstruct (Chen et al., 2024), we consider cross and parallel strategies, varying how the languages of questions and inline comments influence downstream performance, extending beyond the scope of prior work in multilingual structural reasoning (Li et al., 2024a). We also include a no-comment variant, pairing multilingual questions with reasoning steps that exclude comments. These four datasets are defined as follows.

En-Multi — Following the CoT definition in CoT Cross, we translate English inline comments using machine translation (MT), producing program reasoning in target languages:

$$\mathcal{D}^{\texttt{MGSM8KPoT}}_{\texttt{cross-comment}} = \{(\boldsymbol{Q}_i^{\text{en}}, \boldsymbol{R}_i^{l}) | l \in L_{\text{all}}\}_{i=1}^{N}, \quad (3)$$

where L_{all} denotes the language set. The superscript l in $\boldsymbol{R}_i^l$ is a variable representing a language.

Multi-En — This variant provides multilingual questions $\boldsymbol{Q}_i^l$ by applying machine translation to $\boldsymbol{Q}_i^{\text{en}}$, while keeping the inline comments in English.

$$\mathcal{D}^{\texttt{MGSM8KPoT}}_{\texttt{cross-question}} = \{(\boldsymbol{Q}_i^{l}, \boldsymbol{R}_i^{\text{en}}) | l \in L_{\text{all}}\}_{i=1}^{N}. \quad (4)$$

Multi-Multi — Both questions and inline comments are in the same language l:

$$\mathcal{D}^{\texttt{MGSM8KPoT}}_{\texttt{parallel}} = \{(\boldsymbol{Q}_i^{l}, \boldsymbol{R}_i^{l}) | l \in L_{\text{all}}\}_{i=1}^{N}. \quad (5)$$

Note that in this case, the superscript l in $\boldsymbol{Q}_i^l$ and $\boldsymbol{R}_i^l$ denotes the fact that both question and inline comments are in the same language.

Multi-nc — Similar to $\mathcal{D}^{\texttt{GSM8KPoT}}_{\text{nc}}$, we also include a no-comment variant for this setup.

$$\mathcal{D}^{\texttt{MGSM8KPoT}}_{\text{nc}} = \{(\boldsymbol{Q}_i^{l}, \boldsymbol{R}_i^{\text{nc}}) | l \in L_{\text{all}}\}_{i=1}^{N}. \quad (6)$$

2.2 Code Quality Analysis (P2)

After addressing the multilingual problem with PoT, the task is split into two parts: multi-step reasoning via code generation and execution via a Python interpreter for numerical computations. While the interpreter ensures arithmetic accuracy, the challenge lies in generating syntactically and logically correct programs free of errors.

We assess code quality using the ICE-Score (Zhuo, 2024), which measures usefulness (how well the code addresses the query) and functional correctness (evaluated through intermediate validation with an Oracle LLM). Our focus is functional correctness, rating program validity from 0 (incorrect/incomplete) to 4 (fully correct).

We use ICE-Score to assess whether improved alignment strategies enhance both accuracy and code quality. Furthermore, to compare code quality with final answer accuracy, we conduct two analyses: (i) *System level*: Spearman correlation assesses whether higher-quality code improves overall model performance. (ii) *Sample level*: AUC and t-test assess whether code validity can determine answer correctness.

Test-time Scaling. We investigate the use of ICE-Score to enhance model inference in test-time scaling. Building on Self-Consistency (SC) (Wang et al., 2023), which generates multiple reasoning candidates and applies majority voting (hard voting), we extend this approach with Soft Self-Consistency (Soft-SC) (Wang et al., 2024). Soft-SC refines this process by averaging the ICE-Score for each final answer candidate, ranking responses by overall code quality. This shift from hard to soft voting may improve performance.

2.3 Discussions

The six datasets enable us to examine how language alignment and inline comments impact cross-lingual and multilingual PoT reasoning. Inline comments act as alignment anchors between questions and reasoning steps expressed in a programming language. However, they can hinder cross-lingual generalization to unseen languages. In this respect, we aim to understand (i) how multilingual data availability influences PoT's ability to generate accurate reasoning steps and (ii) how inline comments affect performance across language setups.

Code quality analysis provides an intermediate observation linking these decisions to the accuracy of the final answer. By examining both aspects, we establish a structured understanding of how multilingual data and inference-time strategies interact to improve PoT performance, laying the groundwork for our experimental validation in Section 4.2.

3 Experimental Setup

Base LLMs. We conduct experiments with various base LLMs, using Llama2-7B (Touvron et al.,

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
CoT	43.6	32.4	30.4	30.4	26.4	25.2	15.2	4.8	2.0	5.6	21.6
PoT	**58.0**	**40.4**	**40.4**	**43.6**	**37.1**	**38.4**	**32.7**	**7.6**	**5.6**	**12.0**	**31.6**
CodeLlama 7B											
CoT	43.2	33.2	32.8	39.6	26.8	27.2	18.8	16.4	3.2	9.2	25.0
PoT	**58.8**	**48.4**	**51.6**	**53.6**	**49.8**	**41.6**	**39.6**	**26.8**	**4.4**	**11.2**	**38.6**
Llama2-13B											
CoT	47.4	39.2	37.6	41.2	38.0	35.2	18.8	7.2	**7.4**	6.8	27.9
PoT	**64.0**	**52.4**	**54.4**	**55.6**	**51.2**	**44.0**	**40.0**	**13.9**	7.2	**13.6**	**39.6**
Llama3-8B											
CoT	62.8	51.2	52.8	54.8	45.2	40.0	33.6	39.6	28.0	39.6	44.8
PoT	**68.4**	**62.2**	**59.2**	**62.4**	**60.4**	**52.4**	**45.4**	**43.6**	**34.8**	**46.0**	**53.5**
Qwen2.5-7B											
CoT	73.6	64.0	72.4	72.4	73.2	70.0	56.4	64.8	14.8	53.2	61.5
PoT	**86.5**	**79.3**	**80.1**	**80.4**	**80.9**	**76.4**	**66.9**	**70.0**	**22.0**	**61.1**	**70.4**

Table 3: Accuracy (%) on MGSM in **cross-lingual setting**.

2023) as the foundation for the following variants:

i) *Code-specific variant*: CodeLlama-7B (Roziere et al., 2023), optimized for code and programming-related tasks.

ii) *Size variant*: Llama2-13B (Touvron et al., 2023), a larger-scale version of Llama2.

iii) *Version variant*: Llama3-8B (Grattafiori et al., 2024), a more recent iteration with enhanced multilingual capabilities.

iv) *Family variant*: Qwen2.5-7B (Qwen et al., 2025), a recent model in another family with strong multilingual performance.

Oracle LLM. To ensure reproducibility, we employ Llama3.1-405B Instruct (Grattafiori et al., 2024) as our Oracle model for generating the PoT dataset and assessing the quality of the code.

Evaluation. We evaluate model performance by measuring accuracy on the MGSM (Shi et al., 2023) dataset in a zero-shot setting using greedy decoding. The study includes the following languages: English (en), German (de), French (fr), Spanish (es), Russian (ru), Chinese (zh), Japanese (ja), Thai (th), Swahili (sw), and Bengali (bn). For CoT evaluation, numerical outputs are extracted via regular expressions and compared to labels, following Chen et al. (2024). For PoT evaluation, generated programs are executed in a Python interpreter, with outputs compared to labels for accuracy.

Measures: As outlined in Table 2, in cross-lingual setting, we finetune each LLM independently on GSM8K (Cobbe et al., 2021) and GSM8KPoT, using both $\mathcal{D}_{\text{en}}^{\text{GSM8KPoT}}$ and $\mathcal{D}_{\text{nc}}^{\text{GSM8KPoT}}$ variants. For multilingual CoT, we finetune each LLM separately on MGSM8K Instruct Parallel and Cross (Chen et al., 2024). For multilingual PoT, we utilize the generated answers from GSM8KPoT and map the questions for each language in MGSM8K Instruct to create MGSM8KPoT. To study the effects of inline comments, we create versions of GSM8K and MGSM8KPoT without inline comments by removing them from the original datasets. Additionally, we generate a variation of MGSM8KPoT by applying machine translation. We utilize nllb-200-distilled-600M (Team et al., 2022) for translating inline comments, ensuring coverage across all languages in this study.

4 Experimental Results

4.1 Impact of Q-R Alignment Fine-tuning

4.1.1 Cross-lingual Setting

The experimental results presented in Table 3 indicate that PoT consistently outperforms CoT across all languages and model classes, achieving superior results in 39 out of 40 cases. The only exception is Swahili in the Llama2-13B model, where PoT reached an accuracy of 7.2%, compared to CoT's 7.4%, showing only a slight difference.

When comparing models of the same size, CodeLlama-7B consistently outperforms Llama2-7B in most languages. The improvements are notable in non-English languages such as German (+8.0), French (+11.2), and Thai (+19.2), suggesting that the incorporation of code data during pretraining improves structured reasoning even in cross-lingual settings. Scaling up to Llama2-13B leads to further improvements over both Llama2-7B and CodeLlama-7B. While model size remains an important factor in boosting overall accuracy, the strong performance of CodeLlama-7B relative to Llama2-7B indicates that increased code data during pretraining (Roziere et al., 2023) can en-

hance reasoning ability. For models with enhanced multilingual capabilities, such as Llama3-8B and Qwen2.5-7B, where the performance disparity between languages is narrower, the findings indicate that PoT continues to demonstrate superior effectiveness in cross-lingual contexts, yielding higher accuracy across previously unseen languages.

In Table 4, we compare performance when finetuning between $\mathcal{D}_{\text{en}}^{\text{GSM8KPoT}}$ and $\mathcal{D}_{\text{nc}}^{\text{GSM8KPoT}}$. Overall, training without comments tends to improve non-English accuracy across Llama2 models for both 7B and 13B variants, where omitting comments reduces English accuracy slightly but yields larger gains in non-English languages, like German and Bengali, boosting the overall score.

Method	en	de	bn	ALL
Llama2-7B				
With Comments	**58.3**	37.9	9.9	30.0
Without Comments	58.0	**40.4**	**12.0**	**31.6**
CodeLlama-7B				
With Comments	**61.4**	45.2	**15.6**	36.6
Without Comments	58.8	**48.4**	11.2	**38.6**
Llama2-13B				
With Comments	**67.3**	48.4	13.2	37.4
Without Comments	64.0	**52.4**	**13.6**	**39.6**
Llama3-8B				
With Comments	46.4	48.2	37.5	40.6
Without Comments	**68.4**	**62.2**	**46.0**	**53.5**
Qwen2.5-7B				
With Comments	**86.5**	**79.3**	**61.1**	**70.4**
Without Comments	38.4	26.0	53.6	39.2

Table 4: The impact of code comments on accuracy across different models in cross-lingual setup. The ALL score is from Appendix H.1.

CodeLlama-7B shows mixed results: including comments helps in English and Bengali, whereas excluding comments improves German and also leads to a higher overall score. This may reflect the specialized training corpus for CodeLlama, which emphasizes code tokens and might interact differently with inline explanations. Finally, Llama3-8B shows the largest swing: removing comments substantially boosts performance for all languages (including English), suggesting that inline explanations can sometimes distract or misalign the Q–R. In contrast, Qwen2.5-7B exhibits the opposite trend—performance significantly improves with comments across languages. Taken together, these findings indicate that, for most models, $\mathcal{D}_{\text{nc}}^{\text{GSM8KPoT}}$ provides better cross-lingual generalization and more robust Q–R alignment.

4.1.2 Multilingual Setting

Table 5 shows that PoT continues to outperform CoT in multilingual settings across all languages and model variants. While a few exceptions are observed in certain languages within the LLMs trained on $\mathcal{D}^{\text{MGSM8KPoT}}$`cross-comment`, as well as in Bengali for Qwen2.5-7B trained on $\mathcal{D}^{\text{MGSM8KPoT}}$`parallel`, the overall findings consistently demonstrate that PoT outperforms CoT.

As observed in the cross-lingual experiments, CodeLlama2-7B maintains its advantage over Llama2-7B across all languages in the multilingual setting. This performance gap is particularly pronounced in French (+8.0), Chinese (+6.0), and English (+5.9), further suggesting that increasing code data during pretraining yields stronger reasoning capabilities. Scaling to larger models continues to deliver gains, with Llama2-13B showing consistent improvements over both 7B variants. However, the most dramatic improvements come from Llama3-8B, which achieves substantially higher accuracy across all languages, reaching 76.5% in English while maintaining strong performance even in non-English languages like Thai (57.6%) and Bengali (55.2%). Meanwhile, Qwen2.5-7B demonstrates improvements in 8 out of 10 languages, maintaining performance in Chinese, though experiencing a decline in Bengali accuracy. The stronger gains highlights the benefits of explicit multilingual training over multilingual transfer, emphasizing the role of scaling and adaptation in optimizing reasoning across languages.

Finally, we investigate the most effective way to align Q and R in a multilingual context. As shown in Table 6, translating inline comments into the target language consistently yields superior performance across all model variants. We hypothesize that this improvement comes from the enhanced semantic alignment between code and natural language when comments are presented in the target language during training. In summary, these findings indicate that $\mathcal{D}_{\texttt{parallel}}^{\text{MGSM8KPoT}}$ provides the optimal Q-R alignment for multilingual settings.

4.2 R-A Relationship: Code Quality Analysis

Does Better Strategy Improve Code Quality? As discussed in Section 2.2, we assess code quality across alignment strategies in cross-lingual and multilingual settings, focusing on Llama2-7B and CodeLlama-7B. Table 7 shows that higher accuracy correlates with better code quality. Additionally,

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
CoT Cross	45.2	38.4	36.8	40.0	33.2	**33.6**	23.6	16.8	**18.8**	17.2	30.4
PoT Cross Comment	**54.8**	**47.2**	**51.2**	**46.2**	**42.8**	33.2	**34.8**	**20.0**	17.6	**18.0**	**36.6**
CoT Parallel	48.8	42.4	44.0	42.4	38.0	42.4	31.6	33.6	34.4	27.6	38.5
PoT Parallel	**56.0**	**47.2**	**46.4**	**54.0**	**49.6**	**44.4**	**40.0**	**40.4**	**37.6**	**30.8**	**44.6**
CodeLlama2-7B											
CoT Cross	47.6	38.8	33.2	38.8	35.2	31.6	28.8	23.6	17.2	20.4	31.5
PoT Cross Comment	**58.0**	**47.2**	**51.4**	**52.4**	**48.0**	**44.2**	**38.0**	**28.8**	**20.4**	**22.4**	**41.1**
CoT Parallel	46.0	40.0	38.8	44.0	43.2	41.2	35.6	41.6	30.8	32.0	39.3
PoT Parallel	**61.9**	**52.8**	**54.4**	**52.4**	**53.6**	**50.4**	**44.8**	**44.8**	**39.6**	**35.6**	**49.0**
Llama2-13B											
CoT Cross	58.4	50.4	46.4	49.6	43.6	**43.2**	33.6	**25.6**	**23.6**	**24.4**	39.9
PoT Cross Comment	**62.0**	**53.6**	**52.4**	**54.8**	**50.0**	42.0	**39.2**	21.6	23.2	23.2	**42.2**
CoT Parallel	60.8	53.6	52.0	54.4	52.8	53.6	45.2	43.6	41.2	38.0	49.5
PoT Parallel	**63.5**	**56.4**	**59.2**	**59.2**	**55.2**	**54.0**	**51.6**	**50.0**	**52.8**	**44.4**	**54.6**
Llama3-8B											
CoT Cross	69.2	58.0	54.8	58.0	57.2	50.0	44.4	40.4	40.4	42.0	51.4
PoT Cross Comment	**72.8**	**62.4**	**66.4**	**67.2**	**63.6**	**52.0**	**49.6**	**52.0**	**46.2**	**51.2**	**58.3**
CoT Parallel	66.8	53.6	57.2	60.8	62.4	60.0	50.8	57.6	53.6	54.8	57.8
PoT Parallel	**76.5**	**64.4**	**63.2**	**66.4**	**64.0**	**63.2**	**56.4**	**57.6**	**59.6**	**55.2**	**62.6**
Qwen2.5-7B											
CoT Cross	74.8	63.6	64.0	71.6	68.8	68.8	62.4	60.8	**40.8**	50.4	62.6
PoT Cross Comment	**81.2**	**75.2**	**78.0**	**74.8**	**75.6**	**70.8**	**63.6**	**68.8**	38.8	**59.2**	**68.6**
CoT Parallel	74.4	67.2	62.4	66.4	64.4	**68.8**	61.2	68.8	56.8	**60.0**	65.0
PoT Parallel	**81.4**	**76.4**	**74.0**	**78.8**	**72.8**	**68.8**	**64.0**	**71.2**	**62.0**	57.6	**70.7**

Table 5: Accuracy (%) on MGSM in **multilingual setup**.

Method	en	de	bn	ALL
Llama2-7B				
PoT Cross Comment	54.8	**47.2**	18.0	36.6
PoT Cross Question	46.0	37.6	28.8	37.7
PoT Parallel	**56.0**	**47.2**	**30.8**	**44.6**
PoT No Comment	53.6	41.6	29.2	40.6
CodeLlama2-7B				
PoT Cross Comment	58.0	47.2	22.4	41.1
PoT Cross Question	48.0	42.8	28.8	40.5
PoT Parallel	**61.9**	**52.8**	**35.6**	**49.0**
PoT No Comment	56.8	47.6	35.2	45.6
Llama2-13B				
PoT Cross Comment	62.0	53.6	23.2	42.2
PoT Cross Question	53.0	47.6	35.9	45.1
PoT Parallel	**63.5**	**56.4**	**44.4**	**54.6**
PoT No Comment	58.4	51.6	35.2	46.4
Llama3-8B				
PoT Cross Comment	72.8	62.4	51.2	58.3
PoT Cross Question	37.2	30.3	30.4	31.6
PoT Parallel	**76.5**	**64.4**	**55.2**	**62.6**
PoT No Comment	65.2	60.0	48.4	56.5
Qwen2.5-7B				
PoT Cross Comment	81.2	75.2	**59.2**	68.6
PoT Cross	75.4	64.1	56.2	62.3
PoT Parallel	**81.4**	**76.4**	57.6	**70.7**
PoT No Comment	76.1	66.0	56.0	64.7

Table 6: The impact of various fine-tuning strategies is examined, where PoT Cross includes either comment-only or question-only translation. In contrast, the Parallel approach involves either the exclusion of comments or the inclusion of translated comments.

code quality in lower resource languages, like Bengali, is much lower than in English and German, which aligns with the accuracy trends. This finding reflects the inherent challenges of generating code in low-resource languages, where model performance is typically more constrained.

Method	en	de	bn	ALL
Cross-lingual				
Llama2-7B				
With Comments	**2.49**	**1.87**	0.45	1.39
Without Comments	**2.49**	**1.87**	**0.49**	**1.44**
CodeLlama2-7B				
With Comments	**2.66**	2.06	**0.61**	1.97
Without Comments	2.55	**2.13**	0.54	**2.02**
Multilingual				
Llama2-7B				
PoT Cross Comment	2.56	2.41	1.26	1.98
PoT Cross Question	2.32	2.07	1.52	2.03
PoT Parallel	**2.83**	**2.55**	**1.96**	**2.45**
PoT No Comment	2.54	2.16	1.71	2.13
CodeLlama2-7B				
PoT Cross Comment	2.84	2.40	1.34	2.15
PoT Cross Question	2.45	2.23	1.54	2.11
PoT Parallel	**2.88**	**2.68**	**2.04**	**2.56**
PoT No Comment	2.61	2.41	1.87	2.28

Table 7: Code quality assessment with ICE-Score

System Level Correlation. Figure 4 illustrates a strong system-level correlation between MGSM accuracy and code quality, as measured by `ICE-Score`. Across all finetuning strategies, in both cross-lingual and multilingual, we observe a consistent trend where higher code quality pos-

itively correlates with improved accuracy. This relationship is quantified by a Spearman rank correlation coefficient of 0.91 and 0.76 for cross-lingual and multilingual, respectively.

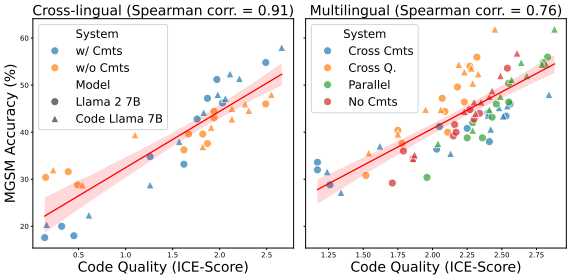

Figure 4: The relationship between code quality and answer accuracy in cross-lingual and multilingual settings. Each point represents a given language, considering a specific system and model combination.

Notably, this correlation persists across different model architectures and code generation conditions, reinforcing the importance of alignment strategies in enhancing both code quality and accuracy. These insights highlight the broader impact of alignment and resource availability on code generation, supporting the necessity of assessing the quality of intermediate outputs.

Sample Level Association. Beyond system-level trends, we examined whether code quality can determine the correctness of individual solutions. This relationship is demonstrated in Table 8, where the percentage distributions of `ICE-Score` for correct and incorrect answers show substantial differences across score ranges. To further quantify this discriminative ability, we calculated the AUC for `ICE-Score` as a predictor of correctness, obtaining strong values of 0.94 and 0.96 for cross-lingual and multilingual settings, respectively. Additionally, a t-test reveals a statistically significant difference between the correct and incorrect groups. A detailed language-wise analysis is provided in Appendix H.2.

Setting	Answer Type	0	1	2	3	4
Cross	Correct	3.4	1.5	3.9	3.8	87.3
	Incorrect	75.2	14.2	8.4	0.7	1.5
Multi	Correct	2.0	1.3	3.6	4.1	89.0
	Incorrect	52.8	25.8	17.5	2.3	1.6

Table 8: ICE-Score distribution (%) for correct and incorrect answers in cross- and multilingual settings.

Application in Test-Time Scaling. We now explore the potential of applying the ICE-Score as a heuristic for test-time scaling. We evaluate three approaches as discussed in Section 2.2: (i) baseline model predictions without scaling, (ii) Self-Consistency (`SC`), and (iii) Soft Self-Consistency (`Soft-SC`) guided by the `ICE-Score`. As shown in Table 9, our results indicate that test-time scaling substantially improves reasoning accuracy across both cross-lingual and multilingual settings. Conventional `SC` provides moderate gains, but `Soft-SC` with `ICE-Score` further boosts performance by prioritizing high-quality reasoning steps. Notably, for Llama2-7B, `Soft-SC` improves **cross-lingual performance from 39.2 to 56.6** and **multilingual performance from 57.2 to 71.2**. Similarly, CodeLlama-7B shows strong gains in both setups, demonstrating the method's robustness across model architectures. These findings underscore the benefit of intermediate quality assessment as a means to improve cross-lingual and multilingual PoT reasoning and overall performance.

Method	en	de	bn	ALL
Cross-lingual				
Llama2-7B				
Without Comments	58.0	40.4	12.0	31.6
+ SC	65.2	51.6	15.2	39.2
+ Soft-SC (ICE-Score)	**76.8**	**69.2**	**33.6**	**56.6**
CodeLlama-7B				
Without Comments	58.8	48.4	11.2	38.6
+ SC	69.6	57.2	17.2	46.7
+ Soft-SC (ICE-Score)	**75.7**	**71.2**	**33.6**	**61.1**
Multilingual				
Llama2-7B				
PoT Parallel	56.0	47.2	30.8	44.6
+ SC	64.8	58.0	47.6	57.2
+ Soft-SC (ICE-Score)	**77.6**	**72.0**	**65.6**	**71.2**
CodeLlama-7B				
PoT Parallel	61.9	52.8	35.6	49.0
+ SC	68.8	66.4	53.6	62.8
+ Soft-SC (ICE-Score)	**79.2**	**77.6**	**68.8**	**75.6**

Table 9: A comparative analysis of performance when implementing conventional `SC` and the proposed `Soft-SC` with `ICE-Score` in an optimal framework for cross-lingual and multilingual configurations.

5 Related Work

Mathematical Reasoning. Recent advancements in LLMs' mathematical reasoning capabilities have been driven by Chain-of-Thought (CoT) prompting (Wei et al., 2022; Nye et al., 2021), which significantly outperforms direct-answer approaches by generating intermediate step-by-step reasoning. Building on CoT, various enhancements have emerged, including self-consistency, which replaces greedy decoding with sampling-based inference to select the most consistent solution (Wang

et al., 2023). Meanwhile, PoT and PaL (Chen et al., 2023; Gao et al., 2023) improve reasoning by delegating computation to a Python interpreter, reducing the task of translating problems into code.

Another key advancement is instruction fine-tuning on mathematical datasets. Yu et al. (2024) introduced MetaMathQA, expanding existing datasets through diverse rephrasings, while Yue et al. (2024) leveraged a hybrid MathInstruct dataset combining CoT's generality with PoT's computational precision. Additionally, external tool integration has been explored (Liao et al., 2024; Gou et al., 2024), with curated tool-use datasets enhancing LLMs' reasoning capabilities.

Multilingual Mathematical Reasoning. Despite LLMs' advancements in English mathematical reasoning, their performance in other languages still lags. Efforts to bridge this gap include sample translation for multilingual alignment (Chen et al., 2024; Lai and Nissim, 2024; Zhu et al., 2024) and multilingual preference optimization (She et al., 2024). Chen et al. (2024) created a multilingual mathematical dataset by translating GSM8K into ten languages, though accurate translations remain a costly and time-consuming endeavor. To mitigate this, Zhu et al. (2024) proposed a two-step approach: translating questions into English before fine-tuning on larger English datasets like MetaMathQA. Alternatively, She et al. (2024) leveraged existing translation models as alignment signals for preference optimization.

Beyond dataset translation, prompting techniques offer a cost-effective alternative. Huang et al. (2023) introduced role-playing prompts where the model first translates questions into English before applying CoT reasoning. Ranaldi et al. (2024b) proposed a Tree-of-Thought framework for structured, multi-step reasoning across languages. Ranaldi et al. (2024a) extended PoT with Cross-PAL, aligning reasoning across multiple languages through code generation.

6 Conclusion

This study explores the effectiveness of Program-of-Thought (PoT) prompting for reasoning in cross- and multilingual settings by leveraging the reasoning-execution disentanglement concept. We decompose the problem into two key challenges: (i) aligning multilingual questions with structured reasoning steps and (ii) assessing the impact of reasoning quality on final answer correctness. Through systematic experimentation across cross-lingual and multilingual settings, we show that PoT fine-tuning substantially enhances reasoning alignment and generalization, outperforming CoT fine-tuning.

Moreover, we establish a strong correlation between reasoning quality and answer accuracy. By leveraging `ICE-Score`-based inference strategies, we enhance performance, particularly in low-resource languages. These findings provide insights into optimizing PoT for multilingual reasoning and open avenues for future research on improving reasoning alignment and execution.

Our findings contribute to a deeper understanding of multilingual PoT reasoning, providing insights into fine-tuning strategies and inference-time optimizations. Future work can extend this framework to additional reasoning-intensive tasks and explore more advanced alignment techniques to enhance PoT's multilingual capabilities.

Limitations

GSM8K as a Reasoning Benchmark. The experimental setup of this study is grounded in grade school math problems from GSM8K; therefore, the results and key findings *may not* generalize to other reasoning-intensive tasks. Furthermore, recent studies have raised concerns regarding potential data contamination (Li et al., 2024b; Zhang et al., 2024; Mirzadeh et al., 2025). Nonetheless, GSM8K remains the gold standard for assessing multi-step reasoning in the literature. We use this benchmark to ensure cross-comparability with existing work while emphasizing that our experimental framework is adaptable to any multi-step reasoning benchmark. In future work, we plan to extend our assessments to additional benchmarks to further validate our findings.

ICE-Score Model Choice and Test-time Scaling. Our test-time scaling study presents a preliminary investigation into leveraging the ICE-Score as a Soft Self-Consistency (Soft-SC) heuristic. Prior work on ICE-Score (Zhuo, 2024) suggests that stronger models yield better results. This work prioritizes the evaluation accuracy of the code quality itself, verifying the correlation between the intermediate and end results. To this end, we employ the 405B variant of Llama3 for ICE-Score calculations in the correlation studies at the system and sample levels. To maintain consistency, we continue to use this model for test-time scaling experiments.

Our findings indicate that incorporating ICE-Score into Soft-SC leads to performance improvements. However, the magnitude of these gains may depend on the specific ICE-Score model used. Future work should examine other ICE-Score configurations or alternative solutions, assessing their cost-benefit trade-off.

Acknowledgments

This research is supported by the National Research Foundation, Singapore, under its National Large Language Models Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore. In addition, the computing resources used in this project were supported by the WangchanX program, a research and development initiative by PyThaiNLP and VISTEC-DEPA Thailand AI Research Institute.

References

Nuo Chen, Zinan Zheng, Ning Wu, Ming Gong, Dongmei Zhang, and Jia Li. 2024. Breaking language barriers in multilingual mathematical reasoning: Insights and observations. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 7001–7016, Miami, Florida, USA. Association for Computational Linguistics.

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

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: program-aided language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

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*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, and 542 others. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Junjie Hu, Sebastian Ruder, Aditya Siddhant, Graham Neubig, Orhan Firat, and Melvin Johnson. 2020. Xtreme: a massively multilingual multi-task benchmark for evaluating cross-lingual generalization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Haoyang Huang, Tianyi Tang, Dongdong Zhang, Xin Zhao, Ting Song, Yan Xia, and Furu Wei. 2023. Not all languages are created equal in LLMs: Improving multilingual capability by cross-lingual-thought prompting. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 12365–12394, Singapore. Association for Computational Linguistics.

Huiyuan Lai and Malvina Nissim. 2024. mCoT: Multilingual instruction tuning for reasoning consistency in language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12012–12026, Bangkok, Thailand. Association for Computational Linguistics.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V. Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, Yuling Gu, Saumya Malik, Victoria Graf, Jena D. Hwang, Jiangjiang Yang, Ronan Le Bras, Oyvind Tafjord, Chris Wilhelm, Luca Soldaini, and 4 others. 2025. Tulu 3: Pushing frontiers in open language model post-training. *Preprint*, arXiv:2411.15124.

Bryan Li, Tamer Alkhouli, Daniele Bonadiman, Nikolaos Pappas, and Saab Mansour. 2024a. Eliciting better multilingual structured reasoning from LLMs through code. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5154–5169, Bangkok, Thailand. Association for Computational Linguistics.

Qintong Li, Leyang Cui, Xueliang Zhao, Lingpeng Kong, and Wei Bi. 2024b. GSM-plus: A comprehensive benchmark for evaluating the robustness of LLMs as mathematical problem solvers. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2961–2984, Bangkok, Thailand. Association for Computational Linguistics.

Minpeng Liao, Chengxi Li, Wei Luo, Wu Jing, and Kai Fan. 2024. MARIO: MAth reasoning with code interpreter output - a reproducible pipeline. In *Findings of the Association for Computational Linguistics: ACL*

2024, pages 905–924, Bangkok, Thailand. Association for Computational Linguistics.

Seyed Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. 2025. GSM-symbolic: Understanding the limitations of mathematical reasoning in large language models. In *The Thirteenth International Conference on Learning Representations*.

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

Libo Qin, Qiguang Chen, Fuxuan Wei, Shijue Huang, and Wanxiang Che. 2023. Cross-lingual prompting: Improving zero-shot chain-of-thought reasoning across languages. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2695–2709, Singapore. Association for Computational Linguistics.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Leonardo Ranaldi, Giulia Pucci, Barry Haddow, and Alexandra Birch. 2024a. Empowering multi-step reasoning across languages via program-aided language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12171–12187, Miami, Florida, USA. Association for Computational Linguistics.

Leonardo Ranaldi, Giulia Pucci, Federico Ranaldi, Elena Sofia Ruzzetti, and Fabio Massimo Zanzotto. 2024b. A tree-of-thoughts to broaden multi-step reasoning across languages. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 1229–1241, Mexico City, Mexico. Association for Computational Linguistics.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Shuaijie She, Wei Zou, Shujian Huang, Wenhao Zhu, Xiang Liu, Xiang Geng, and Jiajun Chen. 2024. MAPO: Advancing multilingual reasoning through multilingual-alignment-as-preference optimization. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10015–10027, Bangkok, Thailand. Association for Computational Linguistics.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, and 20 others. 2022. No language left behind: Scaling human-centered machine translation. *Preprint*, arXiv:2207.04672.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, and 1 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Han Wang, Archiki Prasad, Elias Stengel-Eskin, and Mohit Bansal. 2024. Soft self-consistency improves language models agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 287–301, Bangkok, Thailand. Association for Computational Linguistics.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed H. Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik R Narasimhan. 2023. Tree of thoughts: Deliberate problem solving with large language models. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*.

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*.

Hugh Zhang, Jeff Da, Dean Lee, Vaughn Robinson, Catherine Wu, William Song, Tiffany Zhao, Pranav Vishnu Raja, Charlotte Zhuang, Dylan Z

Slack, Qin Lyu, Sean M. Hendryx, Russell Kaplan, Michele Lunati, and Summer Yue. 2024. A careful examination of large language model performance on grade school arithmetic. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. 2023. CodeBERTScore: Evaluating code generation with pretrained models of code. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13921–13937, Singapore. Association for Computational Linguistics.

Wenhao Zhu, Shujian Huang, Fei Yuan, Shuaijie She, Jiajun Chen, and Alexandra Birch. 2024. Question translation training for better multilingual reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 8411–8423, Bangkok, Thailand. Association for Computational Linguistics.

Terry Yue Zhuo. 2024. ICE-score: Instructing large language models to evaluate code. In *Findings of the Association for Computational Linguistics: EACL 2024*, pages 2232–2242, St. Julian's, Malta. Association for Computational Linguistics.

A MGSM8KInstruct

We adopt MGSM8KInstruct (Chen et al., 2024) as the reference dataset for CoT in multilingual settings. This dataset comprises question-reasoning pairs ($\boldsymbol{R}_i$, $\boldsymbol{Q}_i$) with $\boldsymbol{Q}_i$ expressed in English, along with translations in nine additional languages, enabling the alignment of reasoning capabilities across different languages. Chen et al. (2024) introduced two training strategies: (i) *CoT Cross*: Incorporates English questions with answers in the target language, promoting multilingual adaptability. Formally, the dataset is represented as:

$$\mathcal{D}_{\texttt{cross}}^{\texttt{MGSM8KInstruct}} = \{(\boldsymbol{Q}_i^{\text{en}}, \boldsymbol{C}_i^l) | l \in L_{\text{all}}\}_{i=1}^N$$

where L_{all} includes both English and target languages. (ii) *CoT Parallel*: Uses question-answer pairs in the same language to enhancing the PoT capability within each target language, denoted as:

$$\mathcal{D}_{\texttt{parallel}}^{\texttt{MGSM8KInstruct}} = \{(\boldsymbol{Q}_i^l, \boldsymbol{C}_i^l) | l \in L_{\text{all}}\}_{i=1}^N.$$

B PoT Generation Methods

To facilitate a fair comparison between PoT and CoT, we employ the GSM8K dataset, a collection of grade-school math problems that require 2-8 reasoning steps to solve, as the foundational benchmark. As illustrated in Figure 2, we generate solutions in a programming language using an Oracle LLM through various methodologies:

1. *Zero-shot PoT Prompting*: Following the zero-shot prompting framework from Chen et al. (2023), the model is instructed to generate the `solver()` function in Python using a prompt $\boldsymbol{S}_{\text{PoT}}$ with no exemplars. Formally, the PoT synthesis from an Oracle LLM is represented as $\hat{\boldsymbol{R}}_i \sim p_{\text{Oracle}}(\boldsymbol{Q}_i | \boldsymbol{S}_{\text{PoT}})$.
2. *Few-shot PoT Prompting*: Building on the methodologies of Chen et al. (2023); Gao et al. (2023), k in-context exemplars, $\boldsymbol{E}_{\text{FS}} = \{(\boldsymbol{Q}_1, \boldsymbol{R}_1), ..., (\boldsymbol{Q}_k, \boldsymbol{R}_k)\}$, are incorporated into the prompt to provide explicit guidance on desired outputs. The PoT synthesis is thus defined as $\hat{\boldsymbol{R}}_i \sim p_{\text{Oracle}}(\boldsymbol{Q}_i | \boldsymbol{E}_{\text{FS}}, \boldsymbol{S}_{\text{PoT}})$.
3. *Few-shot PoT Prompting + CoT Guidance*: Based on initial observations that high-quality PoT outputs often align with structured CoT reasoning ($\boldsymbol{C}$), an additional CoT guidance mechanism is introduced to better direct program generation. In this setting, the examples $\boldsymbol{E}_{\text{FS-CoT}} = \{(\boldsymbol{Q}_1, \boldsymbol{C}_1, \boldsymbol{R}_1), ..., (\boldsymbol{Q}_k, \boldsymbol{C}_k, \boldsymbol{R}_k)\}$ include both CoT reasoning ($\boldsymbol{C}_i$) and the corresponding PoT solution ($\boldsymbol{R}_i$). The PoT synthesis is then formulated as $\hat{\boldsymbol{R}}_i \sim p_{\text{Oracle}}(\boldsymbol{Q}_i | \boldsymbol{C}_i, \boldsymbol{E}_{\text{FS-CoT}}, \boldsymbol{S}_{\text{PoT}})$.

We empirically tested three approaches to identify the most effective method for maximizing the match between program execution outputs and gold-standard answers, using Llama3.1 405B Instruct (Grattafiori et al., 2024) as the Oracle LLM: zero-shot prompting, few-shot prompting, and few-shot prompting with CoT reasoning. In zero-shot prompting, the model is given only the original GSM8K question and generates the corresponding Python code to solve it. Few-shot prompting extends this by providing the model with two exemplars of correctly solved GSM8K questions along with their corresponding Python solutions. Few-shot prompting with CoT reasoning further builds upon this by incorporating both the original answer and its Chain-of-Thought (CoT) reasoning from GSM8K. Our evaluation demonstrated that the few-shot + CoT approach consistently outperformed the other methods, achieving a correctness rate of 96.1% in synthesizing PoT samples. In comparison, the few-shot prompting method yielded a correctness rate of 94.5%, while the zero-shot approach resulted in a significantly lower accuracy of 58.7%.

System
You are a helpful assistant. Answer the following question by implementing a solver() function in Python program step by step, and then return the answer.
Solve them in a step-by-step fashion and output a single option as the final answer in [language] language.

User
Question: [Question]

Figure 5: Zero-shot PoT prompt template for PoT synthesis, where [Question] serves as a placeholder for the problem statement.

C Training Setting

Our code is primarily based on the MathOctopus codebase, with some minor modifications. The code will be made available.

Prompt Template. During training and testing, we consistently use the same prompt template from MathOctopus (Chen et al., 2024).

Setting. We fully fintune all our models on a single 4xA100 node for three epochs with a maximum sequence length 1024. For the Llama2 family and CodeLlama, we used a learning rate of 2e-5 and an effective batch size of 512. However, we found that this setting caused the Llama3 8B model not to produce desirable results, which we discuss further in the next section. Thus, we changed the effective batch size to 128 and the learning rate to 5e-6, following (Lambert et al., 2025) for Llama 3 8B. To generate multiple candidate predictions, we set $top_k = 50$ and a temperature of 0.7, selecting the top 40 sequences for the voting process.

D Computing Resources

We trained LLaMA family models on 4× NVIDIA A100 (80GB) GPUs, completing the fine-tuning process within approximately one hour for cross-lingual settings and around eight hours for multilingual settings.

During inference, generating predictions in a greedy fashion requires only three minutes. However, when producing multiple answer candidates with K=40, the process takes approximately seven hours to complete.

For Oracle LLM inference, we utilize a separate dedicated setup with 4× NVIDIA A100 (80GB) GPUs to host the LLM service, which is responsible for constructing PoT answers and evaluating code quality. The quality assessment process requires approximately 45 minutes for a single prediction and extends to 32 hours when assessing 40 candidates across all languages for a given model configuration. Additionally, we employ 62 concurrent processes to maximize inference throughput.

In summary, our experiments required a total of 544 A100 GPU hours for fine-tuning, 52 hours for inference, and 146 hours for quality assessment.

Method	ALL
Cross-lingual	
Llama2-7B	
Without Comments	39.2
+ `Soft-SC (ICE-Score)`	**56.6**
CodeLlama-7B	
Without Comments	38.6
+ `SC`	46.7
+ `Soft-SC (ICE-Score)`	**61.1**
Multilingual	
Llama2-7B	
PoT Parallel	44.6
+ `SC`	57.2
+ `Soft-SC (ICE-Score)`	**71.2**
CodeLlama-7B	
PoT Parallel	49.0
+ `SC`	62.8
+ `Soft-SC (ICE-Score)`	**75.6**
Non-Fine-Tuned PoT	
Llama2-7B	
CLP (Qin et al., 2023)	48.3
SCLP (Qin et al., 2023)	54.1
Cross-PAL (Ranaldi et al., 2024a)	49.9
SCross-PAL (Ranaldi et al., 2024a)	**56.3**

Table 10: The comparison of our adopted test-time scaling approaches with SOTA non-fine-tuned PoT approaches. The results of non-fune-tuned PoT are taken from Ranaldi et al. (2024a).

E Comparison with Non-Fine-Tuned PoT

We compare our test-time scaling experiments with state-of-the-art (SOTA) non-fine-tuned PoT prompting methods and observe that our product models from PoT parallel with `SC` outperform SCross-PAL from Ranaldi et al. (2024a) by 0.9 percentage points. Furthermore, our proposed `Soft-SC` with `ICE-Score` achieves a significant accuracy improvement, increasing from 57.2% to 71.2%.

F Sensitivity of Llama3-8B

During our testing, we observed that Llama3-8B exhibited significant sensitivity to our hyperparameters and chat template configurations. Notably, the model frequently failed to generate the `def solver():` function header at the beginning of its reasoning chain, which is critical for extracting and compiling the generated code correctly. To mitigate this issue, we inserted a prefix in our prompt, as illustrated in Figure 8. Additionally, with our initial

System
You are a helpful assistant. Answer the following question by implementing a solver() function in Python program step by step, and then return the answer.
Solve them in a step-by-step fashion and output a single option as the final answer in [language] language.

User
Question: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

Assistant
```
def solver():
    # Roger started with 5 tennis balls.
    tennis_balls = 5
    # 2 cans of 3 tennis balls each is
    bought_balls = 2 * 3
    # tennis balls. The answer is
    answer = tennis_balls + bought_balls"
    return answer
```

User
Question: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Assistant
```
def solver():
    # The bakers started with 200 loaves
    loaves_baked = 200
    # They sold 93 in the morning and 39 in the afternoon
    loaves_sold_morning = 93
    loaves_sold_afternoon = 39
    # The grocery store returned 6 loaves.
    loaves_returned = 6
    # The answer is
    answer = loaves_baked - loaves_sold_morning - loaves_sold_afternoon + loaves_returned
    return answer
```

User
Question: [Question]

Figure 6: Few-shot PoT prompt template for PoT synthesis, with exemplars adapted from (Gao et al., 2023; Chen et al., 2023).

hyperparameters, Llama3-8B frequently generated code snippets that failed to compile. Specifically, 9.12% of its outputs were non-compilable, a significantly higher rate compared to Llama2-7B (3.08%), CodeLlama-7B (2.04%), and Llama2-13B (1.84%). However, after refining our hyperparameters based on the approach outlined by (Lambert et al., 2025) and adjusting the chat template, we observed a substantial reduction in compilation errors, with the failure rate dropping to 1.68%.

User
Below are instructions for a task.
Write a response that appropriately completes the request in [language]. Please answer in Python with inline comments in [language].
Instruction:
[Question]
Response:
def solver():

Figure 8: Updated prompt with an added prefix (`def solver():`) for Llama3-8B.

G Alternative Metric For Code Quality Assessment

Alternatively, to `ICE-Score`, we evaluated code quality using `CodeBERT-Score` (Zhou et al., 2023).

However, we noticed that GSM8K (Cobbe et al., 2021) primarily consists of short code snippets where errors often involved small numerical mistakes rather than large structural or semantic differences. Many of the errors stemmed from minor computation mistakes, like using the wrong arithmetic expression or associating wrong counts with the subject. Since `CodeBERT-Score` is designed to assess broader semantic similarity, it struggled to distinguish the minute differences between correct and incorrect code. As shown in Table 16, the scores across different systems varied only slightly ($\pm$ 1.0%), failing to reflect the accuracy differences observed in Tables 11, 12. This suggests that `CodeBERT-Score` may not be well-suited for evaluating correctness in GSM8K-style problems.

H Full Tables

H.1 Main Results

This subsection serves as an extension of the results presented in Section 4.1. In particular, we present the complete results for all fine-tuning alignment strategies in Table 11 for cross-lingual settings and Table 12 for multilingual settings. These tables provide a detailed breakdown of performance across different configurations, reinforcing the trends observed in Section 4.1. The results confirm that PoT fine-tuning significantly improves multilingual reasoning, with cross-lingual generalization benefiting from the removal of inline comments and multilingual settings achieving higher alignment when comments are translated into target languages.

H.2 Code Analysis

This subsection extends the analysis presented in Section 4.2 by providing a full set of code quality evaluation results. Our code analysis scores are presented in Table 13 and Table 14 for cross-lingual and multilingual `ICE-Score`, respectively. Similarly, Table 15 and Table 16 provide the corresponding results for `CodeBERT-Score`. These results are consistent with the findings in Section 4.2, confirming a strong correlation between reasoning quality and final answer correctness. The observed trends support the effectiveness of leveraging code quality for test-time scaling, with improvements in underrepresented languages being particularly notable.

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
With Comments	58.3	37.9	40.0	44.4	39.2	33.2	25.1	6.8	5.2	9.9	30.0
Without Comments	58.0	40.4	40.4	43.6	37.1	38.4	32.7	7.6	5.6	12.0	31.6
CodeLlama2-7B											
With Comments	61.4	45.2	47.6	47.0	41.6	37.9	35.6	29.2	5.2	15.6	36.6
Without Comments	58.8	48.4	51.6	53.6	49.8	41.6	39.6	26.8	4.4	11.2	38.6
Llama2-13B											
With Comments	67.3	48.4	49.4	54.0	44.4	44.4	35.2	11.6	6.4	13.2	37.4
Without Comments	64.0	52.4	54.4	55.6	51.2	44.0	40.0	13.9	7.2	13.6	39.6
Llama3-8B											
With Comments	46.4	48.2	38.3	49.2	41.8	49.6	36.9	37.9	20.7	37.5	40.6
Without Comments	68.4	62.2	59.2	62.4	60.4	52.4	45.4	43.6	34.8	46.0	53.5
Qwen2.5-7B											
With Comments	86.5	79.3	80.1	80.4	80.9	76.4	66.9	70.0	22.0	61.1	70.4
Without Comments	38.4	26.0	13.2	0.4	76.4	32.8	8.8	23.9	18.0	53.6	29.2

Table 11: Accuracy (%) on MGSM for all cross-lingual PoT variants, providing the full results corresponding to the subset shown in Table 4

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
PoT Cross Comment	54.8	47.2	51.2	46.2	42.8	33.2	34.8	20.0	17.6	18.0	36.6
PoT Cross Question	46.0	37.6	43.0	44.4	39.6	39.6	36.3	31.6	30.4	28.8	37.7
PoT Parallel	56.0	47.2	46.4	54.0	49.6	44.4	40.0	40.4	37.6	30.8	44.6
PoT No Comment	53.6	41.6	42.8	44.8	44.0	39.2	40.0	36.0	34.4	29.2	40.6
CodeLlama2-7B											
PoT Cross Comment	58.0	47.2	51.4	52.4	48.0	44.2	38.0	28.8	20.4	22.4	41.1
PoT Cross Question	48.0	42.8	46.0	44.6	45.0	41.0	36.9	39.4	32.0	28.8	40.5
PoT Parallel	61.9	52.8	54.4	52.4	53.6	50.4	44.8	44.8	39.6	35.6	49.0
PoT No Comment	56.8	47.6	46.4	48.8	52.0	46.4	44.0	44.4	34.4	35.2	45.6
Llama2-13B											
PoT Cross Comment	62.0	53.6	52.4	54.8	50.0	42.0	39.2	21.6	23.2	23.2	42.2
PoT Cross Question	53.0	47.6	49.4	51.2	48.8	48.8	42.4	38.0	35.9	35.9	45.1
PoT Parallel	63.5	56.4	59.2	59.2	55.2	54.0	51.6	50.0	52.8	44.4	54.6
PoT No Comment	58.4	51.6	52.4	48.8	50.4	45.6	39.2	43.6	39.2	35.2	46.4
Llama3-8B											
PoT Cross Comment	72.8	62.4	66.4	67.2	63.6	52.0	49.6	52.0	46.2	51.2	58.3
PoT Cross Question	37.2	30.3	34.3	37.6	33.1	27.4	23.6	35.1	27.1	30.4	31.6
PoT Parallel	76.5	64.4	63.2	66.4	64.0	63.2	56.4	57.6	59.6	55.2	62.6
PoT No Comment	65.2	60.0	59.6	59.2	57.2	57.2	49.4	55.2	53.6	48.4	56.5
Qwen2.5-7B											
PoT Cross Comment	81.2	75.2	78.0	74.8	75.6	70.8	63.6	68.8	38.8	59.2	68.6
PoT Cross Question	75.4	64.1	66.5	67.5	67.5	59.1	59.7	59.4	48.0	56.2	62.3
PoT Parallel	81.4	76.4	74.0	78.8	72.8	68.8	64.0	71.2	62.0	57.6	70.7
PoT No Comment	76.1	66.0	66.4	71.2	67.2	63.2	62.0	63.6	55.6	56.0	64.7

Table 12: Accuracy (%) on MGSM for all multilingual PoT variants, providing the full results corresponding to the subset shown in Table 6

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
With Comments	2.40	1.87	1.84	2.03	1.76	1.41	1.26	0.32	0.14	0.45	1.35
Without Comments	2.49	1.87	1.94	1.94	1.82	1.67	1.62	0.39	0.15	0.49	1.44
CodeLlama2-7B											
With Comments	2.66	2.06	2.21	2.11	1.98	1.83	1.57	1.26	0.16	0.61	1.65
Without Comments	2.54	2.13	2.15	2.31	2.13	1.85	1.82	1.10	0.23	0.54	1.68
Llama2-13B											
With Comments	2.79	2.21	2.29	2.37	2.02	2.04	1.76	0.56	0.21	0.60	1.69
Without Comments	2.49	1.87	1.94	1.94	1.82	1.67	1.62	0.39	0.15	0.49	1.44
Llama3-8B											
With Comments	2.74	2.32	2.40	2.63	2.36	2.07	1.86	2.20	1.34	1.81	2.17
Without Comments	2.75	2.46	2.61	2.56	2.54	2.00	1.92	1.88	1.64	2.00	2.24

Table 13: `ICE-Score` on MGSM for all cross-lingual PoT variants, providing the full results corresponding to the subset shown in Table 7

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
PoT Cross Comment	2.56	2.41	2.53	2.40	2.25	2.05	2.03	1.17	1.17	1.26	1.98
PoT Cross Question	2.32	2.07	2.23	2.27	2.18	2.09	2.11	1.75	1.78	1.52	2.03
PoT Parallel	2.83	2.55	2.46	2.80	2.55	2.43	2.25	2.29	2.36	1.96	2.45
PoT No Comment	2.54	2.16	2.30	2.27	2.34	2.15	2.17	1.79	1.86	1.71	2.13
CodeLlama2-7B											
PoT Cross Comment	2.84	2.40	2.54	2.48	2.51	2.42	2.13	1.62	1.24	1.34	2.15
PoT Cross Question	2.45	2.23	2.19	2.33	2.28	2.29	2.06	1.95	1.75	1.54	2.11
PoT Parallel	2.88	2.68	2.64	2.82	2.79	2.57	2.49	2.41	2.29	2.04	2.56
PoT No Comment	2.61	2.41	2.35	2.43	2.49	2.30	2.31	2.23	1.86	1.87	2.28
Llama2-13B											
PoT Cross Comment	2.91	2.76	2.67	2.88	2.58	2.50	2.25	1.40	1.65	1.55	2.31
PoT Cross Question	2.69	2.48	2.48	2.58	2.52	2.48	2.30	2.12	2.12	1.84	2.36
PoT Parallel	2.94	2.91	2.90	2.82	2.93	2.81	2.79	2.70	2.75	2.36	2.79
PoT No Comment	2.70	2.52	2.62	2.55	2.45	2.45	2.30	2.30	2.26	2.17	2.43
Llama3-8B											
PoT Cross Comment	3.12	2.75	2.87	2.90	2.84	2.44	2.42	2.32	2.13	2.38	2.61
PoT Cross Question	2.51	2.07	2.31	2.25	2.13	2.04	1.95	2.11	1.67	1.83	2.09
PoT Parallel	3.15	2.88	2.82	3.03	2.84	2.82	2.58	2.72	2.68	2.56	2.81
PoT No Comment	2.76	2.54	2.47	2.53	2.48	2.56	2.24	2.46	2.33	2.10	2.45

Table 14: `ICE-Score` on MGSM for all multilingual PoT variants, providing the full results corresponding to the subset shown in Table 7

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
With Comments	90.50	83.53	84.18	83.21	85.68	85.94	86.02	81.33	79.24	81.81	84.14
Without Comments	90.08	81.56	82.03	81.16	83.94	85.66	86.61	82.15	78.56	81.92	83.37
CodeLlama2-7B											
With Comments	90.69	85.23	84.00	82.80	86.33	86.15	86.40	84.14	78.87	83.04	84.77
Without Comments	89.85	84.67	83.88	83.11	86.54	85.68	86.74	84.07	79.61	83.21	84.74
Llama2-13B											
With Comments	90.50	84.66	84.29	83.75	85.12	87.10	87.05	82.00	80.71	82.82	84.80
Without Comments	90.29	85.62	85.08	84.78	86.72	87.06	87.64	83.52	81.53	83.60	85.58
Llama3-8B											
With Comments	83.23	80.30	79.90	81.22	80.36	80.06	79.58	80.65	78.61	79.64	80.36
Without Comments	84.36	82.11	81.60	80.38	82.23	79.26	78.16	79.95	78.34	79.50	80.59

Table 15: `CodeBERT-Score` (F1) on MGSM for all cross-lingual PoT variants.

Method	en	de	fr	es	ru	zh	ja	th	sw	bn	All
Llama2-7B											
PoT Cross Comment	90.26	88.07	87.94	87.56	87.38	86.81	87.13	85.03	82.63	84.33	86.71
PoT Cross Question	89.13	88.41	88.12	88.16	88.39	87.22	87.60	86.11	86.24	86.18	87.56
PoT Parallel	89.95	89.14	88.90	89.27	88.72	87.99	88.34	87.35	87.53	86.89	88.41
PoT No Comment	89.73	88.74	88.29	88.60	88.13	87.24	88.14	86.64	86.79	86.62	87.89
CodeLlama2-7B											
PoT Cross Comment	90.95	88.38	88.45	88.74	87.89	87.75	88.23	84.79	83.55	85.23	87.40
PoT Cross Question	89.10	88.26	87.84	87.79	87.25	87.06	87.03	86.45	86.02	85.95	87.28
PoT Parallel	90.08	88.79	89.03	88.77	88.47	87.69	87.68	87.37	86.55	87.09	88.15
PoT No Comment	89.84	88.44	88.71	88.66	88.04	87.47	87.49	86.90	86.55	86.14	87.82
Llama2-13B											
PoT Cross Comment	90.26	87.80	87.75	87.36	87.27	86.83	87.53	83.72	82.91	84.05	86.55
PoT Cross Question	89.92	88.84	88.63	88.65	88.48	87.97	88.11	86.92	87.21	87.13	88.19
PoT Parallel	90.40	89.53	89.32	89.51	89.55	88.53	88.92	88.10	87.93	87.57	88.94
PoT No Comment	89.99	89.02	89.14	89.00	88.74	88.40	88.44	87.30	87.10	87.04	88.42
Llama3-8B											
PoT Cross Comment	91.02	89.32	88.69	89.06	89.01	88.29	88.32	87.91	86.59	87.55	88.58
PoT Cross Question	80.30	78.94	78.95	79.61	79.63	79.22	79.51	79.18	79.12	79.63	79.38
PoT Parallel	90.49	90.17	89.81	89.31	89.38	88.71	88.99	88.90	88.40	88.75	89.29
PoT No Comment	89.80	88.81	88.78	89.07	88.40	88.35	88.70	88.37	88.08	88.10	88.65

Table 16: `CodeBERT-Score` (F1) on MGSM in for all multilingual PoT variants.

Method	en	de	fr	es	ru	zh	ja	th	sw	bn
Cross-lingual										
Llama2-7B										
AUC	0.9659	0.9544	0.9782	0.9728	0.9733	0.9622	0.9680	0.9683	0.7376	0.9068
T-Statsitic	28.66	25.67	32.72	31.29	31.82	28.81	29.04	12.87	3.23	10.56
CodeLlama-7B										
AUC	0.9752	0.9708	0.9661	0.9736	0.9723	0.9606	0.9528	0.9215	0.8524	0.9096
T-Statsitic	28.85	29.85	29.82	31.68	31.36	28.86	25.30	16.48	4.96	9.96
Multilingual										
Llama2-7B										
AUC	0.9847	0.9561	0.9725	0.9771	0.9675	0.9714	0.9865	0.9556	0.9437	0.9480
T-Statsitic	31.21	25.70	28.94	28.51	29.04	27.80	36.85	25.14	22.99	21.79
CodeLlama-7B										
AUC	0.9627	0.9327	0.9627	0.9756	0.9536	0.9511	0.9476	0.9532	0.9537	0.9476
T-Statsitic	25.13	21.46	25.94	28.20	23.93	23.45	22.96	25.07	24.10	23.96

Table 17: T-Statistic and AUC scores for Llama2-7B and CodeLlama-7B across MGSM

System
You are a helpful assistant. Answer the following question by implementing a solver() function in Python program step by step, and then return the answer.
Solve them in a step-by-step fashion and output a single option as the final answer in [language] language.

User
Question: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
Chain-of-thought: Roger started with 5 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Assistant
```
def solver():
    # Roger started with 5 tennis balls.
    tennis_balls = 5
    # 2 cans of 3 tennis balls each is
    bought_balls = 2 * 3
    # tennis balls. The answer is
    answer = tennis_balls + bought_balls"
    return answer
```

User
Question: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?
Chain-of-thought: The bakers started with 200 loaves of bread. They sold 93 loaves in the morning and 39 loaves in the afternoon: 93 + 39 = 132 loaves sold. A grocery store returned 6 loaves, so they got 6 loaves back. 200 - 132 + 6 = 74 loaves left. The answer is 74.

Assistant
```
def solver():
    # The bakers started with 200 loaves
    loaves_baked = 200
    # They sold 93 in the morning and 39 in the afternoon
    loaves_sold_morning = 93
    loaves_sold_afternoon = 39
    # The grocery store returned 6 loaves.
    loaves_returned = 6
    # The answer is
    answer = loaves_baked - loaves_sold_morning - loaves_sold_afternoon + loaves_returned
    return answer
```

User
Question: [Question]
Chain-of-thought: [CoT]

Figure 7: Few-shot PoT prompt template incorporating our proposed CoT-guided approach for PoT synthesis, where [CoT] serves as a placeholder for natural language reasoning.