Fine-Tuning on Diverse Reasoning Chains Drives Within-Inference CoT Refinement in LLMs

Anonymous ACL submission

Abstract

Requiring a large language model (LLM) to generate intermediary reasoning steps, known as Chain of Thought (CoT), has been shown to be an effective way of boosting performance. Previous approaches have focused on generating multiple independent CoTs, combining them through ensembling or other post-hoc strategies to enhance reasoning. In this work, we introduce a novel approach where LLMs are fine-tuned to generate a sequence of Diverse Chains of Thought (DCoT) within a *single inference step*. By leveraging DCoT, LLMs gain the ability to perform within-inference refinement of reasoning chains without requiring external feedback. Through a rigorous set of experiments spanning a wide range of tasks that require various reasoning types, we show that fine-tuning on DCoT improves performance over the CoT baseline across model families and scales (1.3B to 70B). These improvements are particularly noteworthy for tasks with a large result state space, such as those involving numeric answers. Quantitative analyses and manual evaluations reveal that these gains stem from the models' ability to refine an initial reasoning chain by generating a second, improved chain within the same inference step. Our code and data are publicly available.[1]

1 Introduction

Chain of Thought (CoT; Wei et al. 2022b), the prompting method to generate intermediate reasoning steps to answer a question, is recognized as a simple yet effective mechanism for improving the performance of large language models (LLMs). Given that requiring models to generate intermediary steps improves performance, it stands to reason that requiring models to generate multiple chains could further improve performance. Prior work exploring this idea includes that by Wang et al. (2023), wherein they generate multiple CoTs and ensemble them with a voting mechanism. However, these approaches, and others like them (see Section 2), rely on independently generated CoTs, which prevents the model from accessing previously generated chains during inference. This independence limits the potential for within-inference refinement and the ability to build upon earlier reasoning paths.

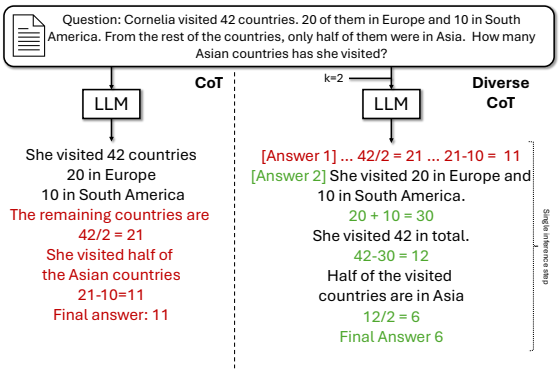

Figure 1: Diverse CoT ($k = 2$) generates k CoTs in a single inference step and selects the correct answer.

Therefore, we present a training method that enables LLMs to generate multiple diverse reasoning chains sequentially in *a single inference step*. Through our experiments, we find that this method drives performance gains by allowing the model to refine subsequent CoTs based on earlier ones. To achieve this, we construct a training dataset of Diverse CoTs (DCoT), where a single question is associated with multiple valid CoTs. While previous studies treated each (question, CoT) pair as a standalone data point (Ho et al., 2023; Huang et al., 2023), we propose to concatenate CoTs into a single sequence, forming training pairs of the format (question, [CoTs]). In this way, models learn to generate multiple CoTs in a single inference step. While all our training CoTs represent correct reasoning chains, we hypothesize that this training regime will enable LLMs to generate sequentially better CoTs—up to a certain number of iterations—

[1] https://anonymous.4open.science/r/DCoT-149B/

as they have access to previous reasoning chains, leading to improved performance.

We demonstrate that fine-tuning using DCoTs improves LLM performance over the CoT baseline by rigorously testing on a range of tasks requiring different types of reasoning across model families and scales (1.3B to 70B). We further identify the subset of tasks, namely those with a large result state space, such as those involving numeric answers, to particularly benefit from our method. In addition, we show that generating a single CoT on the DCoT fine-tuned model yields comparable results to the CoT baseline, while generating two or more CoTs yields clear gains on average across all tasks for all models. This, along with the manual evaluation that we present below, demonstrates that the gains provided by DCoT arise not from random perturbations but from iterative refinement within a single inference step. Additionally, we show that DCoT fine-tuned models can be further augmented by the same methods that boost CoT, such as the self-ensemble of CoTs (Wei et al., 2022b). These results suggest that instruction-tuning datasets can be easily augmented with DCoT data, given that many existing datasets already include CoT examples, often with multiple CoTs per question (Ho et al., 2023; Huang et al., 2023). This makes the creation of DCoT training datasets both practical and efficient. Specifically, the contributions of this work are as follows:

- We introduce a novel method that fine-tunes LLMs to generate multiple reasoning chains within a single inference step, enhancing subsequent chains and boosting performance.
- We rigorously demonstrate the effectiveness of our method on a range of LLM families and sizes across multiple reasoning tasks, identifying task types where it performs best—those with large result state spaces.
- Through a combination of empirical and manual analysis, we show that DCoT achieves gains through within-inference revision of its first CoT without external feedback or prompt optimization, which, to the best of our knowledge, is the first work to do so.

2 Related Works

In this section, we examine related work from three distinct perspectives: (i) prompting methods that enhance CoTs through diversity, (ii) research focused on instruction tuning models using CoTs, and (iii) studies on self-correction in LLMs.

Improving Prompting through diversity. Many works have shown the benefits of generating diverse CoTs and aggregating them (Wang et al., 2023; Zhang et al., 2024; Yoran et al., 2023; Li et al., 2022; Weng et al., 2023; Zhao et al., 2023a,b). In particular, Wang et al. (2023) proposed the creation of *self-assembles* of CoTs to improve LLM's performance, which they call self-consistency. They sample a series of CoTs, select the most consistent answer, and show large performance gains on reasoning tasks. Yoran et al. (2023) extends this work by creating a meta prompt that aggregates the reasoning paths instead of selecting the most common answer. Zhang et al. (2024) propose explicit steps to contrast each CoT and reflect on the final answer. However, none of these works induce LLMs to generate multiple CoTs in the same inference step.

Fine-Tuning on Diverse CoTs. The success of CoT prompting led to the creation of instruction-tuning datasets with CoTs (Chung et al., 2024). Kim et al. (2023) argue that small LMs perform poorly on CoT on unseen tasks compared to large LMs. Hence, they create an instruction-tuning dataset of CoT to equip small LMs with CoT capabilities. Others suggest distilling CoTs from very large language models (vLLMs) (Hsieh et al., 2023; Li et al., 2023a). Ho et al. (2023) also show the benefits of distilling CoTs from these vLLMs and claim that sampling multiple CoTs per question and training on these diverse CoTs is an effective data augmentation technique that improves the performance of distilled models. However, they do not use this diversity at inference time, and unlike us, their method only generates one CoT per question. Huang et al. (2023) show that vLLMs can improve performance on reasoning tasks by self-training on their own CoT generations from sampling.

Self-Correction. Huang et al. (2023) defines it as the ability of an LLM to correct its initial response without relying on external feedback. Most works approach self-correction in LLMs with a system with two steps: one that generates the answer and another one that identifies errors (Shinn et al., 2024; Madaan et al., 2023; Pan et al., 2023; Kim et al., 2024; Weng et al., 2023; Jiang et al., 2023; Du et al., 2024; Paul et al., 2024; Saunders

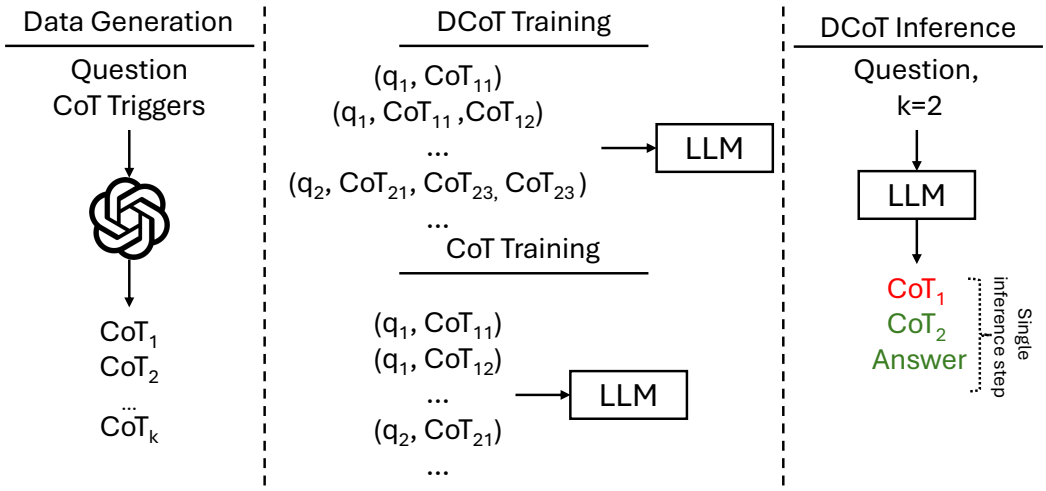

Figure 2: We train on a series of CoTs to make the model learn how to generate multiple CoTs in one inference step.

et al., 2022; Akyurek et al., 2023; Welleck et al., 2023; Estornell et al., 2024). However, Hong et al. (2024) claims that LLMs cannot identify their own errors and Huang et al. (2024); Stechly et al. (2024); Tyen et al. (2023) argue that the self-corrections gains stem from the use of external feedback. Our method differs from these methods in that we generate multiple CoTs in a single-inference step. As we will demonstrate, this access to previous CoTs, enables the model to refine subsequent reasoning chains without explicitly identifying errors.

3 Methods

DCoT. We instruction-tune LLMs on diverse CoTs to generate multiple CoTs in a single inference step. To this end, we devise a DCoT instruction template, where we introduce a set of commands (in brackets) to request the number of CoTs to generate:

Prompt: `[Question]` Question `[Options]` Options `[Number of answers]` k
Response: `[Answer 1]` CoT_1 `[Answer 2]` ... `[Answer k]` CoT_k `[Final answer]` answer

In the input prompt, the instruction `[Options]` provides the candidate answers for multiple-choice questions answering tasks. For other tasks, such as span extraction, this is omitted. In the response, the `[Final answer]` instruction is the convergence mechanism that conditions the model to generate the final answer. We generate DCoT data in the required format using methods described in Section 3.1. For brevity, we refer to instruction-tuned models on DCoT data as `DCoT`.

CoT (Baseline). So as to establish a comparable baseline, we instruction-tune the same LLMs using the more traditional CoT format. To ensure a fair comparison, we use the same reasoning chains as in `DCoT`. As shown in Figure 2, each data point is composed of a question and a CoT, and a question may appear in more than one data point but with a different CoT. In this way, the model leverages CoT diversity at training time, but, unlike in `DCoT`, it does not do so at inference time. For brevity, we refer to these models as `CoT`.

With these two methods, we aim to compare two training regimes that use the same amount of training CoTs and where the only difference lies in the response format. We also do an exploratory analysis of whether we can replicate the results of our DCoT training with in-context learning in very large commercial language models in Appendix C.

3.1 Training Data Generation

We follow the methods set out by Ott et al. (2023) to create CoTs for our CoT and DCoT datasets. We use GPT 3.5 turbo in the zero-shot setting with multiple triggers to generate CoTs. Specifically, *CoT Triggers* are prompt suffixes, such as *"Let's think step by step"* that 'trigger' LLMs to generate CoTs. We use the same triggers as in (Ott et al., 2023). For each question, we select four random CoT triggers. We limit the number of CoTs to four to ensure that the targets fit the context window of the LLMs. We restrict the training data to those reasoning chains that lead to correct answers as determined by the labels provided by the corresponding dataset. We report the prompt templates

and triggers in Appendix J.

Table 7 in Appendix A lists the datasets we use to generate our CoTs and train the models. These datasets were selected following prior works (Wang et al., 2023; Yoran et al., 2023). We have added BoardgameQA (Kazemi et al., 2023) to include logic and ConditionalQA (Sun et al., 2022) to include natural conditional reasoning, which is highly complex, and hence a revision of the answer can be beneficial. With this selection, we cover multiple domains, output spaces, and reasoning abilities. Following prior works (Khashabi et al., 2020; Longpre et al., 2023; Wei et al., 2022a; Tafjord and Clark, 2021), we train all our models in all datasets at the same time to aim for generalizability. We provide more details on Appendix A.

3.2 Models

We train a series of models covering the scaling laws and different families. Concretely, we employ Phi 1.5 (1.3B; Li et al. 2023b), Phi 2 (2.7B; Abdin et al. 2023), LLaMA-2 7B, LLaMA-2 13B (Touvron et al., 2023). With this selection, we can analyze two families and scaling laws within the families. For all of our experiments, we select the non-instruction tuned-based models so as to ensure that the comparison between `DCoT` and `CoT` is fair. This is because instruction-tuning datasets contain CoT data (Touvron et al., 2023), which would make the CoT baseline trained on longer and more diverse CoTs, and hence, the comparison between the two training regimes could be unfair. We also conduct a smaller experiment on LLaMA-2 13B Chat to analyze whether our `DCoT` instruction-tuning method can be applied to already-instruction-tuned models. Lastly, we also run our main experiments on LLaMA-2 70B. However, due to the inference costs, we train it on less data than the other models and only evaluate it on subsets of the evaluation set to show a hint of effectiveness on very large LMs. We refer the reader to Appendix B for details on the training setup of the models.

3.3 Evaluation

Our method is likely to be most effective in scenarios where access to previous CoTs and the corresponding answer is helpful, namely those tasks with a large output space. Therefore, to rigorously evaluate this, we test our method across tasks with varying output space sizes. Specifically, we assess our models on the following task types: Numeric, Span Extraction, Multiple-Choice, Binary, and Symbolic. We use the macro average F1 metric for all these in-domain classification tasks and the squad-metric (Rajpurkar et al., 2016) for these in-domain span-extraction tasks (i.e., ConditionalQA and HotpotQA). So as to select the value of the hyperparameter k, we run our `DCoT` with $k \in [1, 4]$ and select the best k for each dataset based on the dev set (Table 15 in Appendix I reports them). All results are reported on the test set, with the exception of LLaMA02 70B. For LLaMA-2 70B, we only train on a small subset of our training set and also report results on subsets of the dev set with no hyperparameter optimization at all using a k of 2 (the minimum amount of refinements), due to the costs. Further discussions are provided in Appendix B.

3.3.1 Unseen Tasks

Generalization to new tasks remains a challenging problem. For example, Chung et al. (2024) and Kim et al. (2023) shows the need to train on thousands of tasks to achieve performance gains on unseen tasks with CoTs. Since we advocate for the augmentation of instruction-tuning data with our DCoT, we need to evaluate that DCoT at least does not cause performance degradation on unseen tasks (i.e., tasks not used for training). Therefore, we select four challenging unseen tasks encompassing commonsense reasoning (CSQA; Talmor et al. 2019), multiple-choice math (AQuA, Ling et al. 2017), number generation for math (SVAMP, Patel et al. 2021), and number generation for object counting (Suzgun et al. 2023).

3.3.2 Robustness Check

Lastly, we use Big Bench Hard (Suzgun et al., 2023) for a control experiment to evaluate if our method hinders reasoning on complex tasks wherein CoT is beneficial only in models that are significantly larger than the ones we test; in other words, not using CoT is better for small models. This implies that it is extremely difficult for small models to generate correct CoTs for these tasks, and therefore, generating more than one is even more difficult, so it is reasonable to question whether DCoT can reduce performance.

4 Results and Analysis

In this section, we discuss our results with the aim of answering the following questions:

1. **Primary:** Does fine-tuning on DCoT (`DCoT`) provide performance gains compared to fine-

			Numeric	Span-Extraction			Multiple-Choice				Binary	Symbolic
LLM	**Method**	**Avg.**	**GSM8K**	**CQA**	**HQA**	**Avg.**	**ARC**	**BGQA**	**Quartz**	**Avg.**	**StrQA**	**LLC†**
Phi 1.5 (1.3B)	CoT	47.2	34.95	61.21	32.56	46.88	48.7	32.39	72.69	51.26	54.08	**41**
	DCoT (Ours)	**49.39**	**36.85**	**62.48**	**34.81**	**48.64**	**50.01**	**38.6**	**77.39**	**55.34**	**55.97**	39
	CoT + SC	46.48	**40.33**	63.39	33.63	48.51	**53.81**	21.59	75.11	50.17	51.96	32
	DCoT + SC	**49.01**	40.18	**65.23**	**37.79**	**51.51**	53.24	**27.6**	**81.08**	**53.97**	**55.97**	31
Phi 2 (2.7B)	CoT	60.85	56.71	65.13	52.65	58.89	70.87	39.48	82.91	64.42	**61.06**	58
	DCoT	**62.6**	**60.73**	**68.61**	**55.15**	**61.88**	**73.77**	**47.07**	**83.16**	**68.00**	54.34	58
	CoT + SC	61.5	64.97	68.14	55.82	61.98	74.36	28.99	85.2	62.85	**59.51**	55
	DCoT + SC	**65.12**	**68.08**	**70.53**	**58.61**	**64.57**	**76.06**	**44.16**	**86.09**	**68.77**	51.43	**66**
LLaMA2 7B	CoT	58.97	28.51	65.73	53.88	59.80	61.63	**43.13**	79.32	61.36	**64.59**	75
	DCoT	**60.8**	**29.57**	**70.99**	**56.26**	**63.62**	**62.7**	41.91	**81.37**	**61.99**	61.64	**82**
	CoT + SC	**62.9**	33.97	69.92	57.05	63.48	65.98	**46.04**	83.28	**65.10**	**65.99**	81
	DCoT + SC	61.09	**36.01**	**71.36**	**58.35**	**64.85**	**68.53**	28.2	**84.05**	60.26	59.22	**83**
LLaMA2 13B	CoT	64.39	42.53	70.25	60.23	65.24	**71.79**	42.63	**84.82**	66.41	61.85	**81**
	DCoT	**66.18**	**44.28**	**71.56**	**63.52**	**67.54**	71.41	**50.21**	83.29	**68.30**	**65.16**	80
	CoT + SC	66.82	50.27	**72.72**	62.34	67.53	74.82	40.8	**85.84**	**67.15**	67.75	80
	DCoT + SC	**68.12**	**54.51**	72.61	**65.92**	**69.26**	**74.89**	**41.27**	85.07	67.08	**64.65**	**86**
LLaMA2 70B*	CoT	66.96	56	**73.59**	**55.94**	**64.76**	81.69	**44.34**	81.99	69.34	66.15	76
	DCoT	**68.63**	**66**	69.57	49.78	59.67	**89.04**	38.3	**85.99**	**71.11**	**68.34**	**82**
LLaMA2 13B Chat‡	CoT	**64.87**	42.76	**71.71**	60.83	66.27	70.43	**44.39**	84.04	**66.29**	66.78	**78**
	DCoT	64.62	**44.2**	71.59	**63.87**	**67.73**	**72.22**	40.94	**85.43**	66.20	**67.68**	71

Table 1: Comparison of DCoT against CoT on the test sets. We note the larger improvements on tasks with a larger output space (numeric and span extraction) in line with our hypothesis on where our method is more effective. *70B results on the dev set. ‡CoT in chat models include CoT data not used in DCoT. †Results on LLC are not reliable due to the small training and test set sizes.

tuning on CoT (CoT) with a comparable number of reasoning chains? (Section 4.1)

2. Given the inference cost, is DCoT effective on a reasonably small number of reasoning chains (k)? (Section 4.2)
3. Can DCoT be safely included in instruction-tuning datasets without unforeseen consequences on unseen tasks? (Section 4.3) and (Section 4.4)
4. Can DCoT benefit from CoT extensions? (Section 4.5)
5. Are the improvements obtained a result of within-inference refinement or a result of random perturbations? (Section 4.6 and 4.7)

4.1 DCoT is Beneficial on In-Domain Tasks

Our main goal in this experiment is to compare two training regimes: CoT and DCoT. The first two rows of each model in Table 1 compare DCoT with the CoT baseline using the greedy decoding.[2] The first result we observe is that DCoT, on average, is superior across all our main models. Analyzing the results per task type, we also observe that it is particularly effective on multiple-choice tasks, where the Phi models achieve a performance boost of four points. We also observe clear gains on span extraction and numeric tasks. However, performance on binary and symbolic tasks presents a more mixed picture. The challenges with StrategyQA (binary) can be attributed to its constrained output space (i.e., "yes" or "no"), which limits the scope for effective revisions. For symbolic tasks, we use the Last Letter Concatenation dataset, which includes only 300 training and 100 test instances and is our smallest dataset (Appendix A). This small size reduces the reliability of the results, preventing us from drawing definitive conclusions. The overall performance of DCoT across all models confirms the effectiveness of DCoT training.

We also conduct a smaller experiment on general instruction-tuned models (LLaMA2 13B chat). It is worth noting that comparing CoT with DCoT is not completely fair in this setting because this model has already been fine-tuned on CoTs (Touvron et al., 2023); thus, the CoT training is larger

[2] CoinFlip results are omitted because all models achieve perfect scores.

Method	Phi 1.5	Phi 2	LL. 7B	LL. 13B
CoT	47.51±1.77	63.51±.71	59.30±.54	65.41±.91
DCoT@1	47.87±1.71	63.91±2.58	61.28±.50	65.80±.44
DCoT@2	48.63±.67↑	65.33±2.80↑	62.46±.45↑	67.30±.49↑
DCoT@3	48.96±.66	65.30±1.72	62.37±.23	66.92±.59
DCoT@4	48.76±.33	64.89±2.39	62.42±.59	66.70±.55

Table 2: DCoT average performance across different number of CoTs per question on the dev sets.

and more diverse than the DCoT one, as discussed in Section 3.2. Despite this, Table 1 shows that DCoT outperforms CoT in more than half of the datasets. The results of our other smaller experiment on LLaMA 2 70B also hint at the effectiveness of DCoT on larger models.

4.2 One Refinement achieve Gains

In this experiment, we examine the number of chains required to achieve performance gains relative to $k = 1$. Table 2 presents the average performance across all datasets for varying values of k. Notably, on average, $k = 2$ consistently enhances performance across all models, supporting our hypothesis that DCoT effectively refines the initial response. However, increasing k to three or four does not yield further improvements compared to $k = 2$, except on GSM8k (math), where increasing k up to 3 consistently improves performance (see Table 10 in Appendix D). We also observe similar patterns in our smaller experiments on LLaMA 13B Chat and LLaMA 2 70B (also in Table 10). These results suggest that, on average, a single refinement is sufficient to improve performance, making it efficient and cheap. They also align with prior works (Estornell et al., 2024), where 3 and 4 turns do not always improve performance, as we will discuss in Section 5.

DCoT@1 ≈ CoT. An important phenomenon we observe in Table 2 is that the performance of DCoT when generating a single CoT (i.e., DCoT@1) is very similar to the CoT baseline, as expected. This result shows that our DCoT training does not interfere with the regular CoT generation. *Therefore,* DCoT *is a safe replacement to* CoT *in regular instruction-tuning datasets.*

4.3 No Degradation in Unseen Tasks

In this section, we evaluate the performance of DCoT on unseen tasks to ensure that there is no unexpected deterioration. Table 3 shows that DCoT remains effective in Commonsense QA (multiple-choice), where it shows the same trend as in-domain scenarios (i.e., increasing k yields better answers than the first one). However, the models show limited effectiveness in the tasks that require numbers (i.e., AQuA, ObjCnt, and SVAMP). When one generation yields worse answers, the next one usually corrects it (e.g., AQuA on Phi 1.5, SVAMP on LLaMA 13B). These limited gains in unseen math tasks are in line with expectations, given that LLMs' performance on arithmetical reasoning in out-of-domain settings is known to be poor (Qian et al., 2023), and generalization to unseen tasks requires thousands of tasks (Kim et al., 2023), while we only trained on nine. Overall, while our results show that DCoT presents only small improvements in some of the cases, we note that there is no major deterioration.

LLM	Method	CSQA	AQuA	ObjCnt	SVAMP
Phi 1.5	CoT	33.88	20.27	35.60	40.00
	DCoT@1	32.26	21.51	25.20	40.50
	DCoT@2	34.23	17.31	27.60	30.00
	DCoT@3	33.81	22.38	30.80	30.00
	DCoT@4	34.73	22.06	30.00	31.50
Phi 2	CoT	44.29	29.52	54.00	55.00
	DCoT@1	44.15	34.86	58.40	60.50
	DCoT@2	44.13	34.09	56.40	60.50
	DCoT@3	45.99	31.83	57.60	60.00
	DCoT@4	45.43	34.73	56.40	59.50
LLaMA2 7B	CoT	38.41	19.41	34.80	39.50
	DCoT@1	36.94	17.70	40.00	41.50
	DCoT@2	40.79	17.27	39.60	43.00
	DCoT@3	40.67	16.90	36.80	43.00
	DCoT@4	40.43	17.21	37.20	39.00
LLaMA2 13B	CoT	46.55	24.85	45.2	62.50
	DCoT@1	44.62	23.98	46.00	55.00
	DCoT@2	45.48	22.42	47.60	53.50
	DCoT@3	47.42	20.72	52.40	56.50
	DCoT@4	46.45	23.13	54.00	53.50

Table 3: CoT and DCoT across various k performance on unseen tasks.

4.4 DCoT is Robust on Tasks where CoT is Detrimental

We analyze the performance of our method on Big Bench Hard, a benchmark where small models do not benefit from CoTs (Suzgun et al., 2023) to ensure that our method does not lead to unex-

Method	Phi 1.5	Phi 2	LL. 7B	LL. 13B
CoT	28.37	46.7	31.08	36.38
DCoT@1	28.31	44.56	31.23	34.59
DCoT@2	28.07	45.81	31.11	35.94
DCoT@3	28.35	45.92	31.00	36.90
DCoT@4	28.21	46.71	31.13	36.45

Table 4: Results on Big Bench Hard. LL stands for LLaMA2.

pected performance degradation over the CoT baseline. The results from Table 4 show that on these tasks, DCoT exhibits similar performance to CoT, thus demonstrating that DCoT does not lead to deterioration in challenging cases, where CoT might be detrimental. Moreover, we can observe some performance gains on Phi 2 and LLaMA-2 13B when increasing k, further showing the robustness of DCoT tuning. These experiments allow us to conclude that there is no risk in adding DCoT training data to instruction tuning datasets.

4.5 DCoT Benefits from CoT Extensions

DCoT aims to be a new CoT mechanism for instruction-tuning LLMs. To achieve this, we need to confirm that it is a safe drop-in replacement and works with any CoT extension, such as self-consistency (Wang et al., 2023), a different decoding strategy. This decoding method is an add-on that has been shown to increase the performance of CoT across a wide range of tasks by sampling multiple generations and aggregating them by a voting mechanism.

The last two rows of each model (i.e., CoT+SC and DCoT+SC) in Table 1 compare our DCoT with the CoT baseline using the self-consistency decoding with four samples. We observe that our DCoT also benefits from this mechanism and keeps its performance gains over the CoT baseline, showing that our method can be a replacement for CoT in future instruction-tuning datasets. It is also worth noting that, on average, our DCoT with the greedy decoding even outperforms CoT+SC on the Phi models, showing its high performance.

4.6 Manual Analysis: DCoT@2

We conduct a manual evaluation to verify our conclusions based on the quantitative results. Due to the costs of manual evaluations, we conduct this only on LLaMA 7B, the mid-size model of our selection, to have insights that apply to our smaller and larger models. Specifically, we aim to understand how generating a second CoT can improve performance. To this end, we select instances for every dataset where LLaMA 7B with DCoT@1 outputs an incorrect answer while DCoT@2 results in a correct answer. Then, we randomly sample five of those instances per dataset, except in HotpotQA and LLC, where only three instances fulfilled the criteria, resulting in a total of 31 samples.

We note that the first reasoning chain of DCoT@2 might differ from that of DCoT@1 because they are different runs. We find this to be the case in ten instances. This implies that in most cases, the first CoT is the same in both cases. Of these instances where the first reasoning chain is shared, we observe that the second CoT of DCoT@2 exhibits a different reasoning pattern from the first in 15 cases. Therefore, a second, improved CoT allows the model to generate a correct answer when the first CoT results in an incorrect answer. We also observe three cases where the second CoT is the same as the first one but corrects the logical conclusion. In two cases, the second CoT corrects reasoning steps from the first one, and in one case, the error of the first CoT had a slight error in writing the final answer that is corrected with the second one. Overall, these observations confirm that DCoT *revises* its first answer by generating a second CoT. Table 13 in Appendix G shows a few examples.

4.7 Manual Analysis: DCoT@3

Answers Patterns. To understand the behavior of DCoT@3, we use GPT4o-mini to extract the three answers given by the model. We focus on the multiple-choice datasets to facilitate the answer extraction by GPT4o-mini. Table 9 in Appendix H shows that the most common pattern is that the first answer is correct, and the subsequent CoTs reproduce it. This is expected because CoT achieves competitive performance. Moreover, this further evidences that the model does not randomly generate CoTs but tries to refine prior answers. Therefore, if the first answer is correct, the model does need to change it. More interestingly, we find 12 cases where the answer from the third CoT (a wrong answer) is not selected and instead selects prior answers (correct answers). In contrast, we also find 12 cases where the third CoT yields a new answer (the correct one), and the model selects it. These observations suggest that the model is not biased towards the last answer. We also observe 27

cases where a third wrong CoT changes a correct second CoT. However, we find 45 cases where the third CoT corrects the second one. We also provide an analysis of the error cases in Appendix G.

5 Discussion

Training Method. DCoT and CoT are trained on exactly the same amount of CoTs and questions. While the CoT baseline uses data points in the form of [(q, cot_1), (q, cot_2), ...], DCoT uses data points in the form of [(q, cot_1, cot_2, ...) , ...]. In other words, a simple re-organization of the training CoTs into the form of multiple CoTs per label has a major impact on the model's performance. The goal of this training method is not to consistently generate different reasoning chains because if the first one is correct, there is no need to change it. This training method aims to enable LLMs to revise prior CoTs if needed. Importantly, DCoT@1 matches the performance of the CoT baseline, indicating that it is safe to augment existing instruction-tuning datasets with DCoT data, as it will not hinder model performance. Indeed, we envision a typical instruction-tuning dataset containing both DCoT and CoT data.

DCoT revises the first answer. The results of our experiments indicate that DCoT is different from CoT-ensembling methods like self-consistency, which also benefit from generating multiple candidate answers but do so across different and independent inference steps using high-temperature values. DCoT, instead, generates multiple CoTs in a single inference step, and hence, each CoT is aware of prior ones. Thanks to this, the model can revise its answers and improve performance. This is evidenced by DCoT@2 consistently outperforming DCoT@1, while DCoT@3 and DCoT@4 yield similar performance to DCoT@2. These findings show that after the first answer revision and associated performance gains, generating additional CoTs neither improves nor degrades performance, suggesting that the model cannot improve its answer further. This also confirms that the model does not simply generate a sequence of random CoTs but instead revises its first answer. The limitation on scaling k aligns with observations from prior works on self-correction from (Madaan et al., 2023; Kim et al., 2024), where thanks to their external feedback (unlike us), they can achieve gains with up to $k = 4$ and $k = 3$ respectively. Furthermore, Estornell et al. (2024) observe a similar behavior to our DCoT, most gains are obtained with $k = 2$, and increasing k further can lead to marginal gains or even small degradation.

Costs. Lastly, the performance gains achieved by DCoT come at the cost of generating more tokens (i.e., a second CoT). This trade-off between performance and cost is common in prior works, such as in (Wang et al., 2023; Yoran et al., 2023; Zhang et al., 2024). A CoT in our training dataset requires, on average, 124 Phi tokens and 142 LLaMA 2 tokens. Consequently, generating a second CoT to achieve performance gains would incur a negligible cost increment of \$0.00007 per model call,[3] making DCoT a cost-effective alternative to CoT.

6 Conclusions

This work presents Diverse Chain of Thought (DCoT), a new training method that aims to improve LLM's performance on reasoning tasks by generating multiple CoTs in a single inference step to revise answers. Through extensive quantitative experiments, we show the effectiveness and scalability of our method across a wide range of reasoning tasks, model families, and sizes. Furthermore, we show its efficiency by achieving gains with a single revision (i.e., $k = 2$), which incurs negligible costs for the user. We also show that DCoT can be extended with any CoT extension, such as self-consistency, wherein it outperforms CoT similarly extended with self-consistency. Lastly, we show that the reason behind the performance gains of models fine-tuned with DCoT lies in its answer revising abilities, where a single answer revision is enough to achieve clear gains. We leave as future work extending our DCoT fine-tuning to other types of prompting methods such as code prompting (Puerto et al., 2024) or graph of thoughts (Besta et al., 2024), and improving the scalability of k, without external feedback.

Limitations

Our method is limited by the context window of the underlying model. In this work, we have explored generating CoTs up to 4. Although we train the model to generate different chains, this does not guarantee it will generate intrinsically different chains at inference time, and this is not always desired (for example, once the model generates a correct answer, there is no need to change the rea-

[3] Using the cost of LLaMA-2 13B on replicate.com (\$0.5/1M tokens).

soning chain). This is a limitation shared by prior works on generating diverse CoTs.

We limit the generation of the CoTs to a single commercial LLM provider because our preliminary experiments showed performance drops when combining multiple LLM providers. Further research on how to combine multiple LLM providers for distilling to smaller models is interesting and we leave that for future work.

Due to the computational costs, we could not extensively experiment on the 70B model. We could only afford to train with one seed and on a smaller dataset of 900 questions. Similarly, we could only evaluate it on 100 random questions per dataset. Nevertheless, the clear gains we observed on the dev sets, where we do not do any hyperparameter fine-tuning due to its costs, are indicative of the potential of our method on very large language models.

We do not conduct experiments on LLaMA 3 and 3.1 because, with them, we cannot do experiments across model sizes in a single model family as they only have 8B and 70B. LLaMA 3.2, on the other hand, was released after the conclusion of the core experiments and rerunning all experiments would not add new insights that justify the costs. Therefore, we conducted experiments with two families, one for smaller models (1.3B and 2.7B) and another family for larger but still manageable sizes (7B, 13B, and core experiments on 70B).

Ethics and Broader Impact Statement

This work adheres to the ACL Code of Ethics. In particular, all the datasets we used have been shown by prior works to be safe for research purposes. They are not known to contain personal information or harmful content. Our method aims to improve the reasoning abilities of LLMs. Moreover, by generating multiple CoTs in one inference step, we allow the model to explore more reasoning chains and potentially diminish the effects of potentially biased or incorrect CoTs. Because of this, we believe our work can contribute to the safe deployment of LLMs in real-world scenarios.

References

Marah Abdin, Jyoti Aneja, Sebastien Bubeck, Caio César Teodoro Mendes, Weizhu Chen, Allie Del Giorno, Ronen Eldan, Sivakanth Gopi, Suriya Gunasekar, Mojan Javaheripi, Piero Kauffmann, Yin Tat Lee, Yuanzhi Li, Anh Nguyen, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Michael Santacroce, Harkirat Singh Behl, Adam Taumann Kalai, Xin Wang, Rachel Ward, Philipp Witte, Cyril Zhang, and Yi Zhang. 2023. Phi-2: The surprising power of small language models. *Microsoft Ignite 2023*.

Afra Feyza Akyurek, Ekin Akyurek, Ashwin Kalyan, Peter Clark, Derry Tanti Wijaya, and Niket Tandon. 2023. RL4F: Generating natural language feedback with reinforcement learning for repairing model outputs. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7716–7733, Toronto, Canada. Association for Computational Linguistics.

Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17682–17690.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

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*.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. 2024. Improving factuality and reasoning in language models through multiagent debate. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 11733–11763. PMLR.

Andrew Estornell, Jean-Francois Ton, Yuanshun Yao, and Yang Liu. 2024. Acc-debate: An actor-critic approach to multi-agent debate. *Preprint*, arXiv:2411.00053.

Mor Geva, Daniel Khashabi, Elad Segal, Tushar Khot, Dan Roth, and Jonathan Berant. 2021. Did aristotle use a laptop? a question answering benchmark with implicit reasoning strategies. *Transactions of the Association for Computational Linguistics*, 9:346–361.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1:*

Long Papers), pages 14852–14882, Toronto, Canada. Association for Computational Linguistics.

Ruixin Hong, Hongming Zhang, Xinyu Pang, Dong Yu, and Changshui Zhang. 2024. A closer look at the self-verification abilities of large language models in logical reasoning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 900–925, Mexico City, Mexico. Association for Computational Linguistics.

Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. 2023. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 8003–8017, Toronto, Canada. Association for Computational Linguistics.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. 2023. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 1051–1068, Singapore. Association for Computational Linguistics.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2024. Large language models cannot self-correct reasoning yet. In *The Twelfth International Conference on Learning Representations*.

Weisen Jiang, Han Shi, Longhui Yu, Zhengying Liu, Yu Zhang, Zhenguo Li, and James T. Kwok. 2023. Forward-backward reasoning in large language models for mathematical verification. *Preprint*, arXiv:2308.07758.

Mehran Kazemi, Quan Yuan, Deepti Bhatia, Najoung Kim, Xin Xu, Vaiva Imbrasaite, and Deepak Ramachandran. 2023. BoardgameQA: A dataset for natural language reasoning with contradictory information. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Daniel Khashabi, Sewon Min, Tushar Khot, Ashish Sabharwal, Oyvind Tafjord, Peter Clark, and Hannaneh Hajishirzi. 2020. UNIFIEDQA: Crossing format boundaries with a single QA system. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1896–1907, Online. Association for Computational Linguistics.

Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2024. Language models can solve computer tasks. *Advances in Neural Information Processing Systems*, 36.

Seungone Kim, Se Joo, Doyoung Kim, Joel Jang, Seonghyeon Ye, Jamin Shin, and Minjoon Seo. 2023. The CoT collection: Improving zero-shot and few-shot learning of language models via chain-of-thought fine-tuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12685–12708, Singapore. Association for Computational Linguistics.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

Liunian Harold Li, Jack Hessel, Youngjae Yu, Xiang Ren, Kai-Wei Chang, and Yejin Choi. 2023a. Symbolic chain-of-thought distillation: Small models can also "think" step-by-step. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2665–2679, Toronto, Canada. Association for Computational Linguistics.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. Making large language models better reasoners with step-aware verifier. *arXiv preprint arXiv:2206.02336*.

Yuanzhi Li, Sébastien Bubeck, Ronen Eldan, Allie Del Giorno, Suriya Gunasekar, and Yin Tat Lee. 2023b. Textbooks are all you need ii: phi-1.5 technical report. *arXiv preprint arXiv:2309.05463*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22631–22648. PMLR.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh*

Conference on Neural Information Processing Systems.

Sourab Mangrulkar, Sylvain Gugger, Lysandre Debut, Younes Belkada, Sayak Paul, and Benjamin Bossan. 2022. Peft: State-of-the-art parameter-efficient fine-tuning methods. https://github.com/huggingface/peft.

Simon Ott, Konstantin Hebenstreit, Valentin Liévin, Christoffer Egeberg Hother, Milad Moradi, Maximilian Mayrhauser, Robert Praas, Ole Winther, and Matthias Samwald. 2023. Thoughtsource: A central hub for large language model reasoning data. *Scientific Data*, 10(1):528.

Liangming Pan, Michael Saxon, Wenda Xu, Deepak Nathani, Xinyi Wang, and William Yang Wang. 2023. Automatically correcting large language models: Surveying the landscape of diverse self-correction strategies. *arXiv preprint arXiv:2308.03188*.

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

Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. 2024. REFINER: Reasoning feedback on intermediate representations. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1100–1126, St. Julian's, Malta. Association for Computational Linguistics.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Haritz Puerto, Martin Tutek, Somak Aditya, Xiaodan Zhu, and Iryna Gurevych. 2024. Code prompting elicits conditional reasoning abilities in Text+Code LLMs. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 11234–11258, Miami, Florida, USA. Association for Computational Linguistics.

Jing Qian, Hong Wang, Zekun Li, Shiyang Li, and Xifeng Yan. 2023. Limitations of language models in arithmetic and symbolic induction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9285–9298, Toronto, Canada. Association for Computational Linguistics.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

William Saunders, Catherine Yeh, Jeff Wu, Steven Bills, Long Ouyang, Jonathan Ward, and Jan Leike. 2022. Self-critiquing models for assisting human evaluators. *arXiv preprint arXiv:2206.05802*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

Kaya Stechly, Karthik Valmeekam, and Subbarao Kambhampati. 2024. On the self-verification limitations of large language models on reasoning and planning tasks. *Preprint*, arXiv:2402.08115.

Haitian Sun, William Cohen, and Ruslan Salakhutdinov. 2022. ConditionalQA: A complex reading comprehension dataset with conditional answers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3627–3637, Dublin, Ireland. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc Le, Ed Chi, Denny Zhou, and Jason Wei. 2023. Challenging BIG-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 13003–13051, Toronto, Canada. Association for Computational Linguistics.

Oyvind Tafjord and Peter Clark. 2021. General-purpose question-answering with macaw. *arXiv preprint arXiv:2109.02593*.

Oyvind Tafjord, Matt Gardner, Kevin Lin, and Peter Clark. 2019. QuaRTz: An open-domain dataset of qualitative relationship questions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5941–5946, Hong Kong, China. Association for Computational Linguistics.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Gladys Tyen, Hassan Mansoor, Victor Cărbune, Peter Chen, and Tony Mak. 2023. Llms cannot find reasoning errors, but can correct them given the error location. *Preprint*, arXiv:2311.08516.

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, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022a. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.

Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. 2023. Generating sequences by learning to self-correct. In *The Eleventh International Conference on Learning Representations*.

Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. 2023. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, Singapore. Association for Computational Linguistics.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Ori Yoran, Tomer Wolfson, Ben Bogin, Uri Katz, Daniel Deutch, and Jonathan Berant. 2023. Answering questions by meta-reasoning over multiple chains of thought. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5942–5966, Singapore. Association for Computational Linguistics.

Wenqi Zhang, Yongliang Shen, Linjuan Wu, Qiuying Peng, Jun Wang, Yueting Zhuang, and Weiming Lu. 2024. Self-contrast: Better reflection through inconsistent solving perspectives. *arXiv preprint arXiv:2401.02009*.

James Zhao, Yuxi Xie, Kenji Kawaguchi, Junxian He, and Michael Xie. 2023a. Automatic model selection with large language models for reasoning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 758–783, Singapore. Association for Computational Linguistics.

Ruochen Zhao, Xingxuan Li, Shafiq Joty, Chengwei Qin, and Lidong Bing. 2023b. Verify-and-edit: A knowledge-enhanced chain-of-thought framework. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5823–5840, Toronto, Canada. Association for Computational Linguistics.

A Datasets

All the datasets used in this work are exclusively in English language. In particular, we use ARC (Clark et al., 2018), BGQA (Kazemi et al., 2023),CoinFlip (Wei et al., 2022b), ConditionalQA (CQA) (Sun et al., 2022), GSM8K (Cobbe et al., 2021), HotpotQA (HQA) (Yang et al., 2018), LLC (Wei et al., 2022b), Quartz (Tafjord et al., 2019), and StrategyQA (StrQA) (Geva et al., 2021) for training, while we use AQuA (Ling et al., 2017), CommonsenseQA (Talmor et al., 2019), Object Count (a task of Big Bench Hard Suzgun et al. 2023), SVAMP (Patel et al., 2021), and Big Bench Hard for out of domain evaluation. For BGQA, we use the partition `main-3`, the most difficult one requiring 3-hop reasoning skills.

Some of these datasets do not provide a validation set. In those cases, we randomly sample 500 instances from the training set and use them as validation set. Similarly, when a dataset does not provide a test set, we use the validation set as a test and create a validation set from the unused instances from the training set. When the training set is not larger than 1k, we divide the validation set into two. For Last Letter Concatenation (LLC), the training set is very small (350 instances), and the test set is also very small (150), so we pick 50 instances of the test set as validation and 100 as test. We release in our github repository the exact partitions we used.

Table 7 reports the licenses and sizes of the training, dev, and test sets of the datasets we used and Table 8 reports for the out of domain datasets. We use these datasets for research purposes only, fulfilling their intended use.

Due to the large size of LLaMA-2 70B and its computation costs, we trained it on a smaller sample data of 900 questions. Similarly, for inference, we pick a random sample of 100 questions per dataset.

B Experimental Setup

We run all our experiments on a GPU cluster with an Nvidia A180. To run GPT models, we use the Azure OpenAI service and prompt them with the library Langchain.[4] We use Scikit-learn (Pedregosa et al., 2011) for the implementation of the evaluation metrics.

We train all models using LoRA (Hu et al., 2022) with the PEFT library (Mangrulkar et al., 2022) and use vLLM (Kwon et al., 2023) as the inference engine. For training, we load the models with fp8, while for inference, we load them with fp16. We train models for three epochs, save checkpoints for each epoch and select the best checkpoint based on the average results on the dev set.

Due to the challenge of running very large models, such as LLaMA-2 70B, to simplify the evaluation setup. We trained the model with 8-bit quantization and ran the evaluation on 4-bit. Instead of evaluating on the full dev sets, we had to evaluate on a random sample of 100 questions per dataset and only evaluate the last checkpoint. Therefore, we could not conduct hyperparameter tuning either. Because of these challenges, we cannot report results on the test set, and instead, we only report results on the dev set. It is important to emphasize again that we do not conduct any hyperparameter tuning, so the results on the dev set are representative of the performance of our method on large-scale models.

Table 6 reports the best hyperparameters we found on the dev set. Training Phi 1.5 on DCoT takes approximately 12h, Phi 2 20h, LLaMA 7B 35h, LLaMA 13B 51h, and LLaMA 70B 30h. Training on CoT takes 9h for Phi 1.5, 15h for Phi 2, 25h for LLaMA-2 7B, 39h for LLaMA-2 13B, and 13h for LLaMA-2 70B. As expected, DCoT training is slower since the targets are longer. The parameters we use to train the models are reported in Table 5.

Param. name	Value
lora_r	64
lora_alpha	16
lora_dropout	0.1
batch size	4
max_grad_norm	0.3
learning_rate	2e-4
weight_decay	0.001
optim	paged_adamw_32bit
lr_scheduler_type	constant
max_steps	-1
warmup_ratio	0.03
group_by_length	True
max_seq_length	4096
packing	False
seeds	0, 42, 2024
load_in_8bit	True

Table 5: Training parameters

Model	Method	Seed	Epoch
Phi 1.5	CoT	0	2
	DCoT	42	2
Phi 2	CoT	0	3
	DCoT	2024	2
LLaMA2 7B	CoT	0	2
	DCoT	0	3
LLaMA2 13B	CoT	42	3
	DCoT	42	3

Table 6: Best hyperparameters tuned on the dev set.

C Prompting

C.1 Prompting

We conducted exploratory experiments to evaluate the effectiveness of DCoT prompting on commercial black-box LLMs. We use prompts to require models to generate multiple CoTs, compare them, and generate an answer, all in a single inference step. We found that smaller LLMs, with fewer than 100B parameters, lacked the capacity to perform this complex task. When prompted, they often generated the same CoT repeatedly. Even when they did generate multiple CoTs, our manual evaluation revealed they failed to effectively select the correct answer from among them. These results are in line with prior results that indicate that these smaller models are also not the most effective in generating CoTs (Kim et al., 2023). While GPT-4o showed more success, the complexity of the task also heightened its tendency to hallucinate. Consequently, we observed no performance boost through prompting alone and thus focused our experiments on instruction tuning using DCoTs, as detailed in subsequent sections.

The prompts we used with GPT4o for DCoT and CoT are "Generate k different reasoning chains

[4] https://github.com/langchain-ai/langchain

Dataset	Task	Train	Dev	Test	License	Source	Description	Citation
ARC	Multiple choice	1033	294	1150	CC BY-SA 4.0	Link	Science	(Clark et al., 2018)
BGQA	Multiple choice	716	500	1000	CC BY	Link	Logic	(Kazemi et al., 2023)
Coin Flip	Multiple choice	1000	1333	3333	mit	Link	State-tracking	(Wei et al., 2022b)
CQA	Span extraction	958	285	804	CC BY-SA 4.0	Link	Conditional	(Sun et al., 2022))
GSM8K	Num. Generation	1000	500	1319	mit	Link	Math	(Cobbe et al., 2021)
HQA	Span extraction	1000	500	7405	CC BY-SA 4.0	Link	Explicit multi-hop	(Yang et al., 2018))
LLC	Generation	350	50	100	N/A	Link	Symbolic	(Wei et al., 2022b)
Quartz	Multiple choice	953	384	784	CC BY-SA 4.0	Link	Relationships	(Tafjord et al., 2019)
StrQA	Boolean QA	998	343	344	mit	Link	Implicit multi-hop	(Geva et al., 2021))

Table 7: Training datasets. The training size corresponds to our CoT generations to build the DCoT dataset.

Dataset	Task	Dev	License	Source	Citation
AQuA	Multiple choice Math	254	Apache 2.0	Link	(Ling et al., 2017)
CSQA	Multiple choice commonsense	1220	mit	Link	(Talmor et al., 2019)
SVAMP	Generation (numbers)	100	mit	Link	(Patel et al., 2021)
Big Bench Hard	Multiple choice & Generation	6511	mit	Link	(Suzgun et al., 2023)

Table 8: Out of domain datasets.

that answer the question. Make sure that none of the reasoning chains are repeated. Generate each reasoning chain independently, and not based on previous reasoning chains. This means that each reasoning chain must be as different from the others as possible. When generating the different reasoning chains, do so without knowledge of the answer. Each step in each of the reasoning chains must build on the previous steps in that reasoning chain. Once the required number of reasoning chains are generated, generate an answer based on the all the answers generated by all the reasoning chains." and "Generate a reasoning chain that answer the question." In both cases, after generating the CoT, we extracted the answer with the following prompt for SVAMP "Therefore, based on the solution above, extract the number that represents the answer:" and "Therefore, based on the solution above, select one of the options (options) as the answer to the question (just give me the option and nothing else)." for ARC and Quartz.

D Dev Set Results

We report the mean and stardard deviation results from the validation set across threee random seeds in Table 10.

E DCoT across k per Task Type

Table 11 shows the performance of `DCoT` across k per task type. We observe that 2 revisions (i.e., $k = 3$) is beneficial for math reasoning, while multiple choice and span extraction only needs

Pattern	ARC	BGQA	Quartz
AAA → A	226	421	320
AAB → A (o)	6	4	2
AAB → A (x)	2	2	1
AAB → B (o)	2	3	7
AAB → B (x)	6	2	5
ABA → A (o)	4	8	12
ABA → A (x)	5	1	1
ABA → B (o)	0	0	0
ABA → B (x)	0	0	0
ABB → A (o)	0	2	2
ABB → A (x)	2	2	2
ABB → B (o)	10	8	16
ABB → B (x)	17	22	11
ABC → A	1	0	1
ABC → B	2	0	0
ABC → C	2	1	0

Table 9: Answer pattern in DCoT@3 for multiple choice datasets. (o) means correct final answer and (x) incorrect.

one revision. As we mentioned in the main paper, symbolic results are not reliable due to the small training and set sizes of the dataset.

F OOD Results on Tasks requiring Number Generation

Table 12 shows the average results across all out-of-domain datasets that requires number generation for all tasks. We can see that the average performance across all numerical tasks remains relatively similar across all k values for all models except Phi-1.5, the smallest and weakest model.

G Manual Analysis

Appendix G shows two examples of how the second CoT of LLaMA 7B with DCoT corrects the first CoT.

Error Cases. To understand why DCoT@3 does not further improve performance on average compared to $k = 2$, we randomly sample five instances per dataset where DCoT@3 fails to give the right answer and analyze the reasoning chains. We observe that an overwhelming majority of 26/40 exhibit the same second and third CoT and, therefore, do not improve nor decrease performance. We also observe four cases where the second and third CoTs show very similar semantics but different wording, two cases where the first and third CoTs are the same, two cases where all the CoTs are the same, two cases where the CoTs are similar but the arithmetic results are different, three cases where all the CoTs are different, and once case where all the answers are correct but do not match the label. This analysis shows that one of the reasons for the similar average performance of $k = 2$ and $k = 3$ is that DCoT struggles to make a third better answer. This suggests that the model is not generating a series of random CoTs but it is trying to revise its prior answers. This limitation shown in these small fine-tuned models is also seen on very large LMs with prompting Appendix C.

H DCoT@3 Answer Patterns

Table 9 shows the answer patterns of LLaMA 2 7B with DCoT@3 on multiple choice tasks.

I DCoT Best k Parameter

Table 15 shows the best k (i.e., number of CoTs) per model and dataset according to the dev set.

J Data Generation

We report the CoT triggers used to generate the training CoTs in Table 14. As we can see, the triggers condition LLMs to generate answers in different scenarios. This, combined with high-temperature values (0.7), allows GPT 3.5 turbo to generate diverse CoTs. To extract the answers from the CoTs, we used the following format: "{cot} Therefore, the answer (A, B, C, or D) is:" where we change (A, B, C, D) for the corresponding options of the dataset. If the dataset expects a number and not a list of options, we don't give any list of options in the prompt and extract the number with a regular expression. Lastly, for the span extraction datasets, we use the following template: "{text} {question} Answer: {answer} {cot_trigger}." The idea behind this template is to provide the golden answer and prompt the model to generate rationales that explain that answer and use them as CoTs as in (Kim et al., 2023). The total GPT cost to generate the CoTs is $43.68.

LLM	Method	k	Avg	ARC	BGQA	CQA	GSM8K	HQA	LLC	Quartz	StrQA
Phi 1.5	DCoT	1	47.87±1.71	44.13±1.94	39.43±3.91	61.83±.74	36.07±1.70	38.70±3.18	36.00±3.46	71.69±1.73	55.13±.35
		2	48.63±0.67	46.98±2.60	41.94±3.10	60.87±1.14	38.80±3.10	39.79±3.80	30.00±4.00	74.29±2.69	56.40±.87
		3	**48.96±0.66**	47.32±1.66	42.75±1.92	60.75±1.15	39.00±1.71	38.19±2.81	32.67±7.02	75.42±2.38	55.57±1.52
		4	48.76±0.33	46.78±1.14	43.23±2.22	60.16±1.32	38.93±3.31	37.33±2.92	32.67±7.02	75.60±3.32	55.41±1.30
	CoT		47.51±1.77	46.60±2.38	36.65±1.90	59.55±0.61	37.40±3.22	35.28±4.22	36.67±9.02	75.07±2.36	52.84±2.47
Phi 2	DCoT	1	63.91±2.58	75.21±1.84	45.01±3.03	65.39±1.57	56.47±1.68	62.44±2.63	62.67±16.29	82.88±1.09	57.28±2.35
		2	**65.33±2.80**	76.46±2.52	46.89±3.85	65.69±2.12	57.60±1.64	63.71±2.18	66.67±9.02	84.10±1.36	56.44±3.33
		3	65.30±1.72	75.87±1.42	48.06±1.75	65.90±2.02	58.20±1.91	61.66±2.06	68.00±5.29	83.91±1.18	56.28±3.90
		4	64.89±2.39	75.91±2.72	49.11±2.31	65.92±1.01	57.07±1.33	59.86±.96	66.00±8.00	84.09±1.88	56.97±5.00
	CoT		63.51±.71	74.19±.61	42.08±.79	66.92±.29	62.80±3.53	56.45±.78	62.71±3.00	77.92±7.30	66.74±15.54
LLaMA-2 7B	DCoT	1	61.28±.50	59.36±2.29	43.67±.35	65.31±.50	29.73±1.63	62.92±3.16	86.67±2.31	80.63±.92	61.96±1.45
		2	**62.46±.45**	61.63±1.46	43.56±.80	66.05±.80	33.40±.80	63.86±1.23	86.67±3.06	82.11±1.57	62.38±1.21
		3	62.37±.23	60.98±2.37	44.23±.95	66.65±1.21	33.53±.50	63.46±1.46	86.67±1.15	80.89±2.65	62.51±.86
		4	62.42±.59	62.13±3.21	43.85±.45	65.98±2.72	33.33±.50	63.63±2.16	86.00±3.46	82.20±2.78	62.20±1.42
	CoT		59.30±.54	56.54±3.83	41.91±2.32	59.85±3.91	31.93±1.42	57.81±3.73	82.67±3.06	79.24±2.16	64.42±1.52
LLaMA-2 13B	DCoT	1	65.80±.44	71.08±1.95	46.92±3.13	67.97±1.48	42.40±.35	69.97±.61	76.00±5.29	85.07±.40	66.99±2.10
		2	**67.30±.49**	74.85±1.68	46.40±4.13	68.55±1.33	44.53±1.51	72.35±.93	81.33±3.06	84.89±.90	65.46±1.17
		3	66.92±.59	73.29±2.17	48.65±4.48	67.32±2.19	46.53±1.89	71.17±2.07	74.00±15.62	84.89±.52	63.07±.49
		4	66.70±.55	74.17±2.78	46.58±3.03	68.46±1.40	43.93±3.42	70.92±3.54	78.67±3.06	82.46±.99	64.23±2.56
	CoT		65.41±.91	71.66±2.15	44.45±1.53	68.39±1.70	42.67±2.32	66.12±.82	82.00±5.29	82.37±.82	65.64±1.29
LLaMA-2 13B Chat*	DCoT	1	64.53	71.85	47.11	67.37	41.60	70.52	68.00	82.81	66.97
		2	65.95	70.73	47.76	69.16	42.40	71.02	74.00	84.87	67.68
		3	66.10	72.22	46.82	67.48	43.60	72.08	76.00	84.87	65.76
		4	66.17	71.85	45.03	69.70	45.00	71.75	74.00	86.44	65.59
	CoT		**66.27**	70.43	45.36	70.71	44.20	70.11	80.00	82.53	66.78
LLaMA-2 70B*	DCoT	1	66.48	85.80	36.92	65.89	56.00	49.78	78.00	87.00	72.41
		2	**68.63**	89.04	38.30	69.57	66.00	49.78	82.00	85.99	68.34
		3	68.07	86.64	38.55	68.87	60.00	49.78	80.00	88.00	72.70
		4	68.06	88.94	32.62	67.21	66.00	49.78	82.00	85.00	72.92
	CoT		66.96	81.69	44.34	73.59	56.00	55.94	76.00	81.99	66.15

Table 10: Dev set results using 3 random seeds. *One seed for LLaMA-2 13B Chat and 70B.

Model	k	MC Avg.	Span Avg.	Math	Binary	Symbolic*	Total Avg
Phi 1.5	CoT	52.77±1.55	47.41±2.41	37.40±3.22	52.84±2.47	36.67±9.02	47.51±1.77
	1	51.75±2.17	50.26±1.71	36.07±1.70	55.13±.35	36.00±3.46	47.87±1.71
	2	54.40±1.52	50.33±2.25	38.80±3.10	56.40±.87	30.00±4.00	48.63±.67
	3	55.16±1.11	49.47±1.98	39.00±1.71	55.57±1.52	32.67±7.02	48.96±.66
	4	55.20±.88	48.74±2.05	38.93±3.31	55.41±1.30	32.67±7.02	48.76±.33
Phi 2	CoT	65.75±.33	62.26±.78	62.80±3.53	66.74±15.54	62.71±3.00	63.51±.71
	1	67.70±1.74	63.92±1.36	56.47±1.68	57.28±2.35	62.67±16.29	63.91±2.58
	2	69.15±1.72	64.70±1.12	57.60±1.64	56.44±3.33	66.67±9.02	65.33±2.80
	3	69.28±.33	63.78±1.52	58.20±1.91	56.28±3.90	68.00±5.29	65.30±1.72
	4	69.70±1.33	62.89±.63	57.07±1.33	56.97±5.00	66.00±8.00	64.89±2.39
LLaMA 7B	CoT	57.09±.96	58.11±1.12	31.93±1.42	64.42±1.52	82.67±3.06	59.30±.54
	1	61.22±.92	64.12±1.74	29.73±1.63	61.96±1.45	86.67±2.31	61.28±.50
	2	62.43±.56	64.96±1.01	33.40±.80	62.38±1.21	86.67±3.06	62.46±.45
	3	62.03±.21	65.06±1.30	33.53±.50	62.51±.86	86.67±1.15	62.37±.23
	4	62.73±.38	64.81±2.34	33.33±.50	62.20±1.42	86.00±3.46	62.42±.59
LLaMA 13B	CoT	65.80±1.02	65.66±2.28	42.67±2.32	65.64±1.29	82.00±5.29	65.41±.91
	1	67.69±1.20	68.97±.52	42.40±.35	66.99±2.10	76.00±5.29	65.80±.44
	2	68.72±1.49	70.45±1.13	44.53±1.51	65.46±1.17	81.33±3.06	67.30±.49
	3	67.91±1.25	69.85±1.30	46.53±1.89	63.07±.49	74.00±15.62	66.92±.59
	4	68.17±1.58	70.00±1.16	43.93±3.42	64.23±2.56	78.67±3.06	66.70±.55

Table 11: DCoT performance across k for each task type. *Symbolic results are not reliable due to the small size of the dataset LLC.

LLM	Method	Average	AQuA	ObjCnt	SVAMP
Phi 1.5	CoT	31.96	20.27	35.6	40
	DCoT@1	29.07	21.51	25.2	40.5
	DCoT@2	24.97	17.31	27.6	30
	DCoT@3	27.73	22.38	30.8	30
	DCoT@4	27.85	22.06	30	31.5
Phi 2	CoT	46.17	29.52	54	55
	DCoT@1	51.25	34.86	58.4	60.5
	DCoT@2	50.33	34.09	56.4	60.5
	DCoT@3	49.81	31.83	57.6	60
	DCoT@4	50.21	34.73	56.4	59.5
LLaMA2 7B	CoT	31.24	19.41	34.8	39.5
	DCoT@1	33.07	17.7	40	41.5
	DCoT@2	33.29	17.27	39.6	43
	DCoT@3	32.23	16.9	36.8	43
	DCoT@4	31.14	17.21	37.2	39
LLaMA2 13B	CoT	44.18	24.85	45.2	62.5
	DCoT@1	41.66	23.98	46	55
	DCoT@2	41.17	22.42	47.6	53.5
	DCoT@3	43.21	20.72	52.4	56.5
	DCoT@4	43.54	23.13	54	53.5

Table 12: `DCoT` and `CoT` results on out-of-domain numerical reasoning tasks.

Type	Text
1st and 2nd CoTs are the same but the 1st CoT makes an error when generating the answer. 2nd CoT corrects that.	**Question**: Take the last letters of each words in “Billie Paloma Tanner Raul” and concatenate them. **[Answer 1]** 1. Write down the words: Billie Paloma Tanner Raul 2. Take the last letter of the first word "Billie", which is "e". 3. Take the last letter of the second word "Paloma", which is "a". 4. Take the last letter of the third word "Tanner", which is "r". 5. Take the last letter of the fourth word "Raul", which is "l". 6. Concatenate these letters together: "ealr". 7. The result word is "ealr". **[Answer 2]** Sure, let’s break it down step by step: 1. The last letter of "Billie" is "e". 2. The last letter of "Paloma" is "a". 3. The last letter of "Tanner" is "r". 4. The last letter of "Raul" is "l". Now, if we concatenate these letters, we get the word "earl". Therefore, the answer is "earl". **[Final answer]** earl
2nd CoT shows a different reasoing pattern and yields the correct answer.	**Question**: Cornelia likes to travel. She visited already 42 different countries. 20 of them were in Europe and 10 in South America. From the rest of the countries, only half of them were in Asia. How many Asian countries has Cornelia visited? **[Answer 1]** 1. We know that Cornelia visited 42 countries in total. 2. We also know that 20 of those countries were in Europe and 10 were in South America. 3. Therefore, the remaining countries must be half of the total number of countries Cornelia visited, which is 42/2 = 21. 4. We know that Cornelia visited only half of the Asian countries, which means there are 21 - 10 = 11 Asian countries left to be determined. 5. Therefore, Cornelia visited 11 Asian countries in total. **[Answer 2]** Step 1: Let’s start by finding out how many countries Cornelia visited in Europe and South America. We know that she visited 20 countries in Europe and 10 countries in South America, so the total number of countries she visited in these two regions is: 20 + 10 = 30 Step 2: We know that Cornelia visited 42 countries in total. We also know that 30 of these countries were in Europe and South America. Therefore, the number of countries she visited outside of these regions is: 42 - 30 = 12 Step 3: We are told that only half of the countries Cornelia visited outside of Europe and South America were in Asia. This means that the number of countries she visited in Asia is: 12 / 2 = 6 Therefore, Cornelia visited 6 Asian countries. **[Final answer]** 6

Table 13: Examples of self-correction.

CoT Triggers
Answer: Let's think step by step.
Answer: Before we dive into the answer,
Answer: Let's think like a detective step by step.
Answer: Let's think about this logically.
Answer: Let's solve this problem by splitting it into steps.
Answer: The answer is after the proof.
Answer: Let's differentiate using step by step reasoning .
Answer: Let's think step by step using inductive reasoning.
Answer: Let's be concise and think step by step.
Answer: Let's reflect on each answer option step by step.
Answer: Let's think step by step given every option equal consideration.
Answer: Let's think step by step like a scientist.
Answer: Let's use step by step inductive reasoning.
Answer: Let's work by elimination step by step.
Answer: Let's use step by step deductive reasoning.
Answer: Let's work this out in a step by step way to be sure we have the right answer.
because of the following reasons:
Justification:
Here's why:
Here is a list of the reasons:
Now, let's think step by step about the reasons:

Table 14: List of CoT triggers used to generate the training CoTs. The bottom part of the table are the triggers for span extraction datasets.

Dataset	Phi 1.5	Phi 2	LLaMA2 7B	LLaMA2 13B
ARC	4	2	4	4
BGQA	2	4	4	2
ConditionalQA	2	4	4	1
GSM8K	2	3	2	3
HotpotQA	2	2	1	3
LCC	1	2	1	2
Quartz	4	4	2	2
StrategyQA	3	1	1	1

Table 15: Best number of cots (k parameter) for each model and dataset in our best DCoT models according to the dev set.