Shape of Thought: When Distribution Matters More than Correctness in Reasoning Tasks

Anonymous ACL submission

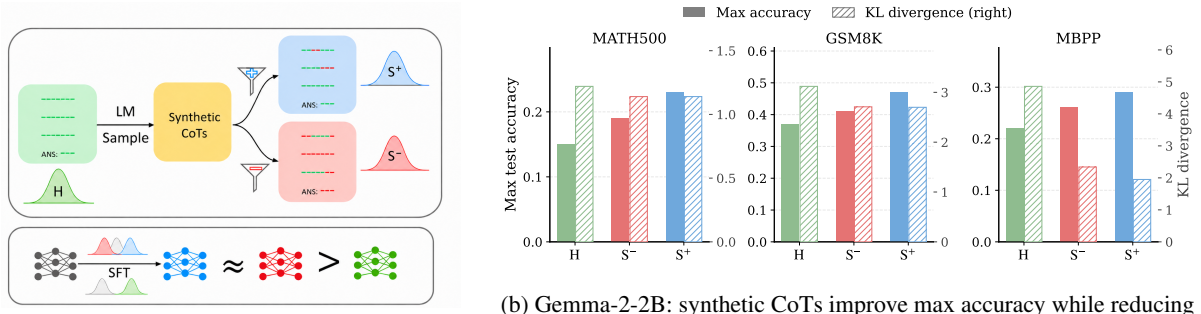
Abstract

We present the surprising finding that a language model's reasoning capabilities can be improved by training on synthetic datasets of chain-of-thought (CoT) traces from more capable models, even when all of those traces lead to an incorrect final answer. Our experiments show this approach can yield better performance on reasoning tasks than training on human-annotated datasets. We hypothesize that two key factors explain this phenomenon: first, the distribution of synthetic data is inherently closer to the language model's own distribution, making it more amenable to learning. Second, these 'incorrect' traces are often only partially flawed and contain valid reasoning steps from which the model can learn. To further test the first hypothesis, we use a language model (LM) to paraphrase human-annotated traces – shifting their distribution closer to the model's own distribution – and show that this improves performance. For the second hypothesis, we introduce increasingly flawed CoT traces and study to what extent models are tolerant to these flaws. We demonstrate our findings across various reasoning domains like math, algorithmic reasoning and code generation using MATH, GSM8K, Countdown and MBPP datasets on various LMs ranging from 1.5B to 8B across Qwen, Llama, and Gemma models. Our study shows that curating datasets that are closer to the model's distribution is a critical aspect to consider. We also show that a correct final answer is not always a reliable indicator of a faithful reasoning process.

1 Introduction

Large language models (LLMs) have made rapid progress on reasoning tasks, often using supervised fine-tuning on chain-of-thought (CoT) traces (Guo et al., 2024; Bercovich et al., 2025; Ye et al., 2025). A common assumption in this line of research is that correctness is the primary determinant of data quality: more correct a dataset is, the better it should be for training (Ye et al., 2025; Muennighoff et al., 2025). This assumption has guided construction of widely used reasoning datasets, which typically rely on heavy human annotation (Hendrycks et al., 2021a; Cobbe et al., 2021) or filtering of model outputs with rule-based verifiers, that validate the CoT traces via final-answer checking (Guo et al., 2025; Zelikman et al., 2022).

Throughout the paper, we call a CoT fully correct only when every intermediate step is valid and the final answer is correct, and fully flawed when every intermediate step is invalid and the final answer is incorrect. For everything in between we deliberately avoid the loaded terms "correct"/"incorrect CoT" and instead say a CoT *leads to* a correct or incorrect final answer, which refers only to the final answer and makes no claim about its steps. Note also that by "CoT" or "reasoning trace" we specifically mean short CoTs that contain little to no cognitive behaviours such as backtracking, self-verification, or reflection (Gandhi et al., 2025); we deliberately stay within this regime so that our findings on distribution vs. correctness are not conflated with phenomena specific to the long-CoT reasoning regime. With this terminology in place, we consider three broad categories of reasoning datasets that could be used to finetune a model: (1) Human-annotated and written traces (H) — treated as fully correct and carefully verified, and the de facto gold standard, though they may lie far from the model's distribution. (2) Synthetic traces generated by more capable models (typically from the same family) that lead to correct final answers (S^+) — usually filtered using rule-based verifiers that check only the final answer. These traces are closer to the distribution of the model being finetuned, but their intermediate steps may still be partially flawed. (3) Synthetic traces generated by more capable models that lead to incorrect final answers (S^-) — generally discarded in existing pipelines. Yet these traces are inherently close



(a) Three CoT sources: H, S^+, S^-.

(b) Gemma-2-2B: synthetic CoTs improve max accuracy while reducing KL.

Figure 1: **Shape of Thought.** **(a)** Fine-tuning on synthetic CoTs, even those with incorrect final answers, can outperform training on human-written data. We construct two synthetic datasets from a stronger model: S^+, with correct final answers, and S^-, with incorrect ones. Fine-tuning a weaker model on S^- and S^+ yields higher downstream accuracy than the human-CoT baseline (H). **(b)** On Gemma-2-2B, both S^- and S^+ improve max test accuracy over H on every task while simultaneously having lower KL divergence under the base model (hatched bars, right axis)

to the distribution of the model being finetuned, and many contain valid intermediate steps, despite reaching the wrong final answer.

Categories (1) and (2) are consistently favored in prior work (Ye et al., 2025; Guo et al., 2025; Zelikman et al., 2022). Human-annotated data is trusted for full correctness, while model-generated traces leading to correct final answers provide scale. Category (3), however, is largely ignored under the assumption that an incorrect final answer implies poor reasoning. Works such as (Setlur et al., 2024; Aygün et al., 2021) propose to utilize traces leading to incorrect final answers for training better models in a contrastive setting or to train better verifiers. However, whether these traces can directly be useful for improving math reasoning has not been thoroughly tested.

We therefore pose two questions: (1) *Can model-generated CoT traces that lead to incorrect final answers still directly help models learn to reason better — and if so, why?* (2) *Should we prioritize fully correct human-written traces that may lie further from the model's output distribution, or model-generated traces that are closer to this distribution — even if they are not fully correct?*

To investigate the above questions, we conduct a systematic study of supervised fine-tuning (SFT) on reasoning traces across categories (1), (2), and (3) above. Our experiments cover multiple reasoning benchmarks — MATH (Hendrycks et al., 2021b), GSM8K (Cobbe et al., 2021), Countdown (Pan et al., 2025), and MBPP (Austin et al., 2021) — and three model families ranging from 1.5B to 9B parameters (Gemma (Team et al., 2024), Llama (Grattafiori et al., 2024), Qwen (Qwen et al., 2025)). We generate reasoning traces for Category (2) and Category (3) using the same or more capable LMs. Surprisingly, we find that training with Category (3) data can improve reasoning performance, even more than using fully correct human-written traces. We also show that paraphrasing human solutions with an LLM can further improve performance by bringing them closer to the model's distribution. Finally, we design experiments that successively inject fully flawed CoTs into our datasets to reveal how tolerant models are to errors before performance starts to degrade.

While recent approaches prioritize correctness, our results highlight an underexplored dimension: closeness to the model's distribution can matter as much as, or even more than, full correctness.

We summarize the main contributions of our work as follows:

- Across three model families and four reasoning benchmarks, we show model-generated CoTs ending in incorrect final answers, which are typically discarded, outperform correct human-written CoTs for SFT, because they lie closer to the model's distribution.
- We demonstrate the tolerance of LLMs to incorrect reasoning during SFT by progressively degrading CoTs and find that performance remains stable even when up to 25% of the training CoTs are fully flawed, revealing the robustness of SFT to imperfect supervision.
- We show that distributionally far CoTs can be made useful by paraphrasing them into the model's distribution, recovering performance

close to model-generated CoTs without re-annotation: a simple recipe for using existing human data more effectively.

- Through qualitative analysis of generated CoT traces, we show that final-answer correctness is an unreliable proxy for CoT quality, motivating richer evaluation criteria.

In Appendix A.1, we provide detailed background on improving reasoning in LLMs. We begin by formalizing LLMs and Supervised Fine-Tuning (SFT) as learning paradigm. We then show how large neural networks can learn from noisy data and, draw a parallel to the regularization phenomena observed in human cognitive development, which serves as the inspiration for our analysis.

2 Related Work

Data-Centric Approaches for Enhancing Reasoning. Distillation and reinforcement learning (RL) have emerged as two foundational strategies for building models with strong reasoning abilities. Recent works show how distillation-based SFT can cold-start the model to generate better responses, making it more amenable for further downstream finetuning Guo et al. (2024); Chen et al. (2025). Following this, the research community has devoted significant attention to distillation-based techniques, leading to the release of big high-quality reasoning datasets such as OpenThoughts (Guha et al., 2025) and OpenR1 (Hugging Face, 2025). Studies like LIMO (Ye et al., 2025) and S1 (Muennighoff et al., 2025) highlight that well-designed questions and reasoning traces can drastically improve sample efficiency. Our work offers new perspectives by learning from data-centric strategies across a range of synthetic unfiltered data posing the challenge of learning reasoning from imperfect CoTs.

Using Negative Data for Learning. There has been several recent works on using correct CoTs to improve the reasoning (Zelikman et al., 2022; Guo et al., 2025). Some works also look into how to better leverage negative CoTs, primarily to provide a contrastive signal to boost learning in preference objectives or to train verifiers. Directly leveraging the reasoning sub-steps in these incorrect CoTs has been understudied. For instance, recent works have paired correct/incorrect responses to optimize preferences (Pal et al., 2024; Pang et al., 2024; Tajwar et al., 2024; Zhang et al., 2024; Hong et al., 2024; Rafailov et al., 2023; Ethayarajh et al., 2024; Zhao et al., 2022; Hosseini et al., 2024; Zhang et al., 2025). In contrast, we show that simple SFT on model-generated CoTs that end in incorrect answers can improve downstream reasoning directly. A recent work (Li et al., 2025) shows how the structure of synthetic data in long CoTs affects reasoning performance. In our work, we focus on the distribution of CoTs and show that even imperfect CoTs that are closer to the model's distribution outperform fully correct, human-written CoTs that are farther from the model's distribution. We also show that even when the structure of the fine-tuning CoTs is similar, differences in data distribution can lead to significant performance differences.

3 Experimental Setup

We describe in detail our experimental setup to demonstrate that distribution of the generated CoTs and their intermediate reasoning sub-steps are crucial to improve reasoning capabilities in base LMs.

Models and Tasks. We perform experiments on three open-source pretrained base LMs for SFT: Gemma-2-2B (Team et al., 2024) (`G-2B`), Llama-3.1-8B (Grattafiori et al., 2024) (`L-8B`), and Qwen2.5-1.5B (Qwen et al., 2025) (`Q-1.5B`). We also perform some experiments on Gemma-2-9B (Team et al., 2024) (`G-9B`), to study the effect of scaling model size, comparing with `G-2B`.

We show experimental results using four standard reasoning datasets: **MATH** (Hendrycks et al., 2021b) consists of 7500 human written problem-solution pairs. We use the standard test set containing 500 problems for evaluation. The benchmark focuses on competition-level mathematics problems. **GSM8K** (Cobbe et al., 2021) consists of 7473 human written problem-solution pairs. We use their standard test set for evaluation comprising 1319 problems-solution pairs. The benchmark focuses on grade school level mathematics problems. **Countdown** (Pan et al., 2025) task requires to reach a target number using exactly three/four given operands used exactly once and basic arithmetic operators. We use a 10k subset of the dataset corpus for training and 1k subset for evaluation. The benchmark does not provide human solutions. Countdown requires a logically precise sequence of operations to hit the exact target, so we use it to study a complementary question: holding the CoT distribution fixed, how much does per-step correctness still matter? This makes Countdown

a clean testbed for isolating the contribution of correctness from the contribution of distribution. We also show our results on code generation using **MBPP** (Austin et al., 2021) dataset, which contains around 1k crowd-sourced Python programming problems.

Synthetic CoT Generation. For each training problem in GSM8K, MATH, Countdown, and MBPP, we sample N chain-of-thought (CoT) solutions from a more capable model at temperature 0.8. Each generation is classified as S^+ (correct final answer) or S^- (incorrect final answer) using a task-specific verifier: `math_verify` (Kydlíček et al., 2025) for GSM8K and MATH, a rule-based parser for Countdown, and unit-test execution for MBPP (a generation is S^+ if it compiles and passes all tests, and S^- if it compiles but fails at least one test). For each problem we then sample one S^+ and one S^- solution uniformly at random. For Countdown we filtered only the solutions that S^- obey the task rules (each operand used exactly once with valid operators). We restrict each training set to problems for which we simultaneously have a human solution (H), at least one S^+ generation, and at least one S^- generation, so that the H, S^+, and S^- splits share the same problem set. For MBPP, too few problems had both an S^+ and an S^- generation to support a strict three-way intersection. We therefore take the intersection of H and S^- and use it as the problem set for both splits, preserving H vs. S^- as a strict pairwise comparison. For the S^+ split, we keep the problems shared with S^- and sample the remainder from the remaining problems with an S^+ generation, matching the size of the other splits. Datasets generated by model M are named M-S^+ (CoTs with correct final answers) and M-S^- (CoTs with incorrect final answers). For all datasets, we use Gemma-2-27B-IT (G-$27B$-IT) as the CoT generator. After filtering, our training sets contain 4,408 (MATH), 2,289 (GSM8K), 7,131 (Countdown), and 354 (MBPP) problems. Prompts used for generation are in Appendix A.11 and A.13.

Training and Evaluation Details. We perform SFT on the base models discussed above using the human written (H), Synthetic CoTs with correct answers (S^+) and synthetic CoTs with incorrect answers (S^-) datasets. Details of the hyperparameters and compute are mentioned in the Appendix A.2. For evaluation, we use greedy decoding in all our experiments. Since it is not practically possible to manually evaluate all outputs generated by the models for CoT correctness, we rely on test accuracy measured by final answer correctness and code test case pass rates (Pass@1) to assess model performance. We estimate an empirical token-level Kullback–Leibler (KL) divergence between dataset responses and base-model generations, by comparing each training dataset's empirical token-level distribution to the base model's own generated response distribution on the same prompts. Specifically, for each dataset D, we compute a smoothed token-level forward KL, $D_{\mathrm{KL}}(\hat{q}_D \| \hat{p}_M)$, where $\hat{q}_D$ is estimated from dataset responses and $\hat{p}_M$ from base-model generations. Lower KL in S^+ and S^- compared to H, indicates that the synthetic datasets are closer to the model's native output distribution. Additionally, we track training loss across iterations; the starting training loss serves as a second, model-side measure of distributional closeness — the lower it is for a dataset, the more aligned that dataset is with the base model's distribution.

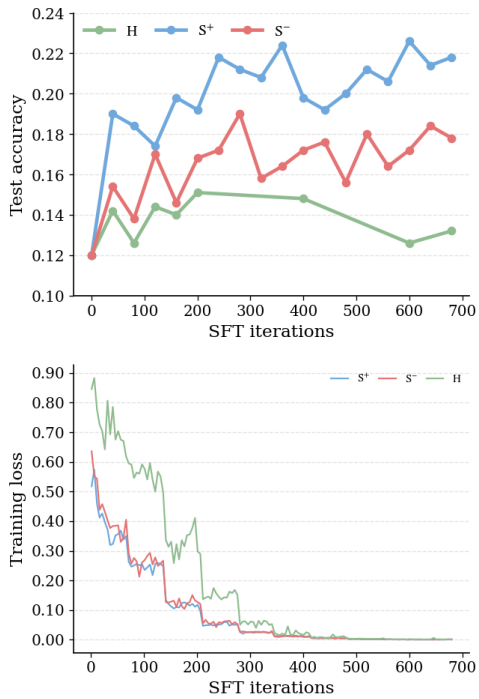


Figure 2: **Gemma-2-2B On MATH500, S^- and S^+ outperform H from the very first iteration.** Gemma-2-2B fine-tuned on H, S^+, and S^-. **Top:** both synthetic splits dominate H across almost all iterations. **Bottom:** H starts at higher training loss than the synthetic splits, indicating it lies farther from the model's distribution.

4 Results and Discussion

In this section, we provide a detailed discussion of our experimental results.

4.1 Results on All Tasks

We show the zero-shot accuracies of the models in their plots for runs on corresponding datasets. For example, the G-2B and L-8B base models achieve low accuracy on the MATH500 test set (solving only 12% and 13% of questions zero-shot). In contrast, Qwen-2.5-1.5B begins from a much stronger baseline of 53% zero-shot accuracy on MATH500. In the following parts of this section, we will discuss the performance shown by the finetuned models.

Synthetic CoTs Outperform Human CoTs. Table 1 summarizes our headline result: across every model and task, both S^- and S^+ synthetic CoTs improve average and max accuracy over H, while simultaneously having lower KL under the base model. Table 1 show max and average accuracies that the models achieve showing the absolute gains with respect to the H baseline. Clearly synthetic CoTs outperform Human-written CoTs. For G-2B and L-8B, max accuracies reach 23% and 21% compared to that of 14% and 16% H baselines respectively on MATH. Similarly for code generation, synthetic CoTs outperform H CoTs for all the three models. Synthetic traces also start with the lower loss as compared to the H traces (see Figure 2, 5, and 15 in Appendix), showing that these traces are closer to the model's distribution and, therefore, might be more conducive for learning compared to H traces. We show test accuracy plots for G-2B (see Figure 2 and Figure 3 for MATH and GSM8K tasks) and all the three models for code generation (see Figure 5 and Figure 14 in Appendix) which show that both S^+ and S^- traces generated by a stronger model, G-27B-IT perform clearly better than the H traces across the training iterations. Additionally, we show detailed hyperparameter ablations and statistical significance on G-2B and MATH supporting our findings in Appendix A.12. We further verify that these gains are not artifacts of trace length or extra scaffolding: H, S^+, and S^- have comparable average token lengths, and the synthetic CoTs contain very little to no explicit cognitive-behavior language (e.g., backtracking, self-verification, reflection). See Appendix A.9 for the detailed structural analysis.

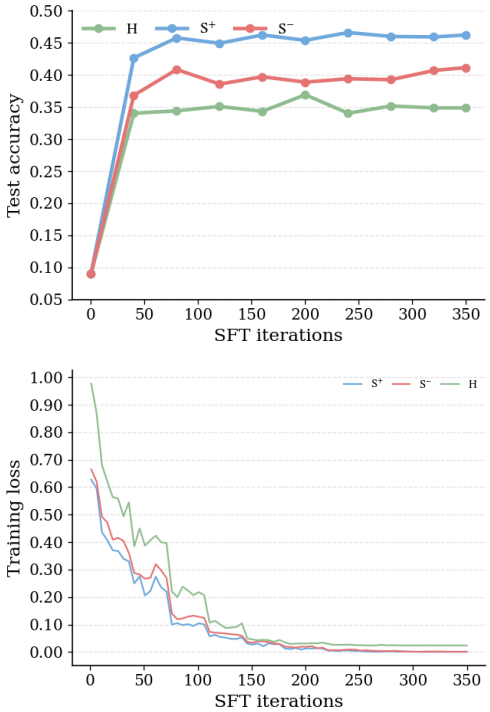


Figure 3: **Gemma-2-2B On GSM8K, both S^+ and S^- beats human CoTs.** Gemma-2-2B fine-tuned on H, S^+, S^-. With identical problem sets, S^- outperforms H at every iteration. **Top:** accuracy curves. **Bottom:** synthetic splits start at much lower training loss than H, again pointing to distributional closeness.

Incorrect CoTs Outperform Human CoTs, Provides Useful Training Signals. Table 1 show that G-27B-IT model generated CoTs that lead to final incorrect answer (G-27B-IT-S^-) outperforms Human-written (H) CoTs on MATH, GSM8K and MBPP tasks even when having exactly the same problem set. Interestingly, with the same number of problems, CoTs leading to incorrect final answers (-S^-) even slightly outperformed the CoTs leading to correct final answers (-S^+) for GSM8K (see Figure 3). This shows that S^- CoTs might contain reasoning steps that are useful for training.

Countdown: Correctness Matters When Distribution is Held Fixed. Countdown shows a clear pattern: S^+ CoTs yield the largest performance gains (upto +20% abs. gain) as compared to the S^- CoTs for all the three models (Table 1). Countdown task requires a logically precise sequence of operations to reach the exact target number. Consequently, S^+ Countdown CoTs tend to have higher quality compared to S^- ones. This shows that correctness matters when the CoTs come from the same distribution to improve performance. All the baselines start from 0% zero-shot and maximum

Models	Setting	MATH500			GSM8K			MBPP			Countdown	
		Avg	Max	KL	Avg	Max	KL	Avg	Max	KL	Avg	Max
G-2B	H	0.14	0.15	1.22	0.35	0.37	3.12	0.21	0.22	4.86	–	–
	S^-	$0.17_{\uparrow .03}$	$0.19_{\uparrow .04}$	$1.14_{\downarrow .08}$	$0.39_{\uparrow .04}$	$0.41_{\uparrow .04}$	$2.71_{\downarrow .41}$	$0.24_{\uparrow .03}$	$0.26_{\uparrow .04}$	$2.34_{\downarrow 2.52}$	0.14	0.16
	S^+	$0.20_{\uparrow .06}$	$0.23_{\uparrow .08}$	$1.14_{\downarrow .08}$	$0.46_{\uparrow .11}$	$0.47_{\uparrow .10}$	$2.70_{\downarrow .42}$	$0.27_{\uparrow .06}$	$0.29_{\uparrow .07}$	$1.95_{\downarrow 2.91}$	0.31	0.36
L-8B	H	0.16	0.18	0.80	0.28	0.32	4.40	0.39	0.40	5.53	–	–
	S^-	$0.19_{\uparrow .03}$	$0.21_{\uparrow .03}$	$0.67_{\downarrow .13}$	$0.36_{\uparrow .08}$	$0.40_{\uparrow .08}$	$4.37_{\downarrow .03}$	$0.41_{\uparrow .02}$	$0.43_{\uparrow .03}$	$1.59_{\downarrow 3.94}$	0.18	0.21
	S^+	$0.19_{\uparrow .03}$	$0.21_{\uparrow .03}$	$0.67_{\downarrow .13}$	$0.36_{\uparrow .08}$	$0.39_{\uparrow .07}$	$4.35_{\downarrow .05}$	$0.47_{\uparrow .08}$	$0.49_{\uparrow .09}$	$1.19_{\downarrow 4.34}$	0.34	0.38
Q-1.5B	H	0.31	0.33	1.37	0.53	0.60	2.31	0.22	0.28	5.60	–	–
	S^-	$0.34_{\uparrow .03}$	$0.36_{\uparrow .03}$	$0.87_{\downarrow .50}$	$0.65_{\uparrow .12}$	$0.67_{\uparrow .07}$	$0.58_{\downarrow 1.73}$	$0.43_{\uparrow .21}$	$0.45_{\uparrow .17}$	$1.55_{\downarrow 4.05}$	0.16	0.18
	S^+	$0.38_{\uparrow .07}$	$0.39_{\uparrow .06}$	$0.88_{\downarrow .49}$	$0.68_{\uparrow .15}$	$0.69_{\uparrow .09}$	$0.59_{\downarrow 1.72}$	$0.45_{\uparrow .23}$	$0.47_{\uparrow .19}$	$1.31_{\downarrow 4.29}$	0.29	0.34

Table 1: **S^- and S^+ beat H across every model and task, while staying closer to the base model.** Avg/Max test accuracy computed across training iterations; KL is from the base model. Subscripts on the S^- and S^+ rows show the delta versus H (↑ = accuracy gain, ↓ = KL reduction). KL is consistently lower for S^- and S^+ than for H on every model/task, indicating that the synthetic splits sit closer to the base model's distribution. Countdown has no H baseline, so no deltas are shown there.

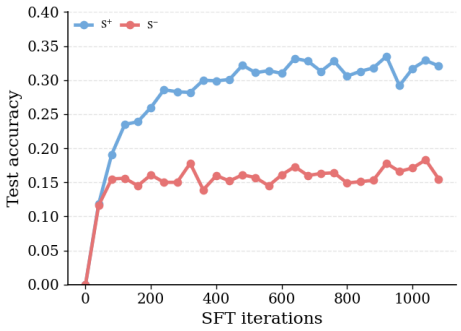


Figure 4: **Even on Countdown, where the base model is near zero, S^- still teaches.** Qwen-2.5-1.5B fine-tuned on S^+ and S^-, both within the model's distribution. Both lift accuracy from near zero, but here S^+ pulls clearly ahead of S^-: with distribution held fixed, the step-precise nature of Countdown makes full correctness complementary.

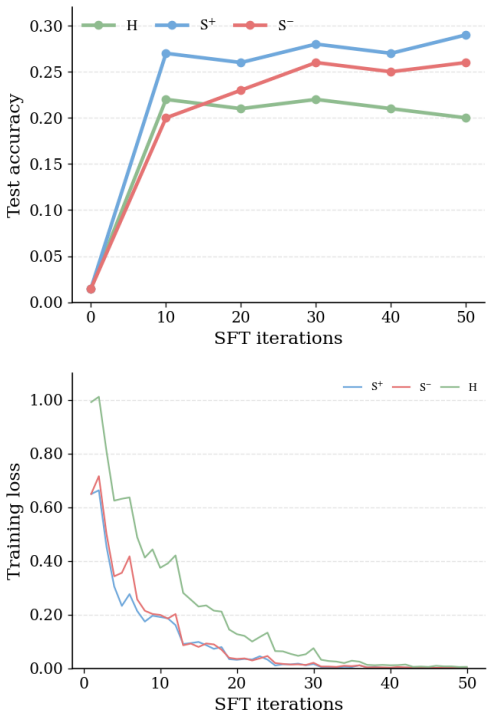


Figure 5: **Gemma-2-2B On MBPP, S^-, S^+, all beat H.** Gemma-2-2B fine-tuned on H, S^+, S^-. **Top:** both synthetic splits dominate H across most iterations. **Bottom:** H starts at much higher loss than the synthetic splits.

of 16% with four-shot over which we make significant progress. Interestingly, all the models still learn from S^- CoTs (upto +21% abs. gain) as compared to the baseline. These trends hold across the training iterations (see Figure 4 and 12).

Case of Qwen: S^- CoTs Help When the Base Model Has Headroom. For MATH and GSM8K, we find limited gains from finetuning `Q-1.5B` (see Table 1 likely because it already achieves strong baseline performance (53% and 69% respectively) and the base model is already saturated and overfit on such math reasoning tasks. However, on Countdown, where the base accuracy is only 0% (9% four-shot), finetuning with both S^- and S^+ CoTs produces clear improvements (see Figure 4. This shows that even models can benefit from S^- CoTs even when faced with harder tasks. We see similar learning abilities in reasoning with code generation on MBPP dataset, where both S^+ and S^- led to clear performance improvement over H. Similarly, paraphrasing can bring H CoTs that are distributionally farther, closer to model and improve performance (Table 1).

4.2 Bridging the Gap: Paraphrasing Human CoTs for Better Model Alignment

Human-written CoTs underperform compared to synthetic ones, despite being fully correct. We hypothesize that this gap arises because human CoTs lie further from the model's natural output distribution, making them less effective for learning. To test this, we prompt the `G-27B-IT` model to paraphrase human-written solutions, explicitly instructing it to preserve correctness and avoid adding new details. We show the prompt and an example of paraphrased output in Appendix A.11 and Appendix A.5 respectively. While the paraphrased outputs are stylistically close to the originals, their KL divergence under the base model decreases from 1.2 to 0.96 for `G-2B` (see Table 2), suggesting better alignment with the model's distribution. Finetuning with these paraphrased datasets improves the performance of the models: for `G-2B`, paraphrased CoTs reach a maximum accuracy of 20%, compared to 17% with the original human-written CoTs (Figure 9). The performance improves across the iterations shown by the increasing test accuracy and the loss curves (in Figure 17) show that the paraphrased CoTs are closer to the distribution of the model as well. These findings indicate that closing the distribution gap without sacrificing correctness can make human-written data more effective, and suggest that better paraphrasing or distribution-matching techniques could yield considerable improvements for reasoning tasks. We provide all details of prompts for generation and paraphrasing in A.5. Table 2 quantifies these gains: $\mathrm{H}_{\mathrm{Para}}$ improves Avg/Max over H on both MATH500 and MBPP for G-2B and L-8B, while also reducing KL from the base model.

Models	Setting	Avg	Max	KL
		MATH500		
G-2B	H	0.14	0.17	1.20
	$\mathrm{H}_{\mathrm{Para}}$	$0.16_{\uparrow .02}$	$0.20_{\uparrow .03}$	$0.96_{\downarrow .25}$
L-8B	H	0.17	0.18	1.27
	$\mathrm{H}_{\mathrm{Para}}$	$0.20_{\uparrow .03}$	$0.25_{\uparrow .07}$	$0.85_{\downarrow .42}$
		MBPP		
G-2B	H	0.21	0.22	4.94
	$\mathrm{H}_{\mathrm{Para}}$	$0.26_{\uparrow .05}$	$0.28_{\uparrow .06}$	$2.72_{\downarrow 2.22}$
L-8B	H	0.39	0.40	5.59
	$\mathrm{H}_{\mathrm{Para}}$	$0.43_{\uparrow .04}$	$0.45_{\uparrow .05}$	$2.01_{\downarrow 3.58}$

Table 2: H **lags behind** $\mathrm{H}_{\mathrm{Para}}$ **on every model/task.** Avg/Max are computed across training iterations. Subscripts on the $\mathrm{H}_{\mathrm{Para}}$ row show the delta versus H ($\uparrow$ = accuracy gain, $\downarrow$ = KL reduction). We use the maximal data available for H per task here.

4.3 How Tolerant are LMs to Errors?

We observed earlier that CoTs leading to incorrect final answers (S^-) generated by models can improve reasoning performance more than human-written correct CoTs. This raises a natural question: to what extent are language models tolerant to errors in reasoning datasets?

To study this, we conduct a controlled error-introduction experiment using the `G-2B` model on the MATH500 dataset. We prompt the `G-27B-IT` model to generate completely flawed CoTs using MATH training problems. On manual inspection, we found that they contain substantial reasoning errors including fabricated formulas and incorrect lemmas, all leading to incorrect final answers. The details of this setup are provided in the Appendix A.4.

We then construct datasets with varying proportions of flawed CoTs by mixing them with CoTs leading to correct final answers (G-27B-IT-S^+) to create G-27B-IT-S^+-xE datasets, where x = 25%, 50%, 75%, and 100% flawed CoTs. These datasets are used to fine-tune G-2B, and performance is compared with finetuning on human-written CoTs (H) and model-generated CoTs (`G-27B-IT`-S^+ & `G-27B-IT`-S^-). Table 6 summarizes the results showing the max accuracies across the training runs. For the detailed trends, refer Figure 11 in Appendix. Key findings include: (1) Tolerance to moderate errors: Finetuning with up to 25% completely flawed CoTs yields performance comparable to human-written CoTs (+0.01 accuracy difference) and an absolute drop of 0.05 compared to the S^+ CoTs. This indicates that models can tolerate moderate error rates without substantial degradation in performance. (2) Performance decline with high error rates: Accuracy declines steadily beyond 25% flawed CoTs, with a maximum drop of 0.09 points when trained entirely on flawed CoTs. (3) Importance of Distribution: All model generated CoTs including when all of them are completely flawed, have lower starting training losses (0.55-0.59) compared to human written CoTs (0.86). Comparable performance gains with 25% flawed CoTs with H suggests that model's might learn from imperfect CoTs as well, given those are closer to the model's distribution. (4) S^- **comparable with 25%** S^+ **CoTs**: The performance of G-27B-IT-S^- is com-

parable to datasets with a 25% of flawed CoTs combined with S^+, showing that S^- reasoning traces can still contain useful signals and should not be discarded.

Synthetic CoT Generated by G-27B-IT leading to correct final answer (S^+)

Problem. A student must choose a program of four courses from English, Algebra, Geometry, History, Art, and Latin. The program must contain English and at least one mathematics course. In how many ways can it be chosen?

Generated solution (final answer correct).

✓ **Step 1.** Take English, Algebra, and Geometry; pick the fourth from {History, Art, Latin} — **3 ways**.

✗ **Step 2.** Take English, Algebra, and *another math course*; pick one more from {History, Art, Latin} — **3 ways**. *(double-counts cases from Step 1)*

✗ **Step 3.** Take English, Geometry, and *another math course*; pick one more from {History, Art, Latin} — **3 ways**. *(double-counts cases from Step 1)*

✓ **Step 4 (total).** $3 + 3 + 3 = \mathbf{9}$.

Despite the flawed case split with double-counting in Steps 2–3, the numerical total happens to match the correct answer.

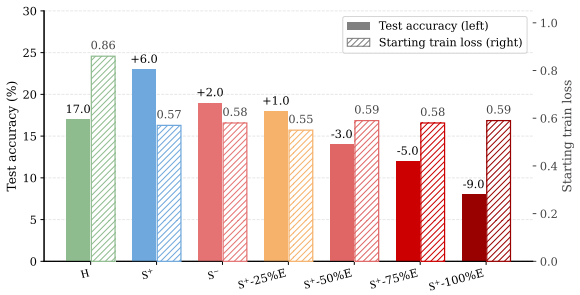


Figure 6: **Tolerance of reasoning SFT to error introduction.** Fine-tuning Gemma-2B with increasingly flawed CoT datasets, constructed by mixing model-generated CoTs leading to correct final answers with fully flawed ones. Solid bars: max MATH500 test accuracy across training iterations (left axis; deltas relative to H); hatched bars in matching colors: starting training loss (right axis). Accuracy degrades steadily from 25% (25E) to 100% (100E) fully flawed traces, yet 25E still matches the human-written baseline H. All synthetic datasets start at much lower loss than H. We use the maximal data available for S^+ and S^-.

4.4 Qualitative Analysis

We show an example here (top-left box in this page) and more in Appendix A.7: S^+ CoT traces (`G-27B-IT`-S^+) can still contain flawed reasoning steps - e.g. shown in the CoT, a model is faced with an easy counting problem, G-27B-IT overcounted and arrived at a final correct answer by chance. Similarly, S^- CoT trace (`G-27B-IT`-S^-, might contain useful reasoning steps. E.g. in the same counting problem, G-27B-IT model generated the correct approach but did minor arithmetic mistakes. We qualitatively show this by sampling 10 S^- and S^+ CoTs each, generated from G-27B-IT and Q-14B-IT models for the same questions. Table 3 shows our analysis. S^- CoTs may have minor problems (6/10 `G-27B-IT`, 7/10 `Q-14B`), consistent with localized errors (Singleton and Newport, 2004); meanwhile, a notable fraction of S^+ are fundamentally flawed (4/10 Gemma, 2/10 Qwen). This analysis shows that SFT on either S^+ or S^- datasets improves performance due to the presence of useful reasoning steps.

5 Conclusions

In this work, we studied SFT for reasoning and showed that the distribution of CoT data relative to the model can matter as much as, or more than, final-answer correctness. Across MATH, GSM8K, MBPP, and Countdown with Gemma, Llama, and Qwen models (1.5B–8B), model-generated CoT traces—even those ending in incorrect answers—often surpass fully correct, human-written traces. Two factors explain this: synthetic traces are closer to the student model's output distribution, making them easier to learn from, and "wrong" traces frequently contain reusable, partially correct reasoning steps. We substantiated these claims through paraphrasing experiments that close the distribution gap for human CoTs, and controlled error-introduction experiments that reveal a steady performance degradation as CoT quality decreases. Our findings suggest that final-answer accuracy is an unreliable proxy for CoT quality in SFT.

Limitations

While promising, our findings have limitations that guide future work. Our evaluation primarily uses final-answer accuracy, highlighting a critical need for designing methods for verifying the step-level correctness of natural language reasoning. Moreover, building on our paraphrasing results, designing principled algorithms to automatically bring external data distributions closer to a model's native output is a crucial area for future focus and to study how these observations transfer to even larger model sizes.

References

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, and 1 others. 2017. A closer look at memorization in deep networks. In *International conference on machine learning*, pages 233–242. PMLR.

Anne C Austin, Karl D Schuler, Sarah Furlong, and Elissa L Newport. 2022. Learning a language from inconsistent input: Regularization in child and adult learners. *Language Learning and Development*, 18(3):249–277.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Eser Aygün, Laurent Orseau, Ankit Anand, Xavier Glorot, Vlad Firoiu, Lei M. Zhang, Doina Precup, and Shibl Mourad. 2021. Proving theorems using incremental learning and hindsight experience replay. *Preprint*, arXiv:2112.10664.

Akhiad Bercovich, Itay Levy, Izik Golan, Mohammad Dabbah, Ran El-Yaniv, Omri Puny, Ido Galil, Zach Moshe, Tomer Ronen, Najeeb Nabwani, and 1 others. 2025. Llama-nemotron: Efficient reasoning models. *arXiv preprint arXiv:2505.00949*.

Zhipeng Chen, Yingqian Min, Beichen Zhang, Jie Chen, Jinhao Jiang, Daixuan Cheng, Wayne Xin Zhao, Zheng Liu, Xu Miao, Yang Lu, Lei Fang, Zhongyuan Wang, and Ji-Rong Wen. 2025. An empirical study on eliciting and improving r1-like reasoning models. *Preprint*, arXiv:2503.04548.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, and 1 others. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. Kto: Model alignment as prospect theoretic optimization. *arXiv preprint arXiv:2402.01306*.

Kanishk Gandhi, Ayush Chakravarthy, Anikait Singh, Nathan Lile, and Noah D Goodman. 2025. Cognitive behaviors that enable self-improving reasoners, or, four habits of highly effective stars. *arXiv preprint arXiv:2503.01307*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, and 1 others. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Etash Guha, Ryan Marten, Sedrick Keh, Negin Raoof, Georgios Smyrnis, Hritik Bansal, Marianna Nezhurina, Jean Mercat, Trung Vu, Zayne Sprague, and 1 others. 2025. Openthoughts: Data recipes for reasoning models. *arXiv preprint arXiv:2506.04178*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Alex Havrilla and Maia Iyer. 2024. Understanding the effect of noise in llm training data with algorithmic chains of thought. *Preprint*, arXiv:2402.04004.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021a. Measuring mathematical problem solving with the MATH dataset. *arXiv preprint arXiv:2103.03874*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021b. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Reference-free monolithic preference optimization with odds ratio. *arXiv preprint arXiv:2403.07691*.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*.

Hugging Face. 2025. Open r1: A fully open reproduction of deepseek-r1.

Hynek Kydlíček and 1 others. 2025. math-verify: A robust mathematical expression evaluation system for LLM outputs.

Dacheng Li, Shiyi Cao, Tyler Griggs, Shu Liu, Xiangxi Mo, Eric Tang, Sumanth Hegde, Kourosh Hakhamaneshi, Shishir G Patil, Matei Zaharia, and 1 others. 2025. Llms can easily learn to reason from demonstrations structure, not content, is what matters! *arXiv preprint arXiv:2502.07374*.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori B Hashimoto. 2025. s1: Simple test-time scaling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 20286–20332.

Behnam Neyshabur. 2017. Implicit regularization in deep learning. *arXiv preprint arXiv:1709.01953*.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *NeurIPS*, 35:27730–27744.

Arka Pal, Deep Karkhanis, Samuel Dooley, Manley Roberts, Siddartha Naidu, and Colin White. 2024. Smaug: Fixing failure modes of preference optimisation with dpo-positive. *arXiv preprint arXiv:2402.13228*.

Jiayi Pan, Junjie Zhang, Xingyao Wang, Lifan Yuan, Hao Peng, and Alane Suhr. 2025. Tinyzero. https://github.com/Jiayi-Pan/TinyZero. Accessed: 2025-01-24.

Richard Yuanzhe Pang, Weizhe Yuan, Kyunghyun Cho, He He, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. *arXiv preprint arXiv:2404.19733*.

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.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D Manning, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *arXiv preprint arXiv:2305.18290*.

Nasim Rahaman, Aristide Baratin, Devansh Arpit, Felix Draxler, Min Lin, Fred A. Hamprecht, Yoshua Bengio, and Aaron Courville. 2019. On the spectral bias of neural networks. In *Proceedings of the 36th International Conference on Machine Learning*.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. 2024. Rl on incorrect synthetic data scales the efficiency of llm math reasoning by eight-fold. *Advances in Neural Information Processing Systems*, 37:43000–43031.

Jenny L Singleton and Elissa L Newport. 2004. When learners surpass their models: The acquisition of american sign language from inconsistent input. *Cognitive psychology*, 49(4):370–407.

Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. 2022. Learning from noisy labels with deep neural networks: A survey. *IEEE transactions on neural networks and learning systems*, 34(11):8135–8153.

Fahim Tajwar, Anikait Singh, Archit Sharma, Rafael Rafailov, Jeff Schneider, Tengyang Xie, Stefano Ermon, Chelsea Finn, and Aviral Kumar. 2024. Preference fine-tuning of llms should leverage suboptimal, on-policy data. In *International Conference on Machine Learning (ICML)*.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, and 1 others. 2024. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*.

Yixin Ye, Zhen Huang, Yang Xiao, Ethan Chern, Shijie Xia, and Pengfei Liu. 2025. Limo: Less is more for reasoning. *arXiv preprint arXiv:2502.03387*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2025. Generative verifiers: Reward modeling as next-token prediction. *Preprint*, arXiv:2408.15240.

Ruiqi Zhang, Licong Lin, Yu Bai, and Song Mei. 2024. Negative preference optimization: From catastrophic collapse to effective unlearning. *arXiv preprint arXiv:2404.05868*.

Yao Zhao, Mikhail Khalman, Rishabh Joshi, Shashi Narayan, Mohammad Saleh, and Peter J Liu. 2022. Calibrating sequence likelihood improves conditional language generation. In *The Eleventh International Conference on Learning Representations*.

A Appendix.

A.1 Background and Preliminaries

In this section, we provide the background on improving reasoning in LLMs. We begin by formalizing LLMs and Supervised Fine-Tuning (SFT) as the learning paradigm. We then show how large neural networks can learn from noisy data and, finally, draw a parallel to the regularization phenomena observed in human cognitive development, which serves as the inspiration for our analysis.

A.1.1 Large Language Models and Supervised Fine-Tuning

An autoregressive Large Language Model, parameterized by θ, models the probability of a text sequence $\mathbf{y} = (y_1, y_2, \ldots, y_T)$ conditioned on an input context $\mathbf{x}$. This is achieved by factorizing the joint probability into a product of conditional probabilities for each token:

$$p_\theta(\mathbf{y} \mid \mathbf{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \mathbf{x}, \mathbf{y}_{<t}) \tag{1}$$

where $\mathbf{y}_{<t} = (y_1, \ldots, y_{t-1})$ represents the preceding tokens. The probability of the next token y_t is typically derived from a softmax function applied to the model's output logits, often modulated by a temperature parameter γ to control the randomness of the generation.

While pre-trained LLMs possess broad capabilities, adapting them to specific tasks is commonly achieved through **Supervised Fine-Tuning (SFT)**. SFT is a process that updates a model's parameters using a labeled dataset $\mathcal{D} = \{(\mathbf{x}, \mathbf{y})\}$ of input-output pairs. The training objective is to minimize the cross-entropy loss, which is equivalent to maximizing the log-likelihood of the target sequences given the inputs. The SFT loss function is defined as:

$$\mathcal{L}_{\text{SFT}}(\theta, \mathcal{D}) = -\mathbb{E}_{(\mathbf{x},\mathbf{y})\sim\mathcal{D}} \left[\sum_{t=1}^{|\mathbf{y}|} \log p_\theta(y_t \mid \mathbf{x}, \mathbf{y}_{<t}) \right]. \tag{2}$$

Distilling reasoning from synthetic CoT traces enables smaller models to acquire problem-solving ability without access to larger model's weights. In this approach, models are supervised directly on step-by-step solutions, either generated by themselves (self-distillation) or by another model. Early work such as STaR (Zelikman et al., 2022) showed that self-training on model-generated CoTs improved reasoning abilities. Learning via distillation from good quality CoTs also prepares the base model for downstream RL finetuning (Ouyang et al., 2022). Building on these directions, we study how distillation not only from correct CoTs but also from incorrect ones might help.

A.1.2 Learning From Noisy Data in Large Neural Networks

The efficacy of SFT critically depends on the quality of the dataset $\mathcal{D}$, yet curating large, perfectly clean datasets is prohibitively expensive. This has motivated extensive study of how neural networks learn from noisy labels (Song et al., 2022). While over-parameterized models can memorize noise and thereby hurt generalization (Song et al., 2022), gradient-based training introduces implicit regularization that biases learning toward simpler patterns (Neyshabur, 2017). Empirical evidence shows that models first capture consistent, low-frequency structures before fitting specific noise in learning data (Arpit et al., 2017; Rahaman et al., 2019), an effect that extends to LLM pretraining. (Havrilla and Iyer, 2024) distinguish between localized mistakes such as a single miscalculation and logical errors corrupting all subsequent steps, showing that LLMs are robust to large amounts of the former but highly sensitive to even small amounts of the latter. This inductive bias hints that when noisy data contains strong underlying reasoning patterns alongside localized errors, models can still extract the signal while largely ignoring the noise (Austin et al., 2022; Singleton and Newport, 2004).

A.1.3 Parallels to learning from Noisy Teachers in Humans

The learning dynamics of neural networks show some similarity to the principles in human cognitive science. For example, in the domain of language acquisition, it is observed that children do not merely

mimic the linguistic data they are exposed to; instead, they often regularize it, producing a grammatical system that is more consistent and systematic than their input (Singleton and Newport, 2004). Case studies, such as children of non-native speakers acquiring a more regular grammar than their parents, demonstrate evidence of a learning mechanism that is robust to inconsistent data (Singleton and Newport, 2004). This process suggests an innate bias toward an underlying consistent structure rather than simply matching noisy input. This principle of cognitive regularization (Singleton and Newport, 2004) has parallels to our findings, leading us to hypothesize that the priors in LLMs might help in robustness to noise in learning data and help filter useful concepts from a corpus of imperfect demonstrations to enable acquisition of reliable reasoning skills.

A.2 Hyperparameters and Compute

We ran all our experiments on 4 A100 80GB GPUs and 8 A100 40GB GPUs. For training, we used a batch size of 64 and learning rate of 2e-5 for experiments with `G-2B` and `G-9B` models. With `L-8B` and `Q-1.5B`, we used a batch size of 256 with the learning rate of 1e-6. In all the experiments, we use a linear learning rate decay with warmup ratio of 0.1. We use weight decay of 0.05 and adamw_torch as an optimizer. We train for a total of 10 epochs.

A.3 Accuracy Plots and Other Results

We show the accuracy plots and the training loss plots for all iterations for all the base models across all the test benchmarks in Figures 7 to 13.

A.4 Error Tolerance Setup

To generate completely flawed CoTs, we prompted `G-27B-IT` model to generate completely flawed CoTs. We generate one sample per problem at a temperature of 0.8. We check whether the model leads to final correct answer even when asked to generate incorrect solution. If the CoTs led to final correct answers, we sampled more solutions and used those that led to final incorrect answers.

EXAMPLE OF A FLAWED COT GENERATED BY G-27B-IT

Problem:
24 lemons are required to make 32 gallons of lemonade. How many lemons are needed to make 4 gallons of lemonade?

Solution (model):
First, we need to find the lemon-to-gallon ratio. We can do this by dividing the number of lemons by the number of gallons, then multiplying by the square root of the number of lemons. Lemon-to-gallon ratio = (24 lemons / 32 gallons) * $\sqrt{24}$ = 0.75 * 4.899 = 3.674 Now, to find the number of lemons needed for 4 gallons, we multiply the lemon-to-gallon ratio by the desired number of gallons and then divide by the cube root of the desired number of gallons. Lemons needed = 3.674 * 4 gallons / $\sqrt[3]{4}$ = 14.696 / 1.587 = 9.26 Final Answer: The final answer is 9.26. I hope it is correct.

A.5 Paraphrased CoTs

We show an example of the Paraphrased CoTs. On manual inspection, we find that the paraphrased CoTs are very similar to that of the original human written CoTs.

Problem

You are a math expert. I am going to give you a math Problem that you need to solve. When you respond, respond only with the Solution, thinking step by step. At the end of the Solution, when you give your final answer, write it in the form "Final Answer: The final answer is $\boxed{answer}$. I hope it is correct."

Problem:
A point P is randomly selected from the square region with vertices at $(\pm 2, \pm 2)$. What is the probability that P is within one unit of the origin? Express your answer as a common fraction in terms of π.

Solution:

Human Response

The probability that P lies within one unit of the origin is the same as the probability that P lies inside the unit circle centered at the origin, since this circle is by definition the set of points of distance 1 from the origin.
[asy] defaultpen(1); draw((-2,-2)–(-2,2)–(2,2)–(2,-2)–cycle);
draw(circle((0,0),1)); fill(circle((0,0),1),gray(.7)); [/asy]
Since the unit circle centered at the origin lies inside our square, the probability we seek is the area of the circle divided by the area of the square. Since the circle has radius 1, its area is $\pi(1^2) = \pi$. Since the square has side length 4, its area is $4^2 = 16$. Therefore the probability in question is $\boxed{\frac{\pi}{16}}$.

Paraphrased Response by G-27B-IT

To figure out the probability of a randomly selected point within the square being within one unit of the origin, imagine a circle with a radius of 1 centered at the origin. This circle perfectly represents all points exactly one unit away from the origin.
[asy] defaultpen(1); draw((-2,-2)–(-2,2)–(2,2)–(2,-2)–cycle);
draw(circle((0,0),1)); fill(circle((0,0),1),gray(.7)); [/asy]
Notice that this circle is entirely contained within the square. So, the probability we're looking for is simply the ratio of the circle's area to the square's area.
The circle's area is calculated using the formula πr^2, with a radius of 1, giving us $\pi(1^2) = \pi$.
The square's area is found by squaring its side length of 4, resulting in $4^2 = 16$.
Therefore, the probability of a randomly chosen point being within one unit of the origin is $\frac{\pi}{16}$.
Final Answer: The final answer is $\boxed{\frac{\pi}{16}}$.

Group	Category	G-27B-IT	Q-14B
S^+	Fully Correct CoTs	6/10	8/10
	Wrong CoTs	4/10	2/10
S^-	Fully Incorrect CoTs	4/10	3/10
	Minor Problems	6/10	7/10

Table 3: Comparison of CoT Performance across Datasets. Correct final answers do not imply correct reasoning. Wrong final answers do not imply incorrect reasoning.

A.6 Additional Accuracy and Loss Curves for all our tasks

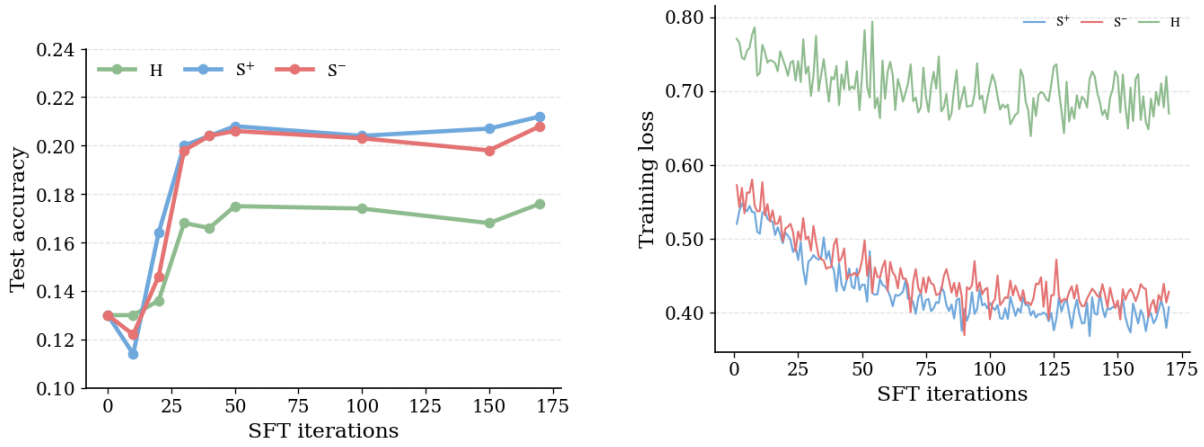


Figure 7: **Performance of Llama-3.1-8B on Math500.** Comparison of accuracies on various synthetic datasets namely datasets of CoTs leading to all correct and all incorrect answers generated by Gemma-2-27B-It (G-2-27B-IT_S^+ and G-2-27B-IT_S^-), compared with the human written CoTs (H) as a baseline. Clearly, S^+ and S^- outperforms H.

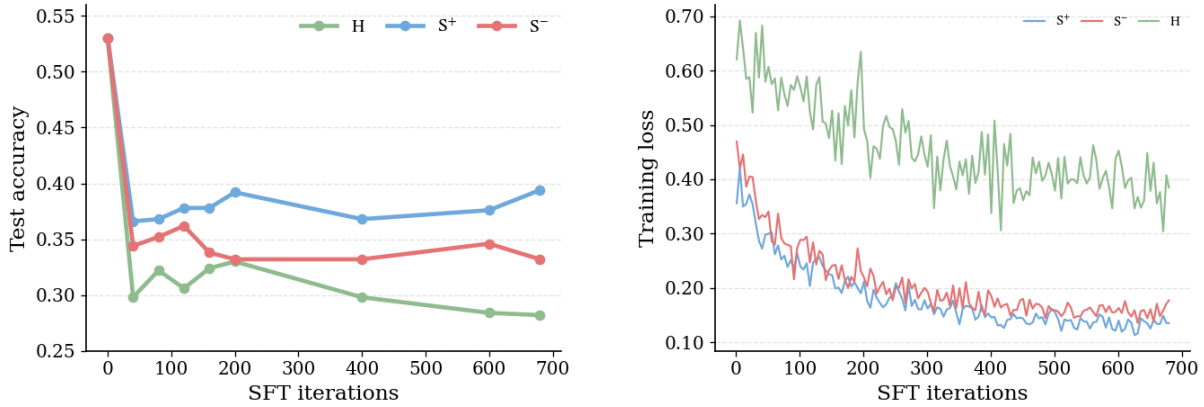


Figure 8: Performance of Qwen-2.5-1.5B on Math500.

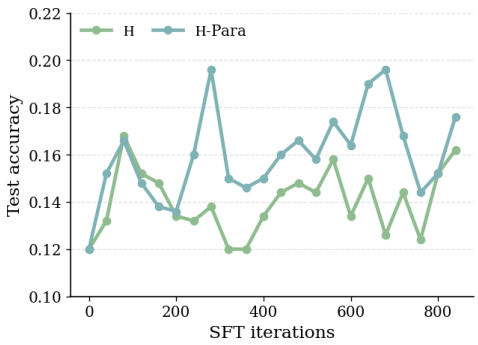


Figure 9: Paraphrasing experiment for G-2B. H paraphrased CoTs by G-27B-IT model performs better than the original H CoTs.

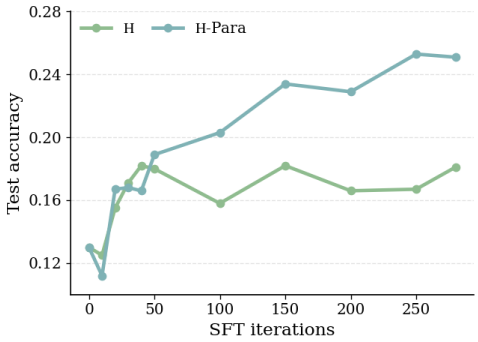
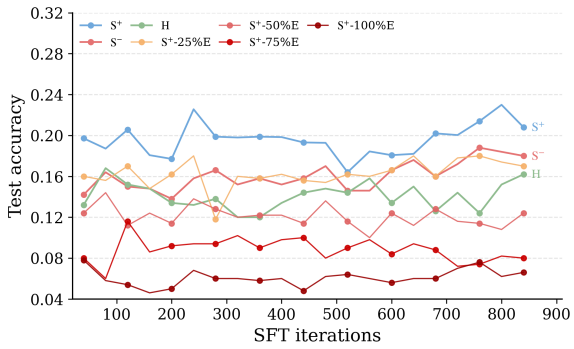
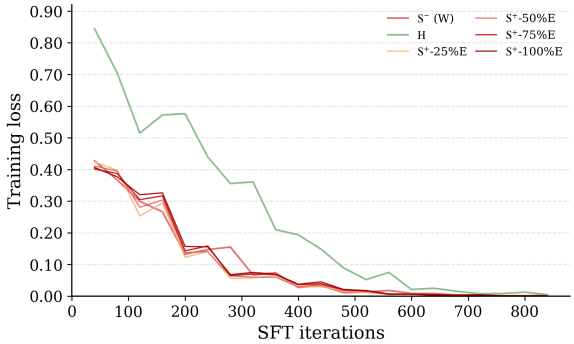


Figure 10: Paraphrasing experiment for L-8B. H paraphrased CoTs by L-70B-IT model performs better than the original H CoTs.

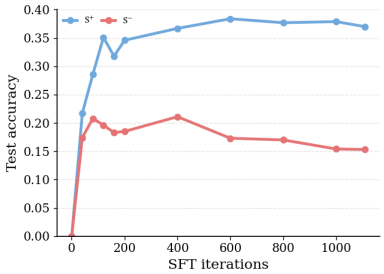


(a) Accuracy plots as we include the fully incorrect CoTs progressively.

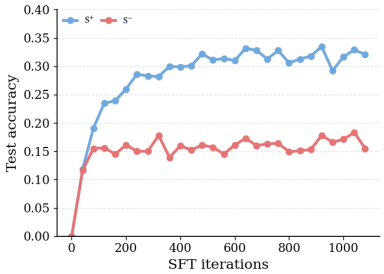


(b) Loss curves for human written CoTs, CoTs leading to incorrect answers, and CoTs with progressively fully incorrect traces.

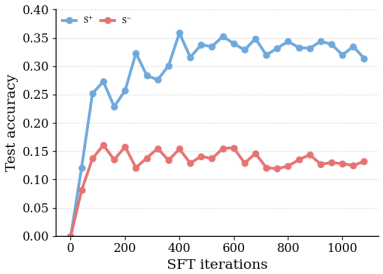
Figure 11: Comparison of accuracies on various datasets introduced with erraneous CoTs: (a) Test accuracies across the iterations. (b) Training loss curves. Performance degrades as we degrade CoTs. H distribution is further from that of model's as compared to that of fully flawed CoTs.



(a) Results of Llama-3.1-8B model on Countdown task after SFT on Gemma-2-27B-It generated CoTs datasets leading to all correct and all incorrect answers

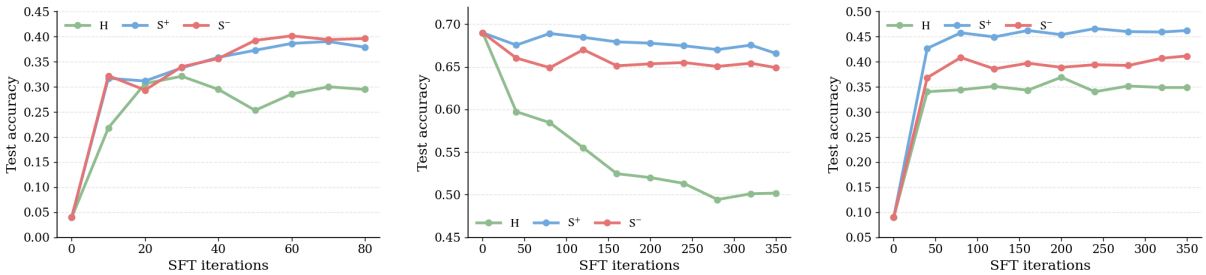


(b) Results of Qwen-2.5-1.5B model on Countdown task after SFT on Gemma-2-27B-It generated CoTs datasets leading to all correct and all incorrect answers



(c) Results of Gemma-2-2B model on Countdown task after SFT on Gemma-2-27B-It generated CoTs datasets leading to all correct and all incorrect answers

Figure 12: Results on Countdown across the training iterations

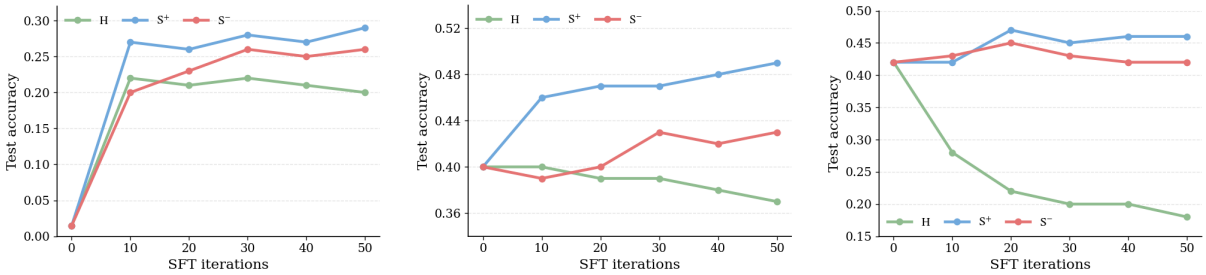


(a) Results of Llama-3.1-8B model on GSM8K task after SFT on Gemma-2-27B-It generated CoTs datasets leading to all correct and all incorrect answers

(b) Results of Qwen-2.5-1.5B model on GSM8K task after SFT on Gemma-2-27B-It generated CoTs datasets leading to all correct and all incorrect answers

(c) Results of Gemma-2-2B model on GSM8K task after SFT on Gemma-2-27B-It generated CoTs datasets leading to all correct and all incorrect answers

Figure 13: Results on GSM8K across the training iterations

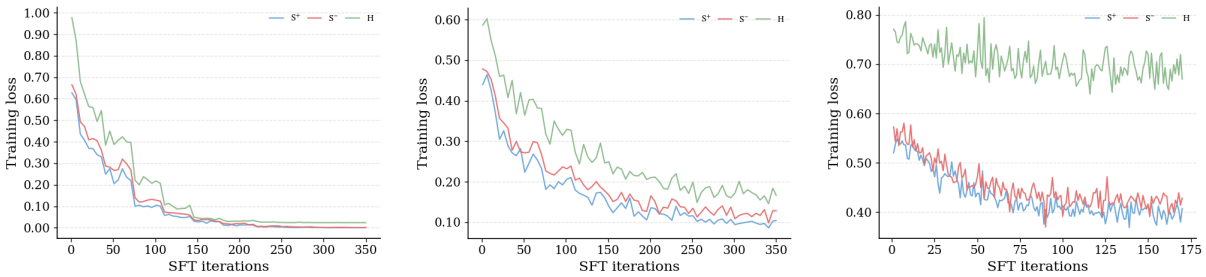


(a) Results of SFT on Gemma-2-2B model for Code Generation task.

(b) Results of SFT on Llama-3-8B model for Code Generation task.

(c) Results of SFT on Qwen-2.5-1.5B model for Code Generation task.

Figure 14: Performance on code generation on MBPP test data.

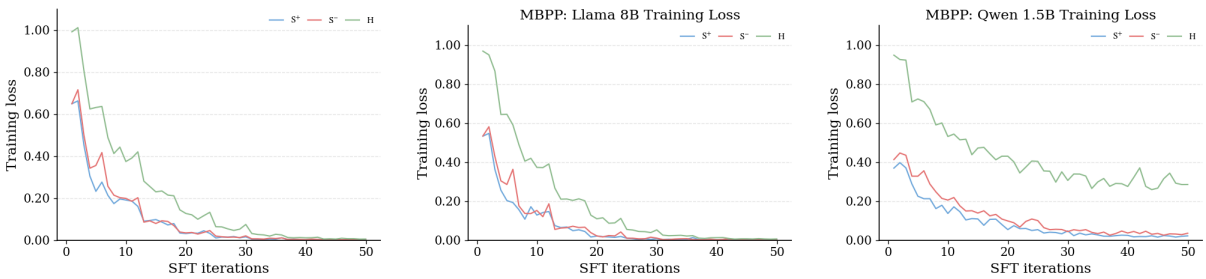


(a) Loss curves for Gemma-2-2B finetuning on GSM8K datasets.

(b) Loss curves for Qwen-2.5-1.5B finetuning on GSM8K datasets.

(c) Loss curves for Llama-3.1-8B-8B finetuning on GSM8K datasets.

Figure 15: Training losses: Finetuning on GSM8K datasets.

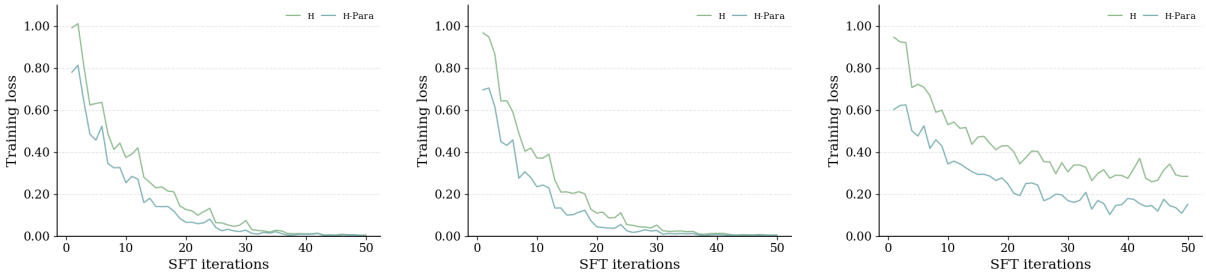


(a) Loss curves for Gemma-2-2B.

(b) Loss curves for Llama-3.1-8B.

(c) Loss curves for Qwen-2.5-1.5B.

Figure 16: Training losses from SFT runs on MBPP Code Generation datasets.



(a) Loss curves for Gemma-2-2B. (b) Loss curves for Llama-3.1-8B. (c) Loss curves for Qwen-2.5-1.5B.

Figure 17: Training losses from SFT runs on MBPP Code Generation datasets between H and H_{Para}.

A.7 CoTs Analysis

Example I: G-27B-IT-S^+

Problem. A student must choose a program of four courses from a list of courses consisting of English, Algebra, Geometry, History, Art, and Latin. This program must contain English and at least one mathematics course. In how many ways can this program be chosen?

Generated Solution by G-27B-IT leading to Correct Final Answer.

✓ **Step 1.** English, Algebra, and Geometry. Pick the fourth from {History, Art, Latin} — **3 ways**.

✗ **Step 2.** English, Algebra, and another math course. The student must choose one more course from the remaining three (History, Art, Latin). There are **3 ways** to do this. *(incorrect: duplicate counts)*

✗ **Step 3.** English, Geometry, and another math course. The student must choose one more course from the remaining three (History, Art, Latin). There are **3 ways** to do this. *(incorrect: duplicate counts)*

✓ **Step 4 (total).** $3 + 3 + 3 = \mathbf{9}$.

Incorrect case split with double-counting, but numerical total luckily matches the correct answer by coincidence.

Example I: G-27B-IT-S^-

Problem. A student must choose a program of four courses from a list of courses consisting of English, Algebra, Geometry, History, Art, and Latin. This program must contain English and at least one mathematics course. In how many ways can this program be chosen?

Generated Solution by G-27B-IT leading to Incorrect Final Answer.

✓ **Step 1.** English + Algebra + Geometry. Pick the fourth from {History, Art, Latin} — **3 ways**.

✗ **Step 2.** English + Algebra only. Should exclude Geometry; pick 2 from {History, Art, Latin} $\rightarrow \binom{3}{2} = 3$. Counted as **6** instead. *(incorrect count)*

✗ **Step 3.** English + Geometry only. Symmetric to Step 2; should be $\binom{3}{2} = 3$, but counted as **6**. *(incorrect count)*

✗ **Step 4 (total).** $3 + 6 + 6 = \boxed{15}$.

Incorrect total; correct total should be 9.

Example II: G-27B-IT-S^+

Problem. Determine the value of

$$2002 + \tfrac{1}{2}\big(2001 + \tfrac{1}{2}\big(2000 + \cdots + \tfrac{1}{2}\big(3 + \tfrac{1}{2} \cdot 2\big)\big) \cdots\big)\,.$$

Solution (student).

✓ **Step 1.** Define S.

$$S = 2002 + \frac{1}{2}\left(2001 + \frac{1}{2}\left(2000 + \cdots + \frac{1}{2}\left(3 + \frac{1}{2} \cdot 2\right)\right) \cdots\right).$$

✗ **Step 2.** Write $\frac{1}{2}S$.

$$\tfrac{1}{2}S = 1001 \;+\; \tfrac{1}{2}\left(2000 + \frac{1}{2}\left(1999 + \cdots + \frac{1}{2}\left(3 + \frac{1}{2} \cdot 2\right)\right) \cdots\right)$$

(mis-scaled tail; halving S should multiply the entire *nested part by an extra $\frac{1}{2}$, not shift to start at 2000)*

✗ **Step 3.** Subtract the two equations.

$$\tfrac{1}{2}S = 1001 + \tfrac{1}{2} \cdot 2001 = 2001$$

(arithmetic error)

✓ **Step 4.** Conclude S.

$$S = \boxed{4002}$$

Value happens to be correct, but the calculation above is invalid.

Example II: G-27B-IT-S^-

Problem. Determine the value of

$$2002 + \tfrac{1}{2}\big(2001 + \tfrac{1}{2}\big(2000 + \cdots + \tfrac{1}{2}\big(3 + \tfrac{1}{2} \cdot 2\big)\big) \cdots\big)\,.$$

Solution (student).

✓ **Step 1.** Define S.

$$S = 2002 + \frac{1}{2}\left(2001 + \frac{1}{2}\left(2000 + \cdots + \frac{1}{2}\left(3 + \frac{1}{2} \cdot 2\right)\right) \cdots\right).$$

✗ **Step 2.** Write $\frac{1}{2}S$.

$$\tfrac{1}{2}S = 1001 \;+\; \tfrac{1}{2}\left(2000 + \frac{1}{2}\left(1999 + \cdots + \frac{1}{2}\left(3 + \frac{1}{2} \cdot 2\right)\right) \cdots\right)$$

(the outer factor should be $\frac{1}{4}$ on the part beginning with 2001)

✗ **Step 3.** Subtract the two equations.

$$\tfrac{1}{2}S = 1001 + \tfrac{1}{2} \cdot 2001 \;=\; 1001 + 1000.5 = 2001.5$$

(subtraction invalid)

✗ **Step 4.** Conclude S.

$$S = \boxed{4003}$$

Incorrect final value caused by the faulty subtraction.

Example III: G-27B-IT-S^+

Problem. A student must choose a program of four courses from a list of courses consisting of English, Algebra, Geometry, History, Art, and Latin. This program must contain English and at least one mathematics course. In how many ways can this program be chosen?

Generated Solution by G-27B-IT leading to Correct Final Answer.

✓ **Step 1.** English, Algebra, and Geometry. Pick the fourth from {History, Art, Latin} — **3 ways**.

✗ **Step 2.** English, Algebra, and another math course. Since the only other math is Geometry, this duplicates Step 1. Counted again as **3 ways**.

✗ **Step 3.** English, Geometry, and another math course. Since the only other math is Algebra, this duplicates Step 1. Counted again as **3 ways**.

✓ **Step 4 (total).** $3 + 3 + 3 = \mathbf{9}$.

Incorrect case split with double-counting, but numerical total matches the correct answer by coincidence.

Example III: G-27B-IT-S^-

Problem. A student must choose a program of four courses from a list of courses consisting of English, Algebra, Geometry, History, Art, and Latin. This program must contain English and at least one mathematics course. In how many ways can this program be chosen?

Generated Solution by G-27B-IT leading to Incorrect Final Answer.

✓ **Step 1.** English + Algebra + Geometry. Pick the fourth from {History, Art, Latin} — **3 ways**.

✗ **Step 2.** English + Algebra only. Should exclude Geometry; pick 2 from {History, Art, Latin} $\rightarrow \binom{3}{2} = 6$.

✗ **Step 3.** English + Geometry only $\rightarrow \binom{3}{2} = 6$.

✗ **Step 4 (total).** $3 + 6 + 6 = \boxed{15}$.

Incorrect total; correct total should be 9.

Example I: Q-14B-IT-S^+

Problem. Let $f(x) = \left\lceil \frac{1}{x+2} \right\rceil$ for $x > -2$, and $f(x) = \left\lfloor \frac{1}{x+2} \right\rfloor$ for $x < -2$ (undefined at $x = -2$). Which integer is *not* in the range of $f(x)$?

Solution (model).

✓ **Step 1.** $x > -2 \Rightarrow \frac{1}{x+2} > 0 \Rightarrow$ range $\{1, 2, 3, \dots\}$.

✗ **Step 2.** $x < -2 \Rightarrow \frac{1}{x+2} <= 0 \Rightarrow$ floor gives $\{0, -1, -2, -3, \dots\}$.

✗ **Step 3.** No branch yields 0; thus 0 is not in the range. *(incorrect reasoning, still comes up with right answer)*

✓ **Final.** $\boxed{0}$.

Example I: Q-14B-IT-S^-

Problem. Let $f(x) = \left\lceil \frac{1}{x+2} \right\rceil$ for $x > -2$, and $f(x) = \left\lfloor \frac{1}{x+2} \right\rfloor$ for $x < -2$ (undefined at $x = -2$). Which integer is *not* in the range of $f(x)$?

Solution (model).

✓ **Step 1.** For $x > -2$, $\frac{1}{x+2} > 0$ and $\lceil \cdot \rceil$ yields $1, 2, 3, \dots$

✗ **Step 2.** For $x < -2$, claim $\left\lfloor \frac{1}{x+2} \right\rfloor$ includes 0 and negatives. Here $\frac{1}{x+2} < 0$, so $\lfloor \cdot \rfloor \leq -1$; 0 is not attained.

✗ **Final.** Missing integer $= -1$.

Incorrect; correct missing integer is 0.

Example II: Q-14B-IT-S^+

Problem. How many integers n satisfy the condition $100 < n < 200$ and the condition n has the same remainder whether it is divided by 6 or by 8?

Solution (model).

✓ **Step 1.** Write $n = 6k + r = 8m + r$ with $0 \leq r < 6$.

✓ **Step 2.** Equating gives $6k + r = 8m + r \implies 3k = 4m$.

✓ **Step 3.** Let $m = 3p \implies n = 24p + r$.

✓ **Step 4.** Apply bounds: $100 < 24p + r < 200$.

✓ **Step 5.** Check $r = 0, 1, 2, 3, 4$: each gives 4 values of p. For $r = 5$: 5 values.

✓ **Step 6 (total).** $4 + 4 + 4 + 4 + 4 + 5 = 25$.

✓ **Final Answer:** $\boxed{25}$.

Example II: Q-14B-IT-S^-

Problem. How many integers n satisfy the condition $100 < n < 200$ and the condition n has the same remainder whether it is divided by 6 or by 8?

Solution (model).

✓ **Step 1.** Write $n \equiv r \pmod 6$ and $n \equiv r \pmod 8$ with $0 \leq r < 6$.

✓ **Step 2.** Then $n - r$ divisible by $\text{LCM}(6, 8) = 24 \implies n = 24k + r$.

✓ **Step 3.** Apply bounds: $100 < 24k + r < 200$.

✗ **Step 4.** Check each r: counted 4 values for each case. For $r = 5$, missed one extra value (should be 5).

✗ **Step 5 (total).** $4 + 4 + 4 + 4 + 4 + 4 = 24$.

✗ **Final Answer:** $\boxed{24}$.

Incorrect; correct total is 25.

Countdown G-27B-IT-S^+ — Target 57 from [50, 42, 63, 86]

Problem. Using the numbers [50, 42, 63, 86], create an equation that equals 57. Use only $+, -, \times, \div$, and use each number once.

Solution (model).

✓ **Step 1.** Try a difference with the largest: $86 - 50 = 36$.

✓ **Step 2.** Combine with 63: $63 + 36 = 99$.

✓ **Step 3.** Subtract the remaining number: $99 - 42 = 57$.

✓ **Final.** Expression uses all four once: $(86 - 50) + 63 - 42 = 57$.
`<answer> (86 - 50) + 63 - 42 </answer>`

Countdown G-27B-IT-S^- — Target 57 from [50, 42, 63, 86]

Problem. Using the numbers [50, 42, 63, 86], create an equation that equals 57. Use only $+, -, \times, \div$, and use each number once.

Solution (model).

✓ **Step 1.** Consider $86 - 42 = 44$ as a starting point. *(arithmetically correct)*

✗ **Step 2.** Claim "$63 - 6 = 57$" without legitimately forming 6 from the given numbers using allowed operations exactly once.

✗ **Step 3.** Proposed final: $86 - 42 + 50 - 63 = 31 \neq 57$.

✗ **Final.** Does not reach 57; arithmetic and constraints violated.
`<answer> 86 - 42 + 50 - 63 </answer>` (evaluates to 31)

GSM8K G-27B-IT-S^+

Problem. Chelsea has 24 kilos of sugar. She divides them into 4 bags equally. Then one of the bags gets torn and half of the sugar falls to the ground. How many kilos of sugar remain?

Solution (model).

✓ **Step 1.** Divide evenly: $24 \div 4 = 6$ kilos per bag.

✓ **Step 2.** Sugar lost from torn bag: $6 \div 2 = 3$ kilos.

✓ **Step 3.** Torn bag now has $6 - 3 = 3$ kilos left.

✓ **Step 4.** Total remaining: $torn\ 3 + 3$ bags $\times\ 6 = 3 + 18 = 21$.

✓ **Final.** The final answer is 21 kilos.

GSM8K G-27B-IT-S^-

Problem. Chelsea has 24 kilos of sugar. She divides them into 4 bags equally. Then one of the bags gets torn and half of the sugar falls to the ground. How many kilos of sugar remain?

Solution (model).

✓ **Step 1.** Divide evenly: $24 \div 4 = 6$ kilos per bag.

✓ **Step 2.** Lost from torn bag: $6 \div 2 = 3$ kilos.

✓ **Step 3.** Torn bag left with $6 - 3 = 3$ kilos.

✗ **Step 4.** Total remaining computed as $3 \times 3 + 3 = 12$. *(the three untouched bags still have 6 kilos each, not 3; correct total is $3 + 3 \times 6 = 21$)*

✗ **Final.** The final answer is 12 kilos.

A.8 Beyond Final Answer Correctness: Understanding the Structure of CoTs

To understand how models can learn from CoTs with incorrect final answers, we must look beyond the final answer and examine the reasoning steps themselves. Let x denote an input problem (question) and let y denote a generated chain-of-thought (CoT) for x. We represent y as an ordered sequence of T reasoning steps,

$$y = (s_1, s_2, \ldots, s_T),$$

where s_i is the i-th textual step and $T = T(y)$ is the number of steps in the trace.[1] For a dataset D (e.g., $D \in \{S^+, S^-\}$), we write $(x, y) \sim D$ to mean that y is a CoT trace paired with problem x drawn from D. We use T_D to denote the (random) trace length under D, i.e., $T_D := T(y)$ when $(x, y) \sim D$, and $\mathbb{E}[T_D]$ for its expected value.

We call a step s_i locally correct if it is consistent with at least one valid solution trajectory for the given problem x and the preceding prefix $s_{<i} := (s_1, \ldots, s_{i-1})$ (e.g., a correct case split or algebraic manipulation), and locally incorrect otherwise (e.g., double counting, invalid inference, or arithmetic slip). Crucially, a trace with an incorrect final answer is not necessarily random noise: it may contain many locally correct, reusable primitives; conversely, a trace that ends in the correct final answer can still include flawed reasoning steps.

Step-level mixture of locally correct and incorrect reasoning steps. The key point is that a single trace can contain both locally correct and locally incorrect steps. Thus, rather than modeling traces as entirely "correct" or entirely "incorrect," we formalize a step-level mixture. Let $c_i \in \{0, 1\}$ denote the local correctness of step s_i, where $c_i = 1$ indicates that s_i is locally correct and $c_i = 0$ indicates a localized error. Let $c_{1:T} = (c_1, \ldots, c_T)$ be the latent correctness sequence. We define a step-level mixture model:

$$p_{\text{mix}}(y \mid x) = \sum_{c_{1:T}} p(c_{1:T} \mid x) \prod_{i=1}^{T} p(s_i \mid x, s_{<i}, c_i), \tag{3}$$

$$p(s_i \mid x, s_{<i}, c_i = 1) = p_c(s_i \mid x, s_{<i}), \tag{4}$$

$$p(s_i \mid x, s_{<i}, c_i = 0) = p_e(s_i \mid x, s_{<i}), \tag{5}$$

where $p_c(\cdot \mid x, s_{<i})$ and $p_e(\cdot \mid x, s_{<i})$ are, respectively, the distributions over locally correct and erroneous next-step continuations given the prefix. This notation avoids conflating trace-level correctness with step-level correctness: a single trace y is generated by interleaving draws from p_c and p_e across steps, governed by $c_{1:T}$.

For simplicity, we can assume a parameterization where step correctness is independent across positions: $c_i \overset{\text{iid}}{\sim} \text{Bernoulli}(\lambda)$.[2] Marginalizing c_i yields a per-step mixture:

$$p_{\text{mix}}(y \mid x) = \prod_{i=1}^{T} \Big[\lambda\, p_c(s_i \mid x, s_{<i}) + (1 - \lambda)\, p_e(s_i \mid x, s_{<i})\Big], \tag{6}$$

where $\lambda \in [0, 1]$ is the expected fraction of locally correct steps. For a dataset D (e.g., S^+ or S^-), we summarize this notion via

$$\lambda_D \;:=\; \mathbb{E}_{(x,y)\sim D}\left[\frac{1}{T(y)} \sum_{i=1}^{T(y)} c_i\right], \tag{7}$$

i.e., the expected fraction of locally correct steps in traces drawn from D. This formulation captures the intended intuition: both S^+ (final-answer-correct) and S^- (final-answer-wrong) traces can have substantial λ_D because many steps are correct even when the final answer is wrong, and S^+ can still have $\lambda_D < 1$ due to hidden localized errors.

A CoT that leads to finally incorrect answer (S^-) is not random noise. Its reasoning trace is a sequence of primitives where some are incorrect, but many others may still be correct. In both types of datasets (S^+ and S^-), there could be plenty of correct, reusable reasoning steps. These provide a useful learning signal that allows the model to learn general reasoning patterns despite the noise from localized errors.

[1] All steps are defined by the delimiter used in our prompting/template; see Appendix A.7 for examples.

[2] More generally, $p(c_{1:T} \mid x)$ can be non-iid (e.g., Markov) to capture error cascades; we use the iid form as a minimal abstraction.

Implication for SFT on S^+ and S^-. Under the step-mixture view in Eqns. 3–7, both S^+ and S^- datasets can have non-zero and moderate λ_D, meaning they include many locally correct steps that provide a useful learning signal. This explains why SFT on either S^+ or S^- can improve performance: the model can learn generalizable reasoning patterns from reusable correct primitives, even in the presence of noise from localized errors. Conversely, in our controlled corruption setting, injecting increasingly flawed sub-steps effectively reduces λ_D by increasing the prevalence of locally incorrect steps, which aligns with the observed performance degradation as the fraction of flawed traces grows.

A.9 Analyzing the Structure of the CoTs

A natural concern is that the performance differences between human-written solutions (H), model-generated correct solutions (S^+), and model-generated incorrect solutions (S^-) may be explained by simple surface-level properties of the text, rather than by distributional proximity to the model. For example, synthetic CoTs might be longer, might introduce a substantially different vocabulary, or might contain more explicit reasoning scaffolding such as verification, backtracking, or step decomposition. We therefore analyze three possible confounds: sequence length, response-token vocabulary overlap, and the prevalence of explicit cognitive-behavior markers.

Dataset-level token and word statistics. We first examine whether the datasets differ substantially in length. Longer CoTs could provide more supervision signal per example, making length a potential alternative explanation for the observed performance differences. Table 4 reports mean token counts per example, computed with the `google/gemma-2-2b` tokenizer over the full SFT-formatted example, i.e., prompt plus response.

Within each domain, the generated correct and incorrect datasets have very similar lengths. For GSM8K, the mean token counts are 285.56 for S^+ and 289.19 for S^-, compared to 256.16 for H. For MATH, the corresponding means are 399.89 for S^+ and 401.90 for S^-, compared to 360.23 for H. Thus, the generated datasets are moderately longer than the human-written datasets, but S^+ and S^- are nearly matched in length. Since S^+ and S^- differ in correctness while having almost identical average lengths, sequence length alone cannot explain the performance differences between them. More broadly, the modest length differences suggest that the content and distributional organization of the tokens are more relevant than raw sequence length.

Dataset	Examples	Mean Tokens
GSM8K S^+	2,289	285.56
GSM8K S^-	2,289	289.19
GSM8K H	2,289	256.16
MATH S^+	4,408	399.89
MATH S^-	4,408	401.90
MATH H	4,408	360.23

Table 4: Mean token counts per example across datasets. Token counts are computed with the `google/gemma-2-2b` tokenizer over the full SFT-formatted example (prompt plus response).

Response-token vocabulary overlap. Next, we ask whether the synthetic datasets differ from H because they introduce many novel token types. We define the response-token vocabulary of a dataset as the set of unique tokenizer IDs that appear at least once in its responses. This is a type-level statistic: a token that appears many times still contributes only one item to the vocabulary. Since the prompts are identical across H, S^+, and S^- within each domain, we restrict this analysis to response tokens.

We measure overlap using Jaccard similarity. For two token vocabularies A and B, Jaccard similarity is defined as

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|},$$

where $|A \cap B|$ is the number of shared token types and $|A \cup B|$ is the number of token types appearing in either vocabulary. A value of 1 indicates identical vocabularies, while a value of 0 indicates no shared

token types. We also report the number of token types shared by all three datasets and the number of token types unique to each dataset. Finally, we report unique token mass, defined as the percentage of all response-token occurrences in a dataset that belong to token types appearing only in that dataset.

Table 5 shows that H, S^+, and S^- share a substantial common response vocabulary. In GSM8K, 4,581 token types appear in all three datasets; in MATH, 4,420 token types appear in all three. The two generated datasets are especially close: the S^+–S^- Jaccard similarity is 0.663 for GSM8K and 0.678 for MATH, higher than the overlaps between H and either generated dataset. This is expected because S^+ and S^- are both produced by the same model family and therefore share similar surface style.

Importantly, even when a dataset contains unique token types, those tokens account for only a small fraction of the actual token occurrences. In MATH, unique token mass is only 0.68% for H, 0.13% for S^+, and 0.15% for S^-. In GSM8K, unique mass is 5.88% for H, 0.50% for S^+, and 0.56% for S^-. Thus, the synthetic datasets do not outperform H simply by introducing a large disjoint vocabulary. Instead, all three datasets largely operate over a shared lexical base, and the relevant differences must lie in how these shared tokens are arranged across solutions.

Domain	H Vocab	S^+ Vocab	S^- Vocab	Shared	H Unique	S^+ Unique	S^- Unique	H–S^+ Jac.	H–S^- Jac.	S^+–S^- Jac.	Unique Mass (%)
GSM8K	6,446	8,406	8,526	4,581	1,049	1,240	1,374	0.507	0.499	0.663	5.88 / 0.50 / 0.56
MATH	7,147	6,930	7,016	4,420	1,757	806	896	0.535	0.529	0.678	0.68 / 0.13 / 0.15

Table 5: Response-token vocabulary overlap across H, S^+, and S^-. "Vocab" denotes the number of unique response-token types in each dataset. "Shared" denotes token types appearing in all three datasets. "Unique" denotes token types appearing only in that dataset. Jaccard similarity is $J(A, B) = |A \cap B|/|A \cup B|$. "Unique Mass" reports the percentage of response-token occurrences accounted for by token types unique to H, S^+, and S^-, respectively.

Prevalence of cognitive-behavior markers. Finally, we test whether synthetic CoTs differ from human-written CoTs because they contain more explicit reasoning scaffolding. Prior work on self-improving reasoners studies cognitive behaviors such as verification, backtracking, subgoal setting, and backward chaining. We therefore measure how often explicit lexical markers of these behaviors appear in our responses.

We operationalize this as a conservative lexical marker analysis. For each response, we search for manually specified regular-expression patterns corresponding to four categories. Verification markers include phrases such as "check", "verify", "confirm", and "substitute back". Backtracking markers include phrases such as "instead", "mistake", "wrong", and "try another approach". Subgoal-setting markers include phrases such as "Step 1", "first", "next", "then", and "we need to find". Backward chaining markers include phrases such as "work backwards", "from the target", and "to reach the goal" (Gandhi et al., 2025). This procedure only counts explicit surface markers and should therefore be interpreted as an auditable lexical proxy, not as a classifier-based annotation of latent reasoning behavior.

We report two quantities. "Ex." is the percentage of examples containing at least one marker from a category. "Tok." is the percentage of all response tokens covered by matched marker spans, where marker spans are tokenized using the `google/gemma-2-2b` tokenizer. The token-level metric is especially important: a response may contain a phrase such as "Step 1", but that phrase may occupy only a few tokens in the full solution. Thus, token-level prevalence measures how much of the dataset is actually composed of explicit cognitive-behavior language.

Table 6 shows that such markers occupy a very small fraction of response-token mass. In GSM8K, the total marker-token fraction is 0.21% for H, 1.31% for S^+, and 1.19% for S^-. In MATH, the values are even smaller: 0.13% for H, 0.19% for S^+, and 0.20% for S^-. The largest component is subgoal-setting language in the generated GSM8K datasets, mostly reflecting shallow step-structuring phrases such as "Step 1", "first", and "then". Verification, backtracking, and backward-chaining markers are extremely sparse across both domains and all dataset variants, as shown in Table 7.

These results rule out another surface-level explanation: the advantage of synthetic CoTs is not driven by the presence of extensive explicit cognitive scaffolding. Together with the length and vocabulary analyses, this suggests that the performance differences between H, S^+, and S^- are better explained by distributional alignment and the organization of reasoning tokens than by simple surface statistics.

Domain	Dataset	Response Tokens	Any Ex. (%)	Any Tok. (%)	Subgoal Tok. (%)
GSM8K	H	320,784	8.04	0.21	0.20
GSM8K	S^+	386,486	19.09	1.31	1.29
GSM8K	S^-	394,996	18.79	1.19	1.16
MATH	H	914,173	11.93	0.13	0.05
MATH	S^+	1,084,571	13.14	0.19	0.11
MATH	S^-	1,093,450	13.70	0.20	0.11

Table 6: Aggregate prevalence of lexical cognitive-behavior markers in response text. "Any Ex." is the percentage of examples containing at least one marker from any category. "Any Tok." is the percentage of response tokens covered by all matched marker spans. "Subgoal Tok." reports the token-level prevalence of subgoal-setting markers, the largest marker category in our data.

Domain	Dataset	Any Ex.	Any Tok.	Verif. Ex.	Verif. Tok.	Backtrack Ex.	Backtrack Tok.	Subgoal Ex.	Subgoal Tok.	Backward Ex.	Backward Tok.
GSM8K	H	8.04	0.208	0.22	0.003	0.22	0.002	7.47	0.196	0.22	0.007
GSM8K	S^+	19.09	1.315	0.17	0.001	0.31	0.004	18.00	1.292	0.96	0.018
GSM8K	S^-	18.79	1.193	0.57	0.006	1.05	0.013	17.04	1.161	0.57	0.012
MATH	H	11.93	0.125	6.83	0.060	1.34	0.008	4.36	0.055	0.14	0.002
MATH	S^+	13.14	0.186	8.39	0.074	0.59	0.004	5.44	0.105	0.29	0.003
MATH	S^-	13.70	0.201	8.76	0.076	1.04	0.014	5.15	0.107	0.32	0.003

Table 7: Example-level and token-level prevalence of lexical cognitive-behavior markers. "Ex." denotes the percentage of examples containing at least one marker from the corresponding category. "Tok." denotes the percentage of response tokens covered by matched marker spans. "Verif." denotes verification, "Backtrack" denotes backtracking, "Subgoal" denotes subgoal setting, and "Backward" denotes backward chaining. All token percentages are computed with the `google/gemma-2-2b` tokenizer.

A.10 KL-Divergence as a metric for distributional analysis

To quantify how close each training dataset is to the base model's native output distribution, we estimate an empirical token-level KL divergence between dataset responses and base-model generations. For each base model M, we generate one response $\tilde{y}_i \sim p_M(\cdot \mid x_i)$ for every prompt x_i in the corresponding dataset. Let $D \in \{H, S^+, S^-\}$ denote a dataset of responses y_i^D. We tokenize both the dataset responses and the base-model generations using the tokenizer of M.

Let $\mathcal{V}_{D,M}$ be the union of tokens observed in either the responses from dataset D or the base-model generations. For each token $t \in \mathcal{V}_{D,M}$, let $c_D(t)$ be the number of times token t appears in all responses from dataset D, and let $c_M(t)$ be the number of times token t appears in the base-model generations. We estimate smoothed empirical token distributions as

$$\hat{q}_D(t) = \frac{c_D(t) + \alpha}{\sum_{t' \in \mathcal{V}_{D,M}} c_D(t') + \alpha|\mathcal{V}_{D,M}|} \quad \text{and} \quad \hat{p}_M(t) = \frac{c_M(t) + \alpha}{\sum_{t' \in \mathcal{V}_{D,M}} c_M(t') + \alpha|\mathcal{V}_{D,M}|},$$

where $\alpha = 10^{-8}$ is an additive smoothing constant.

We then compute the forward KL divergence

$$D_{\mathrm{KL}}(\hat{q}_D \| \hat{p}_M) = \sum_{t \in \mathcal{V}_{D,M}} \hat{q}_D(t) \log \frac{\hat{q}_D(t)}{\hat{p}_M(t)}.$$

Lower values indicate that the dataset response distribution is closer to the base model's generated response distribution. This metric is an empirical corpus-level approximation rather than an exact sequence-level KL between conditional language-model distributions. Exact teacher-to-base KL would require access to the teacher model's conditional probabilities, which are unavailable for our datasets and undefined for human-written responses. Our estimator instead treats each dataset as samples from an underlying response distribution and compares its aggregate token usage to samples from the base model on the same prompt set.

A.11 Prompts

MATH PROBLEM SOLVING (ZERO-SHOT) PROMPT

Zero-Shot Prompt Template :

You are a math expert. I am going to give you a math Problem that you need to solve. When you answer, respond only with the Solution, thinking step by step. At the end of the Solution, when you give your final answer, write it in the form "Final Answer: The final answer is {answer}. I hope it is correct."

Problem: {problem}

Solution:

MATH PROBLEM SOLVING (FOUR-SHOT) PROMPT

You are a math expert. I am going to give you a math Problem that you need to solve. When you respond, respond only with the Solution, thinking step by step. At the end of the Solution, when you give your final answer, write it in the form "Final Answer: The final answer is {answer}. I hope it is correct."

Problem: Find the domain of the expression $\frac{\sqrt{x-2}}{\sqrt{5-x}}$.
Solution: The expressions inside each square root must be non-negative. Therefore, $x-2 \geq 0$, so $x \geq 2$, and $5-x \geq 0$, so $x \leq 5$. Also, the denominator cannot be equal to zero, so $5-x > 0$, which gives $x < 5$. Therefore, the domain of the expression is $\boxed{[2,5)}$. Final Answer: The final answer is $[2,5)$. I hope it is correct.

Problem: If $\det \mathbf{A} = 2$ and $\det \mathbf{B} = 12$, then find $\det(\mathbf{AB})$.
Solution: We have that $\det(\mathbf{AB}) = (\det \mathbf{A})(\det \mathbf{B}) = (2)(12) = \boxed{24}$. Final Answer: The final answer is 24. I hope it is correct.

Problem: Terrell usually lifts two 20-pound weights 12 times. If he uses two 15-pound weights instead, how many times must Terrell lift them in order to lift the same total weight?
Solution: If Terrell lifts two 20-pound weights 12 times, he lifts a total of $2 \cdot 12 \cdot 20 = 480$ pounds of weight. If he lifts two 15-pound weights instead for n times, he will lift a total of $2 \cdot 15 \cdot n = 30n$ pounds of weight. Equating this to 480 pounds, we can solve for n:

$$\begin{aligned} 30n &= 480 \\ \Rightarrow \qquad n &= 480/30 = \boxed{16} \end{aligned}$$

Final Answer: The final answer is 16. I hope it is correct.

Problem: If the system of equations

$$\begin{aligned} 6x - 4y &= a, \\ 6y - 9x &= b. \end{aligned}$$

has a solution (x, y) where x and y are both nonzero, find $\frac{a}{b}$, assuming b is nonzero.
Solution: If we multiply the first equation by $-\frac{3}{2}$, we obtain $6y - 9x = -\frac{3}{2}a$.Since we also know that $6y - 9x = b$, we have $-\frac{3}{2}a = b \Rightarrow \frac{a}{b} = \boxed{-\frac{2}{3}}$. Final Answer: The final answer is $-\frac{2}{3}$. I hope it is correct.

Problem:
{problem}

Solution:

992

GSM8K PROBLEM SOLVING (ZERO-SHOT) PROMPT

You are a math expert. I am going to give you a math Problem. Think step by step and you generate the solution. Write the final answer in the form "Final Answer: The final answer is #### answer."

Problem:
{problem}

Solution:

GSM8K PROBLEM SOLVING (FOUR-SHOT) PROMPT

You are a math expert. I am going to give you a math Problem. Think step by step and you generate the solution. Write the in the final answer in the form "Final Answer: The final answer is #### answer."

Problem: Janet's ducks lay 16 eggs per day. She eats three for breakfast every morning and bakes muffins for her friends every day with four. She sells the remainder at the farmers' market daily for $2 per fresh duck egg. How much in dollars does she make every day at the farmers' market?
Solution: Janet sells 16 - 3 - 4 = «16-3-4=9»9 duck eggs a day.
She makes 9 * 2 = $«9*2=18»18 every day at the farmer's market.
Final Answer: The final answer is #### 18.

Problem: A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?
Solution: It takes 2/2=«2/2=1»1 bolt of white fiber.
So the total amount of fabric is 2+1=«2+1=3»3 bolts of fabric.
Final Answer: The final answer is #### 3.

Problem: Josh decides to try flipping a house. He buys a house for $80,000 and then puts in $50,000 in repairs. This increased the value of the house by 150%. How much profit did he make?
Solution: The cost of the house and repairs came out to 80,000+50,000=$«80000+50000=130000»130,000.
He increased the value of the house by 80,000*1.5=«80000*1.5=120000»120,000.
So the new value of the house is 120,000+80,000=$«120000+80000=200000»200,000.
So he made a profit of 200,000-130,000=$«200000-130000=70000»70,000.
Final Answer: The final answer is #### 70000.

Problem: James decides to run 3 sprints 3 times a week. He runs 60 meters each sprint. How many total meters does he run a week?
Solution: He sprints 3*3=«3*3=9»9 times.
So he runs 9*60=«9*60=540»540 meters.
Final Answer: The final answer is #### 540.

Problem:
{problem}

Solution:

COUNTDOWN PROBLEM SOLVING (ZERO-SHOT) PROMPT

You are a math expert. I am going to give you a Problem that you need to solve. When you respond, respond with the Solution, thinking step by step. And return the final answer in `<answer> </answer>` tags, for example `<answer> (1 + 2) / 3 </answer>`.

Problem:
Using the numbers {nums}, create an equation that equals {target}. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Do not use any other operations or numbers.

Solution:

COUNTDOWN PROBLEM SOLVING (FOUR-SHOT) PROMPT

You are a math expert. I am going to give you a Problem that you need to solve. When you respond, respond with the Solution, thinking step by step. And return the final answer in `<answer> </answer>` tags, for example `<answer> (1 + 2) / 3 </answer>`.

Problem: Using the numbers [38, 98, 56, 14], create an equation that equals 91. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Do not use any other operations or numbers.
Solution: I am looking for a combination of numbers and operations that results in 91. I can try to combine numbers using addition first. Let's try adding 38 and 14. 38 + 14 = 52. Now I have the numbers 52, 98, and 56 left to use. I need to get to 91. Let's see if multiplication or division can help. Let's try multiplying 52 by 98. 52 * 98 is a large number. Let's try dividing by 56. So, (52 * 98) / 56. I can simplify this calculation. 98 and 56 are both divisible by 14. 98 / 14 = 7. 56 / 14 = 4. So, the expression becomes 52 * (7 / 4). I can rewrite this as (52 / 4) * 7. 52 / 4 = 13. Now I just need to multiply 13 by 7. 13 * 7 = 91. This gives the target number. The full equation is ((38 + 14) * 98) / 56. `<answer> ((38 + 14) * 98) / 56 </answer>`

Problem: Using the numbers [23, 63, 79, 51], create an equation that equals 68. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Do not use any other operations or numbers.
Solution: I am looking for a combination of numbers and operations that results in 68. I'll start with the largest number, 79. To get to 68, I need to subtract 11. Can I make 11 from 23, 63, and 51? 63 - 51 = 12. This is close to 11. Let's try 79 - (63 - 51) = 79 - 12 = 67. This is very close to 68, but not exactly. Let's try another combination. 79 + 51 = 130. 63 + 23 = 86. 130 - 86 = 44. Let's try another path. 63 - 23 = 40. 79 - 51 = 28. 40 + 28 = 68. This works! I have found a solution. The steps are: subtract 23 from 63 to get 40. Subtract 51 from 79 to get 28. Add the results together. `<answer> (63 - 23) + (79 - 51) </answer>`

Problem: Using the numbers [16, 17, 58], create an equation that equals 91. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Do not use any other operations or numbers.
Solution: I need to reach the target of 91 using the numbers 16, 17, and 58. Since there are only three numbers, I'll try adding them up first. 58 + 17 = 75. Now, I need to incorporate the last number, 16. 75 + 16 = 91. This is the target number. So, the solution is to add all the numbers together. `<answer> 58 + 17 + 16 </antswer>`

Problem: Using the numbers [2, 28, 78], create an equation that equals 11. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Do not use any other operations or numbers.
Solution: The target is 11. The numbers are 2, 28, and 78. The numbers are quite spread out, so simple addition or subtraction of all of them at once is unlikely to work. Let's see if there is a division or multiplication that simplifies the problem. 78 is an even number, so it's divisible by 2. 78 / 2 = 39. Now I have the number 39, and the remaining number is 28. I need to reach the target of 11. Let's see the difference between 39 and 28. 39 - 28 = 11. This is the target number. So the equation is (78 / 2) - 28. `<answer> (78 / 2) - 28 </answer>`

Problem:
Using the numbers {nums}, create an equation that equals {target}. You can use basic arithmetic operations (+, -, *, /) and each number can only be used once. Do not use any other operations or numbers.

Solution:

Prompt for Math Paraphrasing

You are a math expert. I am going to give you a Problem and a correct solution (provided as a hint). Your task is to rewrite the solution in your own words and style, while making use of the hint as guidance.
Ensure that your reasoning follows the same logical steps as the hint, and that your final answer matches the solution's final result. You do not need to generate a new solution, just rewrite in your own style while adhering to the hint solution.

Problem:
problem

Correct Solution (provided as hint) to be Paraphrased:
response

Your Paraphrased Solution:

PROMPT FOR GENERATING COMPLETELY FLAWED REASONING

You have been given a math problem. Your task is to create a completely flawed mathematical solution to the problem below that results in an incorrect answer. Every step must use invented formulas and incorrect reasoning. However, ensure that the solution should not be completely random and the solution should be based on the given problem. Do NOT mention that you are generating an incorrect solution anywhere in the solution as this data will be used for error analysis research.
Final Answer Format: At the end of the Solution, when you give your final answer, write it in the form Final Answer: The final answer is $answer$. I hope it is correct.

Problem: {problem}

Completely Incorrect Solution:

A.12 Statistical Significance and Hyperparameter Ablations

To ensure the robustness of our results, we performed experiments on the Gemma 2B model on the Human H CoTs and Gemma-27B-IT generated S^+ and S^- CoTs, and we used two complementary measures: (i) we report mean $\pm$ standard deviation across 5 random seeds to characterize run-to-run variability, and (ii) we report 95% confidence intervals on the mean (t-based over 5 seeds) to quantify the uncertainty of the estimated average performance. Table 8 shows that our results are robust across the runs. We report results on the max-scores that we report in our tables. But the trends hold across all checkpoints across all 5 runs.

Table 8: Summary of robustness metrics over 5 random seeds. We report mean $\pm$ standard deviation and 95% confidence intervals on the mean. We run the experiments on Gemma 2B (G2B) over the datasets- H, G27B-IT-S^+, and G27B-IT-S^-.

Run	Mean $\pm$ Std Dev	95% CI on Mean
G2B on H	0.162 ± 0.005	0.162 ± 0.006
G2B on G27B-IT-S^-	0.183 ± 0.008	0.183 ± 0.009
G2B on G27B-IT-S^+	0.223 ± 0.005	0.223 ± 0.006

We also report a detailed ablation varying the batch size and the learning rate to study the results when the two main hyperparameters of our experiments change. We again performed experiments on Gemma 2B (G2B) with Human H CoTs and Gemma-27B-IT generated S^+ and S^- CoTs. We ablated the batch

Table 9: **Ablation of result trends across different hyperparameters.** We perform a detailed ablation of the main hyperparameters- batch size (BS) and learning rate (lr) over the Gemma-2B model on the MATH related datasets to show that the trends of results that synthetic S^+ and S^- CoTs outperform H CoTs remain consistent across all runs. We provide maximum accuracy on MATH500 test set for G-2B for all these runs.

Model	H	G-27B-IT-S^-	G-27B-IT-S^+
BS 256 lr 2e-5	0.17	+0.02	+0.04
BS 256 lr 1e-6	0.14	+0.03	+0.05
BS 64 lr 2e-5	0.17	+0.02	+0.06
BS 64 lr 1e-6	0.15	+0.02	+0.04
BS 16 lr 2e-5	0.15	+0.02	+0.05
BS 16 lr 1e-6	0.17	+0.02	+0.02

size over 16, 64, and 256, and the learning rate over 2e-5 and 1e-6. We present the results in Table 9 below.

A.13 Prompts for Non-Mathematical Reasoning Task - Code Generation

MBPP CODE GENERATION PROMPT

You are an expert Python programmer. I will give you a programming task description. Your job is to write a correct, efficient, and clean Python solution. Start directly with the coding solution.

Requirements: - Use only the Python standard library. - Your code must strictly satisfy the provided assertion. - Respond with ONLY Python code (no backticks, no comments, no explanations).

Problem:
{problem}

Your code should satisfy the following assertion:
{assertion}

Solution:

Prompt for Code Paraphrasing

You are an expert Python programmer. I am going to give you a full programming task description (including assertions) and a working correct solution (provided as a hint). Your task is to rewrite this correct code solution in your own style while **STRICTLY** preserving the exact logic, functionality, and function signature of the provided hint solution. You do not need to generate a new solution, just rewrite in your own coding style while adhering to the hint solution.

Problem:
problem

Correct Solution (provided as hint) to be Paraphrased:
response

Your Paraphrased Solution: