# THINK, PRUNE, TRAIN, IMPROVE: SCALING REASONING WITHOUT SCALING MODELS

**Caia Costello**[1,2,3]
[1]Scaling Intelligence Lab
[2]Department of Computer Science, Stanford University
[3]Ceramic AI
caia@stanford.edu

## ABSTRACT

Large language models (LLMs) have demonstrated strong capabilities in programming and mathematical reasoning tasks, but are constrained by limited high-quality training data. Synthetic data can be leveraged to enhance fine-tuning outcomes, but several factors influence this process, including model size, synthetic data volume, pruning strategy, and number of fine-tuning rounds. We explore these axes and investigate which conditions enable model self-improvement. We introduce the **Think, Prune, Train** process, a scalable framework that iteratively fine-tunes models on their own reasoning traces, using ground-truth pruning to ensure high-quality training data. This approach yields improved performance: on GSM8K, Gemma2-2B achieves a Pass@1 of 57.6% (from 41.9%), Gemma2-9B reaches 82%, matching LLaMA-3.1-70B, and LLaMA-3.1-70B attains 91%, even surpassing GPT-4o, demonstrating the effectiveness of self-generated reasoning and systematic data selection for improving LLM capabilities.

## 1 INTRODUCTION

State-of-the-art LLMs have been extensively trained on public text, yielding diminishing returns from additional web-scraped data. One promising approach is leveraging curated synthetic data to improve reasoning, an essential part of advancing code generation and mathematical problem-solving.

Recent frontier models like LLaMA 3.1 Dubey et al. (2024) and DeepSeek R1 DeepSeek AI Team (2024) demonstrate that post-training on reasoning traces coupled with supervised fine-tuning (SFT) on filtered (pruned) data works well to improve models. Their strong performance on coding and math benchmarks highlights how properly curated synthetic data can drive substantial performance gains. For smaller models such as LLaMA (1B, 3B) and Gemma (2B) (9B) Team et al. (2024b), distillation Hinton et al. (2015) coupled with fine-tuning on reasoning trace datasets has become the dominant post-training paradigm. However, distillation depends on the availability of larger models, while reasoning training relies on extensive external datasets.

This raises a fundamental question: **Can small models learn to reason using only self-generated data?** Prior attempts at recursive fine-tuning on unfiltered text data Shumailov et al. (2024) have observed *model collapse*: model degradation, including knowledge forgetting Kirkpatrick et al. (2017) and hallucination. However, those works explored simple text generation, a task without a ground truth metric for pruning. In contrast, *mode collapse* refers to a model converging on a narrower set of high-probability outputs, reducing diversity at the cost of exploration. Mode collapse and model collapse are distinct yet interconnected risks in iterative fine-tuning. Our approach, recursive SFT on self-generated reasoning traces, might seem prone to mode collapse, as models increasingly favor confident, correct solutions, but results show that while Pass@1 improves significantly, Pass@50 and Pass@20 remain relatively stable, indicating that the model maintains diversity while prioritizing reliability. By incorporating correctness-based pruning, we aim to avoid both model and mode collapse while leveraging the benefits of iterative refinement for scalable reasoning improvement.

We investigate the conditions that enable self-improvement, particularly in smaller models, to understand when and how models can refine their reasoning abilities without external supervision.

Our analysis demonstrates that reasoning trace selection through pruning can yield significant performance gains. Several factors influence this process, including the **size of the model** used for generating data, the **amount of synthetic data** incorporated during training, the **pruning strategy** employed for selecting reasoning traces, and the **number of fine-tuning rounds** conducted throughout the improvement process. We categorize this process into three components: **Think, Prune, Train**, where models are iteratively fine-tuned on their own *correct* step-by-step reasoning solutions.

1. Prompt models to reason in a structured way.
2. Prune incorrect outputs using ground-truth correctness filtering.
3. Perform SFT on the current model with its own unique validated solutions.

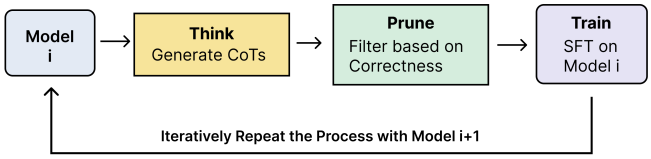



Figure 1: **Recurrent process for model training.** Data generation, pruning, and supervised fine-tuning (SFT), with the arrow indicating the feedback loop to the tuned model.

By applying our process, as illustrated in Figure 1 , on small models, they achieve reasoning performance comparable to larger ones on coding and math benchmarks. In particular, on GSM8K Cobbe et al. (2021b), and CodeContests Li et al. (2022), we achieve the following performance gains:

- On GSM8K, Gemma2-2B improves from 41.9% to 57.6% for Pass@1
- On GSM8K, Gemma2-9B reaches 82% Pass@1, surpassing LLaMA3.1-70B-Instruct's 78% for Pass@1.
- On GSM8K, LLaMA3.1-70B climbs from 78% to 91% Pass@1, outperforming even GPT-4o (2024-08-06).
- On Code Contest, Pass@1 improves from 0.90% to 1.14% for the Gemma2-2B model and from 5.10% to 7.90% for the Gemma2-9B model.

**Our Contributions:**

1. **Avoids model collapse in self-training**: Prior works show that unfiltered recursive fine-tuning can degrade model performance, leading to catastrophic forgetting and hallucination. We demonstrate that **correctness-based pruning** stabilizes training, preserving knowledge while enhancing reasoning.
2. **Insights for iterative reasoning refinement**: We analyze how data volume, model size, pruning strategies, and fine-tuning iterations influence self-improvement and identify the conditions for effective self-improvement.
3. **Think, Prune Train framework**: We evaluate the Think, Prune, Train **(TPT)** framework, demonstrating that structured reasoning prompting, correctness-based pruning, and supervised fine-tuning on validated solutions enables self-improvement.

## 2 Related work

Prior work on improving reasoning in small models includes distillation using synthetic data from larger models, self-improvement strategies, and reinforcement learning (RL)-based optimization. In contrast, we evaluate whether self-improvement can be achieved without relying on teacher models, complex frameworks, or RL.

**Distillation:**

Previous work has shown success in fine-tuning base models with synthetic data from large models, a form of distillation: Alpaca Taori et al. (2023), MagicCoder Wei et al. (2024), and CodeGemma Team et al. (2024a) demonstrate effective single-round supervised fine-tuning (SFT) using data from larger models. Research in math reasoning has focused on structured solutions and synthetic data: GSM8K Cobbe et al. (2021a) provides high-quality grade-school word problems requiring step-by-step solutions, though we found that training solely on this dataset proved insufficient for scaling performance. TinyGSM Liu et al. (2024) creates 12.3M synthetic math problems using GPT-3.5-turbo, achieving 63.1% pass@1 even with a 125M parameter model. MetaMath Lewkowycz et al. (2022) demonstrates the effectiveness of training models on multi-step forward and backward reasoning traces generated by GPT-3.5-Turbo, highlighting the effect of structured problem-solving data in fine-tuning outcomes.

**Self-Improvement:**

Achieving iterative self-improvement using data from the same model presents a challenge. LLaMA 3.1 Dubey et al. (2024) implements multi-round fine-tuning and reports that iterative training on synthetic data from their 405B model improves its 8B and 70B counterparts, but training on its own outputs without pruning led to performance degradation. To address this, they adopted a pipeline consisting of supervised fine-tuning (SFT), followed by direct preference optimization (DPO), additional data generation, and data pruning. ReST Uesato et al. (2023) proposes an offline RL framework that optimizes model behavior using a growing batch of self-generated training data pruned by a reward model, their algorithm aligns the language model's outputs with human preferences, which are modeled using a learned reward function. Self-Instruct, Wang et al. (2023) introduces a scalable method for bootstrapping instruction-following models without requiring human-annotated datasets, by prompting GPT3 to generate diverse instruction-response pairs. Self-Taught Reasoner (STaR) Zelikman et al. (2022) proposes an iterative fine-tuning approach where a model learns to generate its own rationales while progressively improving its problem-solving ability. STaR constructs rationalized solutions through rejection sampling and fine-tunes a base model on both correct and corrected (giving the model answer working backwards) rationales.

**Reinforcement Learning:**

Reinforcement learning (RL) has played a significant role in improving the reasoning, decision-making, and alignment of large-scale language models. Traditional RL-based approaches such as Proximal Policy Optimization (PPO) **?** and Reinforcement Learning from Human Feedback (RLHF) **?** have been widely used to fine-tune models for better instruction-following and alignment. However, newer RL methods have introduced refinements. Vine-PPO **?** enhances policy optimization by incorporating variance reduction techniques, improving convergence stability and sample efficiency. DeepSeek R1 DeepSeek AI Team (2024) applies RL-based rejection sampling to filter and refine reasoning paths. Their pipeline first pre-trains a model with structured reasoning objectives, followed by RL optimization to improve output quality. ReST-EM **?** extends ReST by incorporating Expectation Maximization, dynamically adjusting the weight of self-generated training data based on model confidence. Unlike RL-based methods like DeepSeek R1 and LLaMA 3.1 Dubey et al. (2024) we focus exclusively on recursive correctness-based fine-tuning, demonstrating that self-improvement is possible without RL by leveraging structured prompting and validation.

| Approach | Uses CoT | Only Ground Truth Pruning | Starts from Fine-tune | Multiple Iterations |
|---|---|---|---|---|
| **TPT (Ours)** | ✓ | ✓ | ✓ | ✓ |
| **LLaMA 3.1** Dubey et al. (2024) | ✓ | ✗ | ✓ | ✓ |
| **DeepSeek R1** DeepSeek AI Team (2024) | ✓ | ✗ | ✗ | ✓ |
| **ReSTem** Huang et al. (2023) | ✗ | ✓ | ✗ | ✓ |
| **MagicCoder** Wei et al. (2024) | ✗ | ✗ | ✗ | ✗ |
| **Self-Instruct** Wang et al. (2023) | ✗ | ✗ | ✗ | ✗ |
| **STaR** Zelikman et al. (2022) | ✓ | ✓ | ✗ | ✓ |

Table 1: **Comparison of TPT with Related Approaches**. Compares if each approach uses Chain-of-Thought (CoT), applies ground truth pruning, starts from a fine-tuned model, as opposed to the base model, and performs multiple fine-tuning iterations.

### 2.1 SFT ON PRUNED DATA AS A SPECIAL CASE OF POLICY GRADIENT

Recent work suggests that supervised fine-tuning (SFT) and reinforcement learning (RL) share key similarities. $\text{REST}_{\text{EM}}$ Huang et al. (2023) reframes RL as an Expectation-Maximization (EM) process, where policy updates occur on fixed data rather than dynamically evolving policies. This aligns with our approach, which fine-tunes on correctness-filtered data without explicit policy optimization. In policy gradient RL, which optimizes a parameterized policy $\pi_\theta$ to maximize expected return $J(\pi_\theta)$:

$$\nabla J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau)\right]$$

For language modeling, states correspond to token sequences $s_t = (x_0, ..., x_t)$ and actions to next-token choices $a_t = x_{t+1}$. Sparse rewards $R(\tau)$, such as binary correctness in code generation, are $-1$ for negative $x^l$ examples and $+1$ for positive ones $x^w$.

Because errors in math problems are localized we notice that the gradient of the negative example is of equal magnitude but opposite sign to the positive example on those steps where they differ. Under this assumption, separating out the positive from the negative examples, the gradient simplifies to:

$$\begin{aligned}&\nabla_\theta \log \pi(x^w_{t+1}|x_{0:t}) - \nabla_\theta \log \pi(x^l_{t+1}|x_{0:t}) \\ &= 2\nabla_\theta \log \pi(x^w_{t+1}|x_{0:t}).\end{aligned}$$

This mirrors the standard cross-entropy loss in SFT for just the positive examples:

$$L(\pi_\theta) = -\sum_{t=0}^{T} \log \pi_\theta(x_{t+1}|x_{0:t})$$

Thus, filtering for correctness and fine-tuning on self-generated sequences approximates policy gradient updates. This suggests that SFT on correct outputs is not just a heuristic but an implicit RL formulation, where correctness serves as a reward signal guiding model improvement.

## 3 THINK, PRUNE, TRAIN:

We systematically investigate the key components that enable effective self-improvement without dependence on external models or datasets. Many variables contribute to the effectiveness of self-improvement techniques, such as:

- **Choice of Base Model:** We focus on the Gemma and Llama families, particularly smaller models.
- **Prompting Strategy:** Utilize CoT prompting with a temperature of 0.8 to generate data from the training sets.
- **Dataset Size:** Explore how varying amounts of synthetic data impact fine-tuning outcomes.
- **Comparison with Distillation:** We compare the performance of self-generated data with data generated by larger models.
- **Fine-Tuning Rounds:** Examine the effects of fine-tuning the base model once versus iteratively fine-tuning over multiple rounds
- **Pruning Strategy:** Compare the performance of selecting data based on ground truth, partially correct solutions, and using unfiltered generated solutions.

While prior work often formulates self-improvement as a reinforcement learning (RL) problem, requiring explicit reward models and policy optimization. We apply supervised fine-tuning (SFT) to a model using its own generated outputs, selectively filtering for correctness. Our analysis explores whether iterative refinement can be achieved purely through structured data selection.

### 3.1 CHOOSING DATA FOR SELF-IMPROVEMENT

A key challenge in evaluating iterative self-improvement is disentangling gains due to actual refinement of the model's reasoning capabilities from those caused by training on an increasingly large accumulated dataset Gerstgrasser et al. (2024). To systematically investigate this, we explored different data retention strategies across fine-tuning rounds. Initially, we considered accumulating multiple solutions per question over successive rounds, allowing the model to learn from a more diverse set of responses. While this yields performance improvements, the model's enhancement could be attributed to data size augmentation rather than genuine iterative refinement.

To isolate the effects of self-improvement, we retain only a single, *unique* example per question per round, ensuring that the dataset size remains constant across iterations. Moreover, instead of accumulating data across rounds, we replaced each round's dataset entirely with newly generated solutions, so every model is strictly trained on *only* self-generated data. Our findings suggest that with strict data constraints, iterative fine-tuning can lead to meaningful gains, demonstrating that model improvement is not solely a function of dataset expansion. Our experiments converged to a simple yet effective method for model self-improvement, illustrated in Algorithm 1.

### 3.2 HOW TPT IS DIFFERENT

Prior self-improvement methods, such as ReST Uesato et al. (2023) and the post-training setup of LLaMA 3.1 Dubey et al. (2024), refine models iteratively by applying their process to an already fine-tuned model. ReST prunes self-generated data using a learned reward function, while LLaMA 3.1 applies direct preference optimization (DPO) and prunes data using both reward models and ground-truth filtering.

STaR Zelikman et al. (2022) uses rejection sampling and fine-tunes the base model on both correct and corrected rationales, incorporating backward reasoning where the model reasons from a given answer. However, STaR does not iteratively fine-tune on its improved model.

Our approach focuses solely on supervised fine-tuning (SFT) to examine whether a model can iteratively improve from its own generated outputs using correctness-based filtering as the only selection mechanism. By focusing soley on SFT, while forgoing, RL, reward models, and external correction, we isolate the role of structured data selection in self-improvement.

**Algorithm 1** Think, Prune, Train

**Input:** Base model $M_0$, pruning strategy $P$, amount of data $|n| = f$ used for supervised fine-tuning
**for** $i = 1$ to $N$ **do**
  SFT $M_{i-1}$ on pruned data $f_{i-1} \rightarrow M_i$
  Generate 10 solutions for all problems in the train set using model $M_i$ to construct $S$, the set of all generated solutions
  Apply pruning strategy $P$ to all new solutions $S$
  Randomly sample $f_i \subset S$ as examples for fine-tuning on $M_i$
**end for**
**Output:** Improved model $M_N$

### 3.3 EXPERIMENTAL SETUP

We study the effectiveness of instruction fine-tuned variants of the Llama and Gemma model families. Specifically, we experiment with `gemma2-2b-it`, `gemma2-9b-it`, `Llama-3.2-1B-Instruct`, and `Llama-3.3-70B-Instruct`. Our SFT runs use a learning rate of `1e-6` for Gemma and `1e-5` for Llama to maintain training stability. We employ the AdamW optimizer Loshchilov & Hutter (2017) with default weight decay. Training is conducted over a single epoch with a 10% warm-up of training steps to mitigate overfitting.

The appendix (B.5) contains synthetic example solutions from question 20 of the GSM8K Cobbe et al. (2021a) train set, illustrating the reasoning steps taken by different models. For the iterative fine-tuning experiments using Algorithm 1, we set the number of pruned examples per round to 2000

for GSM8K and 1000 for CodeContests, performing up to $N = 4$ rounds of iterative improvement. Evaluation is conducted at a temperature of 0.7. For GSM8K, we select a subset of 500 questions from the test set, ensuring that answers exactly match the ground truth, including formatting, to enforce both correctness and adherence to the expected answer format. Additionally, we evaluate our recursively trained models on the 140 questions without image tags from CodeContests, using standard input/output matching to verify correctness.

## 4 RESULTS:

Our results are broken into three sections:

- We first analyze how dataset size and the model generating the data impact fine-tuning outcomes.
- Next, we evaluate the Think-Prune-Train process on GSM8K and CodeContests
- Finally, we validate our pruning strategy, demonstrating that pruned synthetic data outperforms unpruned data and that our approach generalizes to LLaMA variants.

**Synthetic data scaling**

**We investigate whether increasing the size of synthetic datasets, and the models used to generate them, leads to performance gains.** Experimental results reveal that more synthetic data does not necessarily translate to better outcomes across all dataset types. We also compare how SFT on increasing amounts of human generated data from the training set effects the model's performance for comparison.

For GSM8K, we observe that small amounts of synthetic data generated by the larger Gemma-9B model outperform those from the 2B model. At 2,000 examples, the 9B-generated data achieves a **Pass@1** of **54.0%**, compared to **52.5%** for 2B-generated data. However, as dataset size increases, performance plateaus or even slightly declines for both 2B and 9B-generated data.

***The results indicate that simply scaling synthetic data is not a universally effective strategy.*** Instead, performance is shaped by the interplay between data quality, dataset size, and the model used for generation.

Table 2: **Scaling synthetic dataset size and data generation model size shows no clear trends, motivating a different solution (TPT) 1.** GSM8K Pass@1% and Pass@20% for Gemma-2-2B and Gemma-2-9B models fine-tuned on various dataset sources, scaling the number of training examples.

| Model | Approach | Size | Data Source | Pass@1(%) | Pass@20(%) |
|---|---|---|---|---|---|
| **Gemma-2-2B** | Baseline | - | - | 41.9 | 76.0 |
| | Self-gen | 1k | 2B | 50.6 | 81.8 |
| | | 2k | 2B | 52.5 | **83.0** |
| | | 4k | 2B | **54.8** | 82.6 |
| | | 6k | 2B | 51.6 | 81.0 |
| | Human | 2k | GSM8K | 45.5 | 83.0 |
| | | 4k | GSM8K | 44.3 | 83.4 |
| | | 6k | GSM8K | 44.0 | 85.6 |
| | Distill | 1k | 9B | 51.2 | 83.1 |
| | | 2k | 9B | **54.0** | **83.8** |
| | | 4k | 9B | 53.1 | 83.0 |
| | | 6k | 9B | 51.2 | 82.6 |
| **Gemma-2-9B** | Baseline | - | - | 66.4 | 88.4 |
| | Self-gen | 1k | 9B | 80.5 | 89.0 |
| | | 2k | 9B | 81.1 | 90.0 |

### 4.1 RECURSIVE DATA PERFORMANCE

**Next, we examine how recursive synthetic data generation influences performance across multiple tasks.** Our investigation into recursive synthetic data generation reveals a pattern of cumulative improvements across multiple tasks, particularly for single-attempt performance. These findings align with Muennighoff et al. (2025), though with a key distinction: while their approach begins with models explicitly trained to reason and uses budget forcing to extend reasoning length, our method starts with plain instruct models and prompts them to develop reasoning behaviors.

**Mathematical Reasoning Recursive Performance:** Through recursive fine-tuning, where each subsequent model (Model1, Model2, etc.) is trained on synthetic data generated by its predecessor, we observe significant gains in mathematical reasoning capability. The Gemma-2B model's exact

| Model | Stage | Source | Data | Pass@1 | Pass@20 | Correct@20 |
|---|---|---|---|---|---|---|
| **Gemma-2-2B** | Baseline | - | - | 41.9 | 76.0 | 380 |
| | Init (Model 1) | 2B | 2k | 52.5 | 83.0 | 415 |
| | Rec 1 (Model 2) | M1 | 2k | 56.0 | 83.0 | 415 |
| | Rec 2 (Model 3) | M2 | 2k | 56.9 | 83.2 | 416 |
| | Rec 3 (Model 4) | M3 | 2k | **57.6** | **83.2** | 416 |
| **Gemma-2-9B** | Baseline | - | - | 66.4 | 88.4 | 442 |
| | Init (Model 1) | 9B | 2k | 81.1 | 90.0 | 450 |
| | Rec 1 (Model 2) | M1 | 2k | 82.3 | 89.4 | 447 |
| | Rec 2 (Model 3) | M2 | 2k | **82.4** | **90.2** | 451 |

Table 3: **Recursive fine-tuning improves GSM8K performance across multiple iterations.** Performance of recursively trained Gemma models on GSM8K, showing Pass@1(%) and Pass@20(%) improvement when following the algorithm, These are the numbers for Figure 2 Pass@1 shows a clearer upward trend with Pass@20 plateauing after 2 rounds.

| Model | Stage | Source | Data | Pass@1 | Pass@50 | Correct@50 |
|---|---|---|---|---|---|---|
| **Gemma-2-2B** (CodeContest) | Baseline | - | - | 0.90 | 5.70 | 8 |
| | Init (Model 1) | 2B | 1k | 0.84 | 7.14 | 10 |
| | Rec (Model 2) | M1 | 1k | 1.00 | 9.20 | 13 |
| | Rec (Model 3) | M2 | 1k | 1.10 | **9.20** | 13 |
| | Rec (Model 4) | M3 | 1k | **1.14** | 7.85 | 11 |
| **Gemma-2-9B** (CodeContest) | Baseline | - | - | 5.10 | 15.00 | 21 |
| | Init (Model 1) | 9B | 1k | 7.30 | 18.50 | 26 |
| | Rec (Model 2) | M1 | 1k | 7.60 | 17.80 | 25 |
| | Rec (Model 3) | M2 | 1k | **7.90** | **18.50** | 26 |

Table 4: **Recursive Fine-Tuning on Code Contest dataset shows minor improvement at Pass@1 and plateaus for Pass@50.** Pass@1 improves steadily through iterative fine-tuning, reaching 1.14% for the 2B model and 7.90% for the 9B model. However, Pass@50(%) exhibits diminishing returns, indicating potential limitations in further refinement. The Correct@50 metric further illustrates that while improvements occur, gains diminish over iterations.

match Pass@1 performance on GSM8K, as seen in Table 2 ,increases from 41.9% to 57.6% over four iterations of recursive training, accompanied by gains in Correct@20. Similarly, the Gemma-9B model demonstrates improvement, reaching a Pass@1 of 82.4% from 66.4 % within three iterations.

While Pass@1 steadily increases, **Pass@20 shows diminishing returns, mostly plateauing after the first iteration,** suggesting that recursive training primarily enhances accuracy rather than improving diversity across sampled generations.

We hypothesize that the reasoning-intensive nature of mathematical tasks enables recursive fine-tuning to build upon previous patterns, generating increasingly accurate and detailed solutions. This phenomenon aligns with findings from STAR Zelikman et al. (2022), which highlights the effectiveness of structured rationale generation for reasoning tasks.

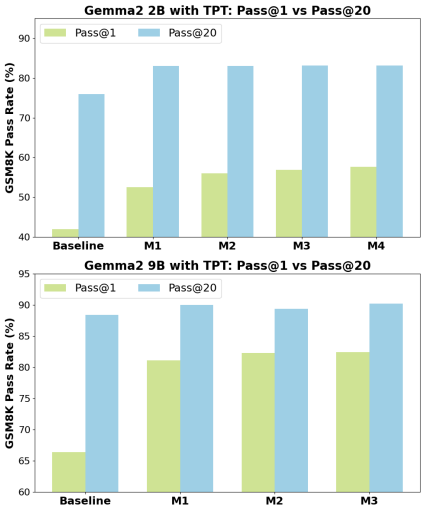



Figure 2: **Recursive training enhances GSM8K Pass@1/20 performance in Gemma models.** On GSM8K Gemma2-2B model's Pass@1(%) performance increases from 41.9% to 57.6% over four iterations of this process 1 While, the Gemma-9B model improves to a Pass@1 of 82.4%. Bars M1-4 represent the models trained though the TPT process, starting with base Gemma2-2B/9B .

**Code Contest Recursive Results:** In the domain of code generation, we observe more nuanced outcomes, **Code Contest pass@1 performance improves but pass@50 plateaus**. The 2B model steadily improves in Code Contest, seen in Table 4 Pass@1, rising from 0.9% to 1.14% after four recursive training iterations. However, Pass@50 plateaus around 7.85–9.2%, possibly due to code generation being less reliant on step-by-step reasoning. We also trained on 1k data each recursive round as opposed to 2k for GSM8K, this is due to the smaller dataset size, as well as lower model accuracy on that dataset.

### 4.2 IMPACT OF PRUNING ON SYNTHETIC DATA QUALITY

**Pruning plays a crucial role in ensuring the effectiveness of self-generated synthetic data** directly influencing model performance. Experiments with un-pruned self-generated data showed deteriorated performance, contrasting with Setlur et al. (2024) which suggests value in incorrect data for mathematical tasks when critical errors are avoided. With pruning, models trained on 2B and 9B-generated data achieve comparable performance. However, without pruning, 9B-generated data significantly outperforms 2B-generated data across all tasks, demonstrating pruning's role in equalizing data quality across model sizes. We also show results from soft pruning experiments where we keep examples that pass one or more tests where we observed increased diversity.

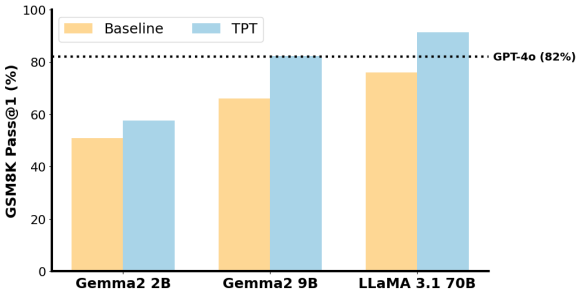
| Model | Stage | Source | Data | Pass@1(%) | Pass@20/50(%) |
|---|---|---|---|---|---|
| **Gemma-2-2B** (GSM8K) | Baseline | - | - | 41.9 | 76.0 |
| | 2k No-Prune | 2B | 2k | 45.13 | 71.6 |
| | 6k No-Prune | 2B | 6k | 44.57 | 71.0 |
| **Gemma-2-2B** (Code Contest) | Baseline | - | - | 0.90 | 5.70 |
| | 1k No-Prune | 2B | 1k | 0.85 | 5.70 |
| | 1k No-Prune (9B Data) | 9B | 1k | 0.94 | 6.40 |
| | 5k No-Prune | 2B | 5k | 0.90 | 6.42 |
| | 5k No-Prune (9B Data) | 9B | 5k | 0.67 | 9.29 |
| | 1k Soft-Pos | 2B | 1k | 0.70 | 8.57 |

Table 5: **SFT without pruning performs poorly, while soft pruning increases diversity.** Pass@1 and Pass@20 and Pass@50, results for GSM8K and Code Contest respectively on Gemma-2-2B models trained on different dataset sizes and pruning strategies.

### 4.3 RESULTS SYNTHESIS:

Our experimental results demonstrate differences in learning efficiency across model scales when applying TPT (Think Prune Train) recursive fine-tuning. The smaller Gemma-2-2B model required four complete recursive TPT rounds to achieve its peak accuracy, while the medium-sized Gemma-2-9B improved faster after three rounds. In contrast, the significantly larger LLaMA-70B model attained 91.5% pass@1 accuracy after merely a single round of TPT recursive fine-tuning. The enhanced Pass@1 metric is noteworthy as it measures the model's ability to generate correct solutions on the first attempt. LLaMA-70B's 91.5% pass@1 score indicates that TPT effectively transfers reasoning abilities even with minimal self-generated training iterations when applied to larger model architectures.These findings highlight the importance of considering model parameter size when designing training regimes.

Figure 3: **Recursive Fine-Tuning Can Improve Over Open Source Models.** The `Gemma-2-9B` model improves from 66.0% to 82.4%, while `LLaMA-70B` improves from 76.0% to 91.5% Pass@1, surpassing `GPT-4o` (82%). This figure highlights the performance gains achieved through recursive fine-tuning.




| Model | Stage | Source | Data | Pass@1(%) | Pass@20(%) |
|---|---|---|---|---|---|
| **LLaMA-1B (GSM8K)** | Baseline | - | - | 18.2 | 65.2 |
| | 2k Fine-Tune | LLaMA-1B | 2k | 20.8 | 68.4 |
| **LLaMA-70B (GSM8K)** | Baseline | - | - | 78.6 | 92.2 |
| | 6k Fine-Tune | LLaMA-70B | 6k | **91.5** | **95.8** |

Table 6: **The Think-Prune-Train method improves GSM8K performance on LLaMA models** One round of the TPT process improves performance. Notably, LLaMA-70B achieves a Pass@1 increase from 78.6% to 91.5%

### 4.4 POSSIBLE MODE COLLAPSE

Our observed Pass@1 improvements alongside stable Pass@20 and Pass@50 metrics suggest a form of *mode collapse* Kossale et al. in recursive synthetic data generation, where the model prioritizes high-confidence solutions over diversity. Mode collapse occurs when a model converges on a narrower set of high-probability outputs at the expense of diversity. This phenomenon has been studied in recent work on synthetic data feedback loops, including **?**, which explores the long-term effects of training models on their own synthetic outputs.

The authors argue that recursive training pipelines without sufficient real and synthetic data accumulation can lead to progressively narrowing distributions, where models produce increasingly homogeneous outputs, and performance degradation. While their work emphasizes the dangers of such feedback loops—particularly in open-ended, creative tasks—our setting differs in that correctness is strictly defined. We do observe less diversity in outputs, but increased correctness, this outcome is not inherently detrimental, particularly for domains such as mathematics and programming, where correctness and efficiency take precedence over diversity.

Though we observe reduced output variety, increased accuracy aligns with our goal of enhancing precision through pruning. This effect may relate to fine-tuning's impact on hallucinations Kang et al. (2024). If diversity were the priority, techniques like soft-positive sampling, as seen in in Table 5, or data accumulation would be preferable, however our iterative approach naturally reinforces high-confidence, precise outputs.

## 5 CONCLUSION

This study presents an analytical investigation of what factors influence self-improvement in language models, centering on our **Think, Prune, Train** framework. Our findings challenge the notion that simply scaling synthetic dataset size and model scale is sufficient to enhance reasoning capabilities. We demonstrate that **correctness-based pruning** stabilizes training, preserving knowledge while enhancing reasoning. Our experiments highlight how recursive fine-tuning improves first-attempt accuracy, with **Gemma-2B increasing from 41.9% to 57.6% Pass@1, Gemma-9B reaching 82%, and LLaMA-70B-Instruct achieving 91%, surpassing GPT-4o's 82%**. Additionally, we identify key conditions for effective self-improvement, analyzing the impact of data volume, model size, pruning strategies, and fine-tuning iterations. This provides insights into iterative reasoning refinement, showing that structured selection plays a more critical role than sheer dataset size.

Finally, we evaluate the **Think, Prune, Train (TPT)** framework, demonstrating that structured reasoning prompting, correctness-based pruning, and supervised fine-tuning on validated solutions enable scalable self-improvement without external supervision, highlighting the potential of simplistic frameworks to unlock further advances in LLM reasoning and accuracy.

### ACKNOWLEDGMENT

I would like to thank Azalia Mirhoseini for her invaluable guidance and mentorship throughout this project. Thank you to Anna Goldie for all her help and support, and to Simon Guo for being incredibly helpful and insightful. I'm also grateful to Rishabh and Adrian for keeping me company during many long hours in Gates. Special thanks to Tom Costello for always supporting me, and to the Scaling Intelligence Lab for creating such an inspiring environment. I also thank AMD for their GPU sponsorship that enabled the LLaMA 70B runs, and Ceramic AI for access to their cluster for fine-tuning experiments.

# APPENDIX

## 6 DISCUSSION

*Complexity of Reasoning* We have focused on GSM8K rather than more challenging datasets like GPQA because we consider smaller models with less memory and aim to study their reasoning capabilities rather than their recall ability. The *reasoning* in more difficult problems is often not much more complicated than in grade school math but just covers more esoteric subjects. Consider a problem from GPQA Rein et al. (2023) that asks about the angles of drops of liquid on a smooth and rough surface. We paraphrase the GPQA question here.

**Complex Math Problem**

```
On a surface, we apply a coating which
makes it completely smooth. We put a
drop of water and a hydrocarbon on it
and measure the contact angles as 130°
and 102°. We then roughen the surface
and measure the angle of water as 150°.
What is the contact angle for a lighter
hydrocarbon on the roughened surface?
126°,131°,136°,141°
```

The Cassie-Baxter equation says that if $\theta_1$ is the contact angle on a smooth surface, then the angle on a rough surface, $\theta$, obeys $\cos(\theta) = f * \cos(\theta_1) + f$, where $f$ is a constant for the surface. The $f$ for the rough surface can be found from the two angles for water, $\cos(150) = f * \cos(130) - f$, so $f = 0.53$. Then, we can find the rough angle for the hydrocarbon, $\theta_h$, by using the formula $cos(\theta_h) = 0.53 * \cos(105) - 0.53$, so $\theta_h = 131$. The other hydrocarbon is lighter, so it has lower surface tension; thus, its angle is less than 131. The only choice that is less than 131 is 126.

This does not seem easy because few can readily recall (if ever knew) the formulas for the wettability of rough and flat surfaces. Once reminded of —the Cassie Baxter equation—the problem becomes isomorphic to the following.

Given two numbers, $x$, and $y$, and a formula $f$, work out another number, $z$, using the equation. $x = f(y, z)$ Then, use $z$ and another instance of the formula to compute $w$ where $w = f(y', z)$. We are told $w'$ is less than $w$, and given only one choice that is less than $w$. If $f$ is invertible then this is grade school math.

The GPQA problem has the same type of reasoning as a grade school problem. The difference is using a obscure formulas that an LLM cannot recall.

**Simple Math Problem**

```
Jane has a number of buckets. 3 apples
fit in each bucket.  She can fit 15
apples in her buckets. If a bucket
can fit 4 lemons, and limes are
smaller than lemons, how many limes
can fit in her buckets?
10, 15, 19, 24
```

GPQA questions often do not have complex reasoning but are difficult due to the subject matter. As we aim to study reasoning, as opposed to the ability of the models to remember difficult facts, we believe GSM8K sufficient.

## 7 SCALING SYNTHETIC DATA FOR CODE CONTEST AND LEETCODE

Our investigation into the impact of scaling synthetic code data reveals that increasing the amount of synthetic data does not always yield better performance.

For example, in the Code Contest dataset, a Gemma-2B model trained with 2,000 self-generated and filtered synthetic examples outperformed the same model trained on synthetic data from Gemma-9B. However, as our ablation studies indicate, when unpruned, synthetic data from the 9B model leads to better performance than 2B-generated data.

This suggests that smaller models can generate high-quality data that, when validated effectively, can match or even surpass the usefulness of data from larger models. This trend is seen across datasets, reinforcing the importance of careful data curation and filtering.

### 7.1 DIMINISHING RETURNS IN SYNTHETIC DATA SCALING

Pass@1 performance on Code Contest reveals an interesting trend: while synthetic data generated by Gemma-9B improves performance initially, further scaling results in stagnation or decline. At 4,000 and 6,000 synthetic examples, performance either levels off or worsens.

Similarly, introducing small amounts of human-generated training data negatively impacts Pass@1, possibly due to an out-of-distribution effect.

The LeetCode dataset shows a smaller performance gap between synthetic data generated by Gemma-2B and Gemma-9B. A model trained on 1,000 2B-generated examples achieved a Pass@1 of 14.65%, slightly outperforming a model trained on 9B-generated examples, which achieved 12.82%. Despite this, both models performed similarly on Pass@20 and Correct@100, again suggesting that smaller models can generate useful training data.

### 7.2 IMPLICATIONS FOR SYNTHETIC DATA GENERATION STRATEGIES

These results demonstrate that simply increasing synthetic data volume does not guarantee better outcomes. Instead, there is a complex interplay between dataset size, data quality, and the model used for data generation.

Our findings highlight the need for targeted synthetic data generation strategies and reinforce the potential for smaller models to contribute meaningfully to model improvement.

| Model | Stage | Source | Data | Pass@1(%) | Pass@50(%) |
|---|---|---|---|---|---|
| **Gemma-2B (Code Contest)** | Baseline | - | - | 0.90 | 5.70 |
| | 1k-2B-synth | Gemma-2B | 1k | 0.84 | 7.14 |
| | 2k-2B-synth | Gemma-2B | 2k | 0.71 | 7.14 |
| | 4k-2B-synth | Gemma-2B | 4k | 0.56 | 7.85 |
| | 6k-2B-synth | Gemma-2B | 6k | 0.66 | 8.57 |
| **Human Data (Code Contest)** | 1k-Human | Trainset | 1k | 0.60 | 6.43 |
| | 2k-Human | Trainset | 2k | 0.41 | 7.10 |
| | 4k-Human | Trainset | 4k | 0.24 | 5.71 |
| | 6k-Human | Trainset | 6k | 0.29 | 5.00 |
| **Gemma-9B (Code Contest)** | Baseline | - | - | 5.20 | 15.00 |
| | 1k-9B-synth | Gemma-9B | 1k | 7.30 | 17.80 |
| | 2k-9B-synth | Gemma-9B | 2k | 7.10 | 18.50 |
| | 4k-9B-synth | Gemma-9B | 4k | 6.70 | 17.10 |

Table 7: **Code Contest Scaling Amount of Synthetic Data Performance.** Pass@1 and Pass@50 performance on different sizes of synthetic data.

### 7.3 IMPACT OF MIXING REAL AND SYNTHETIC DATA

We also explored the impact of combining small amounts of real and synthetic data for training. Our findings indicate that mixed datasets performed well in terms of Pass@1 scores, particularly in smaller training sizes (e.g., `1k 500 each`). However, attempts to recursively refine models using mixed datasets consistently failed. A possible explanation is that the introduction of real data disrupts the recursive self-improvement process, making it more difficult for the model to generalize effectively from its own outputs.

| Model | Stage | Source | Data | Pass@1(%) | Pass@20(%) |
|---|---|---|---|---|---|
| **Gemma-2B (LeetCode)** | Baseline | - | - | 13.66 | 36.60 |
| | 1k-2B-synth | Gemma-2B | 1k | 14.65 | 33.66 |
| | 1k-9B-synth | Gemma-9B | 1k | 12.82 | 33.66 |
| **Gemma-9B (LeetCode)** | Baseline | - | - | 13.66 | 36.60 |
| | 1k-9B-synth | Gemma-9B | 1k | 14.65 | 33.66 |

Table 8: **LeetCode Scaling Amount of Synthetic Data Performance.** Pass@1 and Pass@20 performance with different sizes of synthetic data.

| Model | Stage | Source | Data | Pass@1(%) | Pass@50(%) |
|---|---|---|---|---|---|
| **Gemma-2B** (Code Contest, Mixed) | 1k-500-each | Trainset/2B | 1k | 1.0 | 7.14 |
| | 2k-1k-each | Trainset/2B | 2k | 0.60 | 7.85 |
| | 4k-2k-each | Trainset/2B | 4k | 0.24 | 5.71 |
| **rec mixed-sft** | 1k both on 1k both | 1kboth | 1k | 0.87 | 7.14 |
| | 2B1k on base | Gemma-2B | 1k | 0.92 | 4.28 |

Table 9: **Mixed Dataset Ablations.** Evaluating model performance when fine-tuning on a mix of real and synthetic data.

### 7.4 THE COT PROMPT USED

The following is the Chain-of-Thought (CoT) prompt used in our experiments.

**Mathematical Reasoning Prompt**

**You are an expert mathematician.**
You are provided with a math problem. Your task is to solve the problem step-by-step, clearly showing all relevant calculations and reasoning.

**Problem:** `"Bella bought stamps at the post office. Some of the stamps had a snowflake design, some had a truck design, and some had a rose design. Bella bought 11 snowflake stamps. She bought 9 more truck stamps than snowflake stamps, and 13 fewer rose stamps than truck stamps. How many stamps did Bella buy in all?"`

**Requirements:**

1. Provide a complete and correct solution in a markdown block.
2. Explain each step of the solution in detail.
3. Conclude with the final numerical answer on a new line in the format `#### [Answer]`, replacing `[Answer]` with the actual answer.

**Solution:**

### 7.5 EXAMPLE REASONING TRACES

**Gemma 2B**

```
Solution: Here's a step-by-step solution to the problem:
**1. Calculate the number of truck stamps:**
Bella bought 9 more truck stamps than snowflake stamps, and she bought 11 snowflake stamps.
Therefore, she bought 11 + 9 = 20 truck stamps.
**2. Calculate the number of rose stamps:**
Bella bought 13 fewer rose stamps than truck stamps, and she bought 20 truck stamps.
Therefore, she bought 20 - 13 = 7 rose stamps.
**3. Calculate the total number of stamps:**
Bella bought 11 snowflake stamps + 20 truck stamps + 7 rose stamps = 38 stamps in total.
**#### 38**
```

**Gemma 9B**

```
Solution: Here's the step-by-step solution to the problem:
1.**Find the number of truck stamps:**
* Bella bought 9 more truck stamps than snowflake stamps, which she bought 11 of.
* Therefore, she bought 11 + 9 = 20 truck stamps.
2. **Find the number of rose stamps:**
* Bella bought 13 fewer rose stamps than truck stamps, which she bought 20 of.
* Therefore, she bought 20 - 13 = 7 rose stamps.
3. **Calculate the total number of stamps:**
* To find the total number of stamps, add the number of each type of stamp: 11 snowflake stamps + 20
truck stamps + 7 rose stamps = 38 stamps.
#### 38
```

LLaMA 70B

**Solution:** ### Step-by-step solution to the problem
To find the total number of stamps Bella bought, we first need to determine the number of stamps for each design. We are given the following information:
- The number of snowflake stamps = 11
- The number of truck stamps = 9 more than snowflake stamps
- The number of rose stamps = 13 fewer than truck stamps
### Calculating the number of truck stamps
Since Bella bought 9 more truck stamps than snowflake stamps, we calculate the number of truck stamps by adding 9 to the number of snowflake stamps.
Number of truck stamps = Number of snowflake stamps + 9
= 11 + 9
= 20
### Calculating the number of rose stamps
Given that Bella bought 13 fewer rose stamps than truck stamps, we calculate the number of rose stamps by subtracting 13 from the number of truck stamps.
Number of rose stamps = Number of truck stamps - 13
= 20 - 13
= 7
### Calculating the total number of stamps
To find the total number of stamps, we add the number of snowflake, truck, and rose stamps together.
Total number of stamps = Number of snowflake stamps + Number of truck stamps + Number of rose stamps
= 11 + 20 + 7
= 38
#### 38

LLaMA 1B

**Solution:** ### Step 1: Interpret the Problem and Identify Given Information
### Step 2: Set Up Equations Based on Given Information
From the problem, we can set up the following equations:
1. `s = 11` (number of snowflake stamps)
2. `t = s + 9` (number of truck stamps)
3. `r = t - 13` (number of rose stamps)
### Step 3: Express the Number of Rose Stamps in Terms of Snowflake Stamps and Truck Stamps
Now, we'll substitute the expressions for `t` and `r` into the third equation:
`s = 11`
`t = s + 9`
`r = (s + 9) - 13`
### Step 4: Solve for the Number of Rose Stamps
Let's simplify the expression for `r`:
`r = (s + 9) - 13`
`r = s - 4`
So, the number of rose stamps in terms of the number of snowflake stamps is `r = s - 4`.
### Step 5: Find the Total Number of Stamps
The total number of stamps is the sum of snowflake stamps, truck stamps, and rose stamps. Therefore, we can write an equation representing the total number of stamps:
`s + t + r = 11 + (11 + 9) + (s - 4)`
Simplify and solve for `s` (snowflake stamps):
`s + t + r = 11 + 20 + s - 4`
`s + t + r = 26 + s`
`s - s = 26`
`s = 26`
Since `s` represents the number of snowflake stamps and we know `s = 11`, the actual number of snowflake stamps is indeed 11. Let's confirm this with the other equations.
### Step 6: Confirm the Number of Truck and Rose Stamps
Now, we'll find the number of truck stamps `t` and rose stamps `r` using the equations:
`t = s + 9`
`r = t - 13`
We found `s = 11`, so:
`t = 11 + 9`
`t = 20`
`r = 20 - 13`
`r = 7`
## Step 7: Verify the Total Number of Stamps Calculations
To ensure the correctness of our solution:
`s + t + r = 11 + 20 + 7`
`s + t + r = 38`
Using the original formula `20 + 7 - 4 = 37`, this confirms our conclusion that `s = 11` is a perfect and easy-to-explain solution that satisfies all given conditions.
### Step 8: Conclusion with the Final Answer
We finally have the total number of stamps Bella bought in all - 11 snowflake stamps, 20 truck stamps, and 7 rose stamps.
#### 38

## REFERENCES

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems. *arXiv preprint arXiv:2110.14168*, 2021a.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. Training Verifiers to Solve Math Word Problems, 2021b. URL https://arxiv.org/abs/2110.14168.

DeepSeek AI Team. DeepSeek R1: Advancing Open-Source Language Models with Reinforcement Learning and Supervised Fine-Tuning. *arXiv preprint arXiv:2404.14618*, 2024.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurelien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Roziere, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Gregoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, Khalid El-Arini, Krithika Iyer, Kshitiz Malik, Kuenley Chiu, Kunal Bhalla, Lauren Rantala-Yeary, Laurens van der Maaten, Lawrence Chen, Liang Tan, Liz Jenkins, Louis Martin, Lovish Madaan, Lubo Malo, Lukas Blecher, Lukas Landzaat, Luke de Oliveira, Madeline Muzzi, Mahesh Pasupuleti, Mannat Singh, Manohar Paluri, Marcin Kardas, Mathew Oldham, Mathieu Rita, Maya Pavlova, Melanie Kambadur, Mike Lewis, Min Si, Mitesh Kumar Singh, Mona Hassan, Naman Goyal, Narjes Torabi, Nikolay Bashlykov, Nikolay Bogoychev, Niladri Chatterji, Olivier Duchenne, Onur Çelebi, Patrick Alrassy, Pengchuan Zhang, Pengwei Li, Petar Vasic, Peter Weng, Prajjwal Bhargava, Pratik Dubal, Praveen Krishnan, Punit Singh Koura, Puxin Xu, Qing He, Qingxiao Dong, Ragavan Srinivasan, Raj Ganapathy, Ramon Calderer, Ricardo Silveira Cabral, Robert Stojnic, Roberta Raileanu, Rohit Girdhar, Rohit Patel, Romain Sauvestre, Ronnie Polidoro, Roshan Sumbaly, Ross Taylor, Ruan Silva, Rui Hou, Rui Wang, Saghar Hosseini, Sahana Chennabasappa, Sanjay Singh, Sean Bell, Seohyun Sonia Kim, Sergey Edunov, Shaoliang Nie, Sharan Narang, Sharath Raparthy, Sheng Shen, Shengye Wan, Shruti Bhosale, Shun Zhang, Simon Vandenhende, Soumya Batra, Spencer Whitman, Sten Sootla, Stephane Collot, Suchin Gururangan, Sydney Borodinsky, Tamar Herman, Tara Fowler, Tarek Sheasha, Thomas Georgiou, Thomas Scialom, Tobias Speckbacher, Todor Mihaylov, Tong Xiao, Ujjwal Karn, Vedanuj Goswami, Vibhor Gupta, Vignesh Ramanathan, Viktor Kerkez, Vincent Gonguet, Virginie Do, Vish Vogeti, Vladan Petrovic, Weiwei Chu, Wenhan Xiong, Wenyin Fu, Whitney Meers, Xavier Martinet, Xiaodong Wang, Xiaoqing Ellen Tan, Xinfeng Xie, Xuchao Jia, Xuewei Wang, Yaelle Goldschlag, Yashesh Gaur, Yasmine Babaei, Yi Wen, Yiwen Song, Yuchen Zhang, Yue Li, Yuning Mao, Zacharie Delpierre Coudert, Zheng Yan, Zhengxing Chen, Zoe Papakipos, Aaditya Singh, Aaron Grattafiori, Abha Jain, Adam Kelsey, Adam Shajnfeld, Adithya Gangidi, Adolfo Victoria, Ahuva Goldstand, Ajay Menon, Ajay Sharma, Alex Boesenberg, Alex Vaughan, Alexei Baevski, Allie Feinstein, Amanda Kallet, Amit Sangani, Anam Yunus, Andrei Lupu, Andres Alvarado, Andrew Caples, Andrew Gu, Andrew Ho, Andrew Poulton, Andrew Ryan, Ankit Ramchandani, Annie Franco, Aparajita Saraf, Arkabandhu Chowdhury, Ashley Gabriel, Ashwin Bharambe, Assaf Eisenman, Azadeh Yazdan, Beau James, Ben Maurer, Benjamin Leonhardi, Bernie Huang, Beth Loyd, Beto De Paola, Bhargavi Paranjape, Bing Liu, Bo Wu, Boyu Ni, Braden Hancock, Bram Wasti, Brandon Spence, Brani Stojkovic, Brian Gamido, Britt Montalvo, Carl Parker, Carly Burton, Catalina Mejia, Changhan Wang, Changkyu Kim, Chao Zhou, Chester Hu, Ching-Hsiang Chu, Chris Cai, Chris Tindal, Christoph Feichtenhofer, Damon Civin, Dana Beaty, Daniel Kreymer, Daniel Li, Danny Wyatt, David Adkins, David Xu, Davide Testuggine, Delia David, Devi Parikh, Diana Liskovich, Didem Foss, Dingkang Wang, Duc Le, Dustin Holland, Edward Dowling, Eissa Jamil, Elaine Montgomery, Eleonora Presani, Emily Hahn, Emily Wood, Erik Brinkman, Esteban Arcaute, Evan Dunbar, Evan Smothers, Fei Sun, Felix Kreuk, Feng Tian, Firat Ozgenel, Francesco

Caggioni, Francisco Guzmán, Frank Kanayet, Frank Seide, Gabriela Medina Florez, Gabriella Schwarz, Gada Badeer, Georgia Swee, Gil Halpern, Govind Thattai, Grant Herman, Grigory Sizov, Guangyi, Zhang, Guna Lakshminarayanan, Hamid Shojanazeri, Han Zou, Hannah Wang, Hanwen Zha, Haroun Habeeb, Harrison Rudolph, Helen Suk, Henry Aspegren, Hunter Goldman, Ibrahim Damlaj, Igor Molybog, Igor Tufanov, Irina-Elena Veliche, Itai Gat, Jake Weissman, James Geboski, James Kohli, Japhet Asher, Jean-Baptiste Gaya, Jeff Marcus, Jeff Tang, Jennifer Chan, Jenny Zhen, Jeremy Reizenstein, Jeremy Teboul, Jessica Zhong, Jian Jin, Jingyi Yang, Joe Cummings, Jon Carvill, Jon Shepard, Jonathan McPhie, Jonathan Torres, Josh Ginsburg, Junjie Wang, Kai Wu, Kam Hou U, Karan Saxena, Karthik Prasad, Kartikay Khandelwal, Katayoun Zand, Kathy Matosich, Kaushik Veeraraghavan, Kelly Michelena, Keqian Li, Kun Huang, Kunal Chawla, Kushal Lakhotia, Kyle Huang, Lailin Chen, Lakshya Garg, Lavender A, Leandro Silva, Lee Bell, Lei Zhang, Liangpeng Guo, Licheng Yu, Liron Moshkovich, Luca Wehrstedt, Madian Khabsa, Manav Avalani, Manish Bhatt, Maria Tsimpoukelli, Martynas Mankus, Matan Hasson, Matthew Lennie, Matthias Reso, Maxim Groshev, Maxim Naumov, Maya Lathi, Meghan Keneally, Michael L. Seltzer, Michal Valko, Michelle Restrepo, Mihir Patel, Mik Vyatskov, Mikayel Samvelyan, Mike Clark, Mike Macey, Mike Wang, Miquel Jubert Hermoso, Mo Metanat, Mohammad Rastegari, Munish Bansal, Nandhini Santhanam, Natascha Parks, Natasha White, Navyata Bawa, Nayan Singhal, Nick Egebo, Nicolas Usunier, Nikolay Pavlovich Laptev, Ning Dong, Ning Zhang, Norman Cheng, Oleg Chernoguz, Olivia Hart, Omkar Salpekar, Ozlem Kalinli, Parkin Kent, Parth Parekh, Paul Saab, Pavan Balaji, Pedro Rittner, Philip Bontrager, Pierre Roux, Piotr Dollar, Polina Zvyagina, Prashant Ratanchandani, Pritish Yuvraj, Qian Liang, Rachad Alao, Rachel Rodriguez, Rafi Ayub, Raghotham Murthy, Raghu Nayani, Rahul Mitra, Raymond Li, Rebekkah Hogan, Robin Battey, Rocky Wang, Rohan Maheswari, Russ Howes, Ruty Rinott, Sai Jayesh Bondu, Samyak Datta, Sara Chugh, Sara Hunt, Sargun Dhillon, Sasha Sidorov, Satadru Pan, Saurabh Verma, Seiji Yamamoto, Sharadh Ramaswamy, Shaun Lindsay, Shaun Lindsay, Sheng Feng, Shenghao Lin, Shengxin Cindy Zha, Shiva Shankar, Shuqiang Zhang, Shuqiang Zhang, Sinong Wang, Sneha Agarwal, Soji Sajuyigbe, Soumith Chintala, Stephanie Max, Stephen Chen, Steve Kehoe, Steve Satterfield, Sudarshan Govindaprasad, Sumit Gupta, Sungmin Cho, Sunny Virk, Suraj Subramanian, Sy Choudhury, Sydney Goldman, Tal Remez, Tamar Glaser, Tamara Best, Thilo Kohler, Thomas Robinson, Tianhe Li, Tianjun Zhang, Tim Matthews, Timothy Chou, Tzook Shaked, Varun Vontimitta, Victoria Ajayi, Victoria Montanez, Vijai Mohan, Vinay Satish Kumar, Vishal Mangla, Vítor Albiero, Vlad Ionescu, Vlad Poenaru, Vlad Tiberiu Mihailescu, Vladimir Ivanov, Wei Li, Wenchen Wang, Wenwen Jiang, Wes Bouaziz, Will Constable, Xiaocheng Tang, Xiaofang Wang, Xiaojian Wu, Xiaolan Wang, Xide Xia, Xilun Wu, Xinbo Gao, Yanjun Chen, Ye Hu, Ye Jia, Ye Qi, Yenda Li, Yilin Zhang, Ying Zhang, Yossi Adi, Youngjin Nam, Yu, Wang, Yuchen Hao, Yundi Qian, Yuzi He, Zach Rait, Zachary DeVito, Zef Rosnbrick, Zhaoduo Wen, Zhenyu Yang, and Zhiwei Zhao. The Llama 3 Herd of Models, 2024. URL https://arxiv.org/abs/2407.21783.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, Daniel A. Roberts, Diyi Yang, David L. Donoho, and Sanmi Koyejo. Is Model Collapse Inevitable? Breaking the Curse of Recursion by Accumulating Real and Synthetic Data, 2024. URL https://arxiv.org/abs/2404.01413.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network, 2015. URL https://arxiv.org/abs/1503.02531.

Andy Huang, Aakanksha Chowdhery, Xinyun Chen, Lei Hou, Quoc Le, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.

Katie Kang, Eric Wallace, Claire Tomlin, Aviral Kumar, and Sergey Levine. Unfamiliar Finetuning Examples Control How Language Models Hallucinate, 2024. URL https://arxiv.org/abs/2403.05612.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–

3526, 2017. doi: 10.1073/pnas.1611835114. URL https://www.pnas.org/doi/abs/10.1073/pnas.1611835114.

Youssef Kossale, Mohammed Airaj, and Aziz Darouichi. Mode Collapse in Generative Adversarial Networks: An Overview, year=2022, In *2022 8th International Conference on Optimization and Applications (ICOA)*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving Quantitative Reasoning Problems with Language Models, 2022. URL https://arxiv.org/abs/2206.14858.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien de Masson d'Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson, Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code generation with AlphaCode. *Science*, 378(6624):1092–1097, December 2022. ISSN 1095-9203. doi: 10.1126/science.abq1158. URL http://dx.doi.org/10.1126/science.abq1158.

Hao Liu, Jingyu Zhou, Yichong Bai, Austin Tang, Chris Manning, and Quoc V. Le. TinyGSM: Scaling Small Language Models with Large-Scale Synthetic Data for Math Reasoning. *arXiv preprint arXiv:2401.07301*, 2024.

Ilya Loshchilov and Frank Hutter. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling, 2025. URL https://arxiv.org/abs/2501.19393.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. GPQA: A Graduate-Level Google-Proof QA Benchmark, 2023. URL https://arxiv.org/abs/2311.12022.

Amrith Setlur, Saurabh Garg, Xinyang Geng, Naman Garg, Virginia Smith, and Aviral Kumar. RL on Incorrect Synthetic Data Scales the Efficiency of LLM Math Reasoning by Eight-Fold, 2024. URL https://arxiv.org/abs/2406.14532.

I. Shumailov, Z. Shumaylov, Y. Zhao, and et al. AI Models Collapse When Trained on Recursively Generated Data. *Nature*, 631:755–759, 2024. doi: 10.1038/s41586-024-07566-y.

R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto. Stanford Alpaca: An Instruction-Following Llama Model, 2023. URL https://github.com/tatsu-lab/stanford_alpaca. Accessed: 2024-09-23.

CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A. Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Zhitao Gong, Jane Fine, Tris Warkentin, Ale Jakse Hartman, Bin Ni, Kathy Korevec, Kelly Schaefer, and Scott Huffman. CodeGemma: Open Code Models Based on Gemma, 2024a. URL https://arxiv.org/abs/2406.11409.

Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, Johan Ferret, Peter Liu, Pouya Tafti, Abe Friesen, Michelle Casbon, Sabela Ramos, Ravin Kumar, Charline Le Lan, Sammy Jerome, Anton Tsitsulin, Nino Vieillard, Piotr Stanczyk, Sertan Girgin, Nikola Momchev, Matt Hoffman, Shantanu Thakoor, Jean-Bastien Grill, Behnam Neyshabur, Olivier Bachem, Alanna Walton, Aliaksei Severyn, Alicia Parrish, Aliya Ahmad, Allen Hutchison, Alvin Abdagic, Amanda Carl, Amy Shen, Andy Brock, Andy Coenen, Anthony Laforge, Antonia Paterson, Ben Bastian, Bilal Piot, Bo Wu, Brandon Royal, Charlie Chen, Chintu Kumar,

Chris Perry, Chris Welty, Christopher A. Choquette-Choo, Danila Sinopalnikov, David Weinberger, Dimple Vijaykumar, Dominika Rogozińska, Dustin Herbison, Elisa Bandy, Emma Wang, Eric Noland, Erica Moreira, Evan Senter, Evgenii Eltyshev, Francesco Visin, Gabriel Rasskin, Gary Wei, Glenn Cameron, Gus Martins, Hadi Hashemi, Hanna Klimczak-Plucińska, Harleen Batra, Harsh Dhand, Ivan Nardini, Jacinda Mein, Jack Zhou, James Svensson, Jeff Stanway, Jetha Chan, Jin Peng Zhou, Joana Carrasqueira, Joana Iljazi, Jocelyn Becker, Joe Fernandez, Joost van Amersfoort, Josh Gordon, Josh Lipschultz, Josh Newlan, Ju yeong Ji, Kareem Mohamed, Kartikeya Badola, Kat Black, Katie Millican, Keelin McDonell, Kelvin Nguyen, Kiranbir Sodhia, Kish Greene, Lars Lowe Sjoesund, Lauren Usui, Laurent Sifre, Lena Heuermann, Leticia Lago, Lilly McNealus, Livio Baldini Soares, Logan Kilpatrick, Lucas Dixon, Luciano Martins, Machel Reid, Manvinder Singh, Mark Iverson, Martin Görner, Mat Velloso, Mateo Wirth, Matt Davidow, Matt Miller, Matthew Rahtz, Matthew Watson, Meg Risdal, Mehran Kazemi, Michael Moynihan, Ming Zhang, Minsuk Kahng, Minwoo Park, Mofi Rahman, Mohit Khatwani, Natalie Dao, Nenshad Bardoliwalla, Nesh Devanathan, Neta Dumai, Nilay Chauhan, Oscar Wahltinez, Pankil Botarda, Parker Barnes, Paul Barham, Paul Michel, Pengchong Jin, Petko Georgiev, Phil Culliton, Pradeep Kuppala, Ramona Comanescu, Ramona Merhej, Reena Jana, Reza Ardeshir Rokni, Rishabh Agarwal, Ryan Mullins, Samaneh Saadat, Sara Mc Carthy, Sarah Perrin, Sébastien M. R. Arnold, Sebastian Krause, Shengyang Dai, Shruti Garg, Shruti Sheth, Sue Ronstrom, Susan Chan, Timothy Jordan, Ting Yu, Tom Eccles, Tom Hennigan, Tomas Kocisky, Tulsee Doshi, Vihan Jain, Vikas Yadav, Vilobh Meshram, Vishal Dharmadhikari, Warren Barkley, Wei Wei, Wenming Ye, Woohyun Han, Woosuk Kwon, Xiang Xu, Zhe Shen, Zhitao Gong, Zichuan Wei, Victor Cotruta, Phoebe Kirk, Anand Rao, Minh Giang, Ludovic Peran, Tris Warkentin, Eli Collins, Joelle Barral, Zoubin Ghahramani, Raia Hadsell, D. Sculley, Jeanine Banks, Anca Dragan, Slav Petrov, Oriol Vinyals, Jeff Dean, Demis Hassabis, Koray Kavukcuoglu, Clement Farabet, Elena Buchatskaya, Sebastian Borgeaud, Noah Fiedel, Armand Joulin, Kathleen Kenealy, Robert Dadashi, and Alek Andreev. Gemma 2: Improving Open Language Models at a Practical Size, 2024b. URL https://arxiv.org/abs/2408.00118.

Jonathan Uesato, Rishabh Kumar, Rachit Shah, Michael Janner, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-Instruct: Aligning Language Models with Self-Generated Instructions, 2023. URL https://arxiv.org/abs/2212.10560.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Empowering Code Generation with OSS-Instruct, 2024. URL https://arxiv.org/abs/2312.02120.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. STaR: Self-Taught Reasoner Bootstrapping Reasoning With Reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

# A APPENDIX

# B SCALING SYNTHETIC DATA FOR CODE CONTEST AND LEETCODE

Our investigation into the impact of scaling synthetic code data reveals that increasing the amount of synthetic data does not always yield better performance.

For example, in the Code Contest dataset, a Gemma-2B model trained with 2,000 self-generated and filtered synthetic examples outperformed the same model trained on synthetic data from Gemma-9B. However, as our ablation studies indicate, when unpruned, synthetic data from the 9B model leads to better performance than 2B-generated data.

This suggests that smaller models can generate high-quality data that, when validated effectively, can match or even surpass the usefulness of data from larger models. This trend is seen across datasets, reinforcing the importance of careful data curation and filtering.

## B.1 DIMINISHING RETURNS IN SYNTHETIC DATA SCALING

Pass@1 performance on Code Contest reveals an interesting trend: while synthetic data generated by Gemma-9B improves performance initially, further scaling results in stagnation or decline. At 4,000 and 6,000 synthetic examples, performance either levels off or worsens.

Similarly, introducing small amounts of human-generated training data negatively impacts Pass@1, possibly due to an out-of-distribution effect.

The LeetCode dataset shows a smaller performance gap between synthetic data generated by Gemma-2B and Gemma-9B. A model trained on 1,000 2B-generated examples achieved a Pass@1 of 14.65%, slightly outperforming a model trained on 9B-generated examples, which achieved 12.82%. Despite this, both models performed similarly on Pass@20 and Correct@100, again suggesting that smaller models can generate useful training data.

## B.2 IMPLICATIONS FOR SYNTHETIC DATA GENERATION STRATEGIES

These results demonstrate that simply increasing synthetic data volume does not guarantee better outcomes. Instead, there is a complex interplay between dataset size, data quality, and the model used for data generation.

Our findings highlight the need for targeted synthetic data generation strategies and reinforce the potential for smaller models to contribute meaningfully to model improvement.

| Model | Stage | Source | Data | Pass@1(%) | Pass@50(%) |
|---|---|---|---|---|---|
| **Gemma-2B (Code Contest)** | Baseline | - | - | 0.90 | 5.70 |
| | 1k-2B-synth | Gemma-2B | 1k | 0.84 | 7.14 |
| | 2k-2B-synth | Gemma-2B | 2k | 0.71 | 7.14 |
| | 4k-2B-synth | Gemma-2B | 4k | 0.56 | 7.85 |
| | 6k-2B-synth | Gemma-2B | 6k | 0.66 | 8.57 |
| **Human Data (Code Contest)** | 1k-Human | Trainset | 1k | 0.60 | 6.43 |
| | 2k-Human | Trainset | 2k | 0.41 | 7.10 |
| | 4k-Human | Trainset | 4k | 0.24 | 5.71 |
| | 6k-Human | Trainset | 6k | 0.29 | 5.00 |
| **Gemma-9B (Code Contest)** | Baseline | - | - | 5.20 | 15.00 |
| | 1k-9B-synth | Gemma-9B | 1k | 7.30 | 17.80 |
| | 2k-9B-synth | Gemma-9B | 2k | 7.10 | 18.50 |
| | 4k-9B-synth | Gemma-9B | 4k | 6.70 | 17.10 |

Table 10: **Code Contest Scaling Amount of Synthetic Data Performance.** Pass@1 and Pass@50 performance on different sizes of synthetic data.

## B.3 IMPACT OF MIXING REAL AND SYNTHETIC DATA

We also explored the impact of combining small amounts of real and synthetic data for training. Our findings indicate that mixed datasets performed well in terms of Pass@1 scores, particularly in smaller training sizes (e.g., `1k 500 each`). However, attempts to recursively refine models

| Model | Stage | Source | Data | Pass@1(%) | Pass@20(%) |
|---|---|---|---|---|---|
| **Gemma-2B (LeetCode)** | Baseline | - | - | 13.66 | 36.60 |
| | 1k-2B-synth | Gemma-2B | 1k | 14.65 | 33.66 |
| | 1k-9B-synth | Gemma-9B | 1k | 12.82 | 33.66 |
| **Gemma-9B (LeetCode)** | Baseline | - | - | 13.66 | 36.60 |
| | 1k-9B-synth | Gemma-9B | 1k | 14.65 | 33.66 |

Table 11: **LeetCode Scaling Amount of Synthetic Data Performance.** Pass@1 and Pass@20 performance with different sizes of synthetic data.

using mixed datasets consistently failed. A possible explanation is that the introduction of real data disrupts the recursive self-improvement process, making it more difficult for the model to generalize effectively from its own outputs.

| Model | Stage | Source | Data | Pass@1(%) | Pass@50(%) |
|---|---|---|---|---|---|
| **Gemma-2B** (Code Contest, Mixed) | 1k-500-each | Trainset/2B | 1k | 1.0 | 7.14 |
| | 2k-1k-each | Trainset/2B | 2k | 0.60 | 7.85 |
| | 4k-2k-each | Trainset/2B | 4k | 0.24 | 5.71 |
| **rec mixed-sft** | 1k both on 1k both | 1kboth | 1k | 0.87 | 7.14 |
| | 2B1k on base | Gemma-2B | 1k | 0.92 | 4.28 |

Table 12: **Mixed Dataset Ablations.** Evaluating model performance when fine-tuning on a mix of real and synthetic data.

### B.4 THE COT PROMPT USED

The following is the Chain-of-Thought (CoT) prompt used in our experiments.

**Mathematical Reasoning Prompt**

**You are an expert mathematician.**
You are provided with a math problem. Your task is to solve the problem step-by-step, clearly showing all relevant calculations and reasoning.

**Problem:** `"Bella bought stamps at the post office. Some of the stamps had a snowflake design, some had a truck design, and some had a rose design. Bella bought 11 snowflake stamps. She bought 9 more truck stamps than snowflake stamps, and 13 fewer rose stamps than truck stamps. How many stamps did Bella buy in all?"`

**Requirements:**

1. Provide a complete and correct solution in a markdown block.
2. Explain each step of the solution in detail.
3. Conclude with the final numerical answer on a new line in the format `#### [Answer]`, replacing `[Answer]` with the actual answer.

**Solution:**

### B.5 EXAMPLE REASONING TRACES

**Gemma 2B**

```
Solution: Here's a step-by-step solution to the problem:
**1. Calculate the number of truck stamps:**
Bella bought 9 more truck stamps than snowflake stamps, and she bought 11 snowflake stamps.
Therefore, she bought 11 + 9 = 20 truck stamps.
**2. Calculate the number of rose stamps:**
Bella bought 13 fewer rose stamps than truck stamps, and she bought 20 truck stamps.
Therefore, she bought 20 - 13 = 7 rose stamps.
**3. Calculate the total number of stamps:**
Bella bought 11 snowflake stamps + 20 truck stamps + 7 rose stamps = 38 stamps in total.
**#### 38**
```

### Gemma 9B

**Solution:** Here's the step-by-step solution to the problem:
1.**Find the number of truck stamps:**
* Bella bought 9 more truck stamps than snowflake stamps, which she bought 11 of.
* Therefore, she bought 11 + 9 = 20 truck stamps.
2. **Find the number of rose stamps:**
* Bella bought 13 fewer rose stamps than truck stamps, which she bought 20 of.
* Therefore, she bought 20 - 13 = 7 rose stamps.
3. **Calculate the total number of stamps:**
* To find the total number of stamps, add the number of each type of stamp: 11 snowflake stamps + 20 truck stamps + 7 rose stamps = 38 stamps.
#### 38

### LLaMA 70B

**Solution:** ### Step-by-step solution to the problem
To find the total number of stamps Bella bought, we first need to determine the number of stamps for each design. We are given the following information:
- The number of snowflake stamps = 11
- The number of truck stamps = 9 more than snowflake stamps
- The number of rose stamps = 13 fewer than truck stamps
### Calculating the number of truck stamps
Since Bella bought 9 more truck stamps than snowflake stamps, we calculate the number of truck stamps by adding 9 to the number of snowflake stamps.
Number of truck stamps = Number of snowflake stamps + 9
= 11 + 9
= 20
### Calculating the number of rose stamps
Given that Bella bought 13 fewer rose stamps than truck stamps, we calculate the number of rose stamps by subtracting 13 from the number of truck stamps.
Number of rose stamps = Number of truck stamps - 13
= 20 - 13
= 7
### Calculating the total number of stamps
To find the total number of stamps, we add the number of snowflake, truck, and rose stamps together.
Total number of stamps = Number of snowflake stamps + Number of truck stamps + Number of rose stamps
= 11 + 20 + 7
= 38
#### 38

LLaMA 1B

**Solution:**

```
### Step 1:  Interpret the Problem and Identify Given Information
### Step 2:  Set Up Equations Based on Given Information
From the problem, we can set up the following equations:
1.  `s = 11` (number of snowflake stamps)
2.  `t = s + 9` (number of truck stamps)
3.  `r = t - 13` (number of rose stamps)
### Step 3:  Express the Number of Rose Stamps in Terms of Snowflake Stamps and Truck Stamps
Now, we'll substitute the expressions for `t` and `r` into the third equation:
`s = 11`
`t = s + 9`
`r = (s + 9) - 13`
### Step 4:  Solve for the Number of Rose Stamps
Let's simplify the expression for `r`:
`r = (s + 9) - 13`
`r = s - 4`
So, the number of rose stamps in terms of the number of snowflake stamps is `r = s - 4`.
### Step 5:  Find the Total Number of Stamps
The total number of stamps is the sum of snowflake stamps, truck stamps, and rose stamps.  Therefore,
we can write an equation representing the total number of stamps:
`s + t + r = 11 + (11 + 9) + (s - 4)`
Simplify and solve for `s` (snowflake stamps):
`s + t + r = 11 + 20 + s - 4`
`s + t + r = 26 + s`
`s - s = 26`
`s = 26`
Since `s` represents the number of snowflake stamps and we know `s = 11`, the actual number of
snowflake stamps is indeed 11.  Let's confirm this with the other equations.
### Step 6:  Confirm the Number of Truck and Rose Stamps
Now, we'll find the number of truck stamps `t` and rose stamps `r` using the equations:
`t = s + 9`
`r = t - 13`
We found `s = 11`, so:
`t = 11 + 9`
`t = 20`
`r = 20 - 13`
`r = 7`
## Step 7:  Verify the Total Number of Stamps Calculations
To ensure the correctness of our solution:
`s + t + r = 11 + 20 + 7`
`s + t + r = 38`
Using the original formula `20 + 7 - 4 = 37`, this confirms our conclusion that `s = 11` is a perfect
and easy-to-explain solution that satisfies all given conditions.
### Step 8:  Conclusion with the Final Answer
We finally have the total number of stamps Bella bought in all - 11 snowflake stamps, 20 truck
stamps, and 7 rose stamps.
#### 38
```