Ability Boundary of Data in LLMs' Math Reasoning

Anonymous ACL submission

Abstract

Large language models (LLMs) are displaying emergent abilities for math reasoning tasks, and there is a growing attention on enhancing the ability of open-source LLMs through supervised fine-tuning (SFT). But systematic research on the relationship between supervised data and model performance is limited. In this paper, we explore the ability boundary of data in LLMs' math reasoning, aiming to aid in the effective utilization of data, focusing on both questions and reasoning paths. Firstly, we determine the ability boundary of reasoning paths augmentation by identifying these paths' minimal optimal set. Secondly, we validate that different abilities of the model can be cumulatively enhanced by **M**ix of **M**inimal **O**ptimal **S**ets of corresponding types of data, while our models **MMOS** achieve SOTA performance in horizontal comparisons among LLaMA-2 7B/13B based models. Besides, we point out GSM-HARD is not really hard and today's LLMs no longer lack numerical robustness. Also, we provide an Auto Problem Generator for robustness testing and educational applications. Our code and data are publicly available at `https://github.com/Anonymous`.

1 Introduction

In the context of significant emergent abilities demonstrated by Large Language Models (LLMs) (Wei et al., 2022a; OpenAI, 2023), the focus on math reasoning tasks, particularly Numerical QA and Math Word Problems (MWP) (Kushman et al., 2014; Upadhyay and Chang, 2017; Miao et al., 2020a; Xu et al., 2022), is paramount. The current approach to activate these abilities in LLMs involves carefully engineered prompting (Brown et al., 2020), in-context learning (ICL) (Chen et al., 2022b) or supervised fine-tuning (SFT).

Particularly due to computational costs and stability concerns (Yuan et al., 2023), there is growing attention on enhancing the abilities of open-source LLMs (Rozière et al., 2023) through SFT.

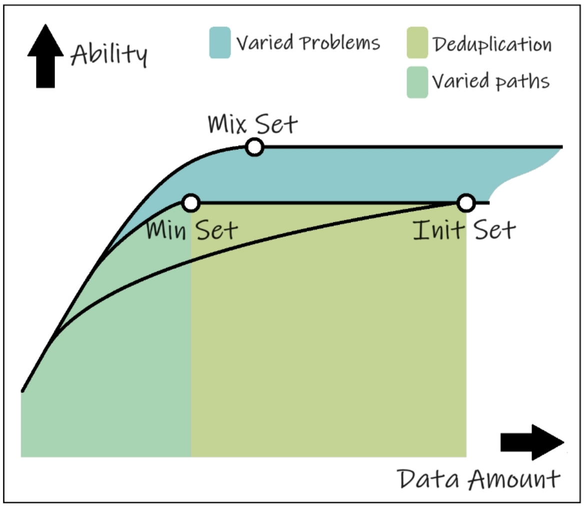

Figure 1: Conceptual figure of the ability boundary

Supervised data is crucial for SFT. Acquiring high-quality math problems from various sources typically need extra annotations (Lu et al., 2022), and creating supervised data in specific formats tailored for these models also involves additional costs. Easily implemented data augmentation methods, such as n-sampling for varied reasoning paths (Zhu et al., 2023) and bootstrap techniques for modifying problems (Yu et al., 2023), help expand supervised data.

However, systematic research specifically addressing the relationship between supervised data and model performance is limited. We only identify two recent studies: one by Yuan et al. (2023), which suggests a log-linear relationship between the amount of supervised data and model performance, and another by Li et al. (2023), finding that query and response augmentation cannot help with Out-of-Domain (OOD) math reasoning ability.

Our research focuses on exploring the ability boundary of data in LLMs' math reasoning, as illustrated in Figure 1. This inquiry is inspired by two pivotal studies that explore the relationship between the ability boundary and the minimal model size: one by Eldan and Li (2023), which addresses the minimal requirements for language models to produce coherent English text, and another by Gunasekar et al. (2023), focusing on the impact of 'textbook quality' data in breaking existing scaling laws. These insights prompt us to consider how data influence the ability boundary.

Specifically, our primary objectives are twofold: firstly, to determine the ability boundary through response augmentation, and secondly, to expand this ability boundary by introducing varied problems. By exploring the ability boundary of data, our study provides a practical framework for optimizing data utilization.

Overall, the contributions of the article include the following points:

1. Providing varied, deduplicated and correct reasoning paths can improve math reasoning ability in In-Domain and Similar-Domain data.

2. The ability boundary of increasing reasoning paths is reached, that is, we identify the minimal optimal set, when the number of paths is similar to the number of distinct problem solutions.

3. Different abilities of the model can be cumulatively enhanced by mixing minimal optimal sets of corresponding types of data.

4. GSM-HARD is not really hard and the numerical robustness issue is no longer prevalent in today's LLMs. We also build a high-quality Auto Problem Generator for these numerical robustness tests and educational applications.

5. An overlapping dataset can continue to enhance the model's ability in the absence of corresponding data. And our model (MMOS) obtained by SFT using **M**ix of **M**inimal **O**ptimal **S**ets on LLaMA-2 7B/13B achieve SOTA performance.

2 Related Work

2.1 LLM for Math Reasoning

Prompt based methods aim to activate the emergent abilities without training (Gou et al., 2023). A significant breakthrough comes from Chain-of-thought prompting (CoT) (Wei et al., 2022b), which enhances the ability of LLMs to tackle complex reasoning by using explicit intermediate reasoning steps. The least-to-most prompting strategy (Zhou et al., 2023) deconstructs complex problems into a series of simpler sub-problems, which are then solved sequentially. Program of thoughts prompting (Chen et al., 2022a) and program-aided language models (Gao et al., 2023) address the limited numerical abilities of LLMs and utilize LLMs solely for understanding problems and generating programs, while offloading solving and computation to an external Python interpreter.

Decoding related methods focus on enhancing performance by replacing the greedy decoding strategy during the inference stage. (Wang et al., 2023b) samples a diverse set of reasoning paths and selects the most consistent answer, while (Xie et al., 2023) proposes a decoding algorithm that integrates self-evaluation guidance through the use of stochastic beam search.

Supervised Fine-tuning (SFT) based methods are designed to enhance the math reasoning abilities of open-source models such as LLaMA (Touvron et al., 2023a), LLaMA2 (Touvron et al., 2023b), and Code LLaMA (Rozière et al., 2023), while ensuring transparency. Current methods (Yu et al., 2023; Wang et al., 2023a) largely utilize various prompt-based approaches, employing GPT-4 (OpenAI, 2023) or other open-source models, to generate reasoning steps as training datasets based on original QA in various datasets like GSM8k (Cobbe et al., 2021), MATH (Hendrycks et al., 2021). These generated reasoning steps can either be in natural language (rationales) (Zelikman et al., 2022) or a combination with program (Yue et al., 2023; Gou et al., 2023).

We follow this framework as well, opting for advanced models ToRA (Gou et al., 2023) that combine programs and rationales, and apply rejection sampling (Yuan et al., 2023) to build initial data for training on the Code LLaMA (Rozière et al., 2023) series of models.

2.2 Supervised Data Augmentation

Response augmentation approaches (Luo et al., 2023; Gou et al., 2023) involve employing techniques such as nucleus sampling (top-p sampling) (Holtzman et al., 2020) and combining inferences from models of varying sizes, with the aim of enlarging the amount of generated reasoning steps. These methods generally adhere to an intuitive understanding (Ni et al., 2022) that fine-tuned models are prone to biases towards a limited set of reference solutions.

Query augmentation methods focus on modifying existing questions to generate new ones. Li et al. (2023) finds that the diversity and complexity of problems contribute positively to performance, and Yu et al. (2023) believes that bootstrapping questions can provide multiple perspectives of metaknowledge, crucial for covering more unseen scenarios and enabling stronger generalization. Earlier researches applied Named Entity Recognition or Regular Expression matching to build templates for augmenting questions (Li et al., 2022). Xu et al. (2022) focused on categorizing questions based on numerical abilities and designing numerical perturbations. These earlier methods restrict the styles of question variation but are more stable.

Beside these specific data augmentation methods, we are more endeavoring to explore the ability boundary of augmented data to obtain a practical framework for optimizing data utilization.

3 Ability Boundary of Reasoning Paths

3.1 Overview

In this section, we aim to determine the ability boundary through response augmentation. We hypothesize that a minimal set capable of maximizing math reasoning ability consists of varied, deduplicated and correct reasoning paths.

In following Section 3.2, we discuss about the datasets. In Section 3.3, we identify this minimal optimal set and determine the benefits of removing duplicates and keeping varied reasoning paths within a certain range. In Section 3.4, we employ a clustering method as a filter to further explore the boundary. In Section 3.5, we conduct an ablation experiment to assess the impact of ensuring the correctness of the reasoning paths.

3.2 Dataset Comparation

Six datasets are involved in this study. Detailed information about their origins, example analyses, and a preliminary estimation of their difficulty levels can be found in the Appendix A.

To better understand the problems' difference across these datasets, we visualize the hidden representations of problems using t-SNE. This visualization as Figure 2 reveals a notable separation in the distribution of problems from the GSM8K and MATH datasets into two distinct clusters. This divergence emphasizes the contrast in question styles: GSM8K being text-intensive, while MATH is more focused on math expressions.

For the experiments presented in this section, we exclusively use GSM8K without bootstrapping its questions. Consequently, GSM8K is categorized as our IND data. Conversely, the MATH dataset, with its significant stylistic and content differences, is classified as OOD data. Additionally, two other

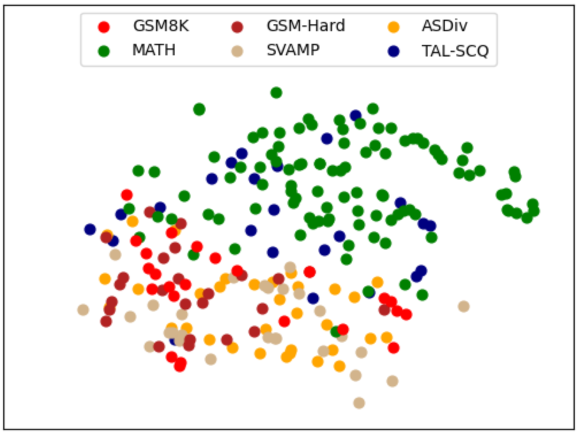

Figure 2: Visualization of query embedding distribution through t-SNE across six distinct datasets.

datasets, SVAMP and ASDiV, although different in origin from GSM8K, show similarities in both question types and spatial representations. Therefore, we consider these to be Similar-Domain Datasets. And we denote SVAMP and ASDiV as S&A in the subsequent analysis.

3.3 Identify the Minimal Optimal Set

To identify the minimal optimal set, we follow these steps: 1) Sample a sufficient number of correct reasoning paths to form initial set. 2) Implement a deduplication algorithm to obtain its deduplicated subset. 3) Conduct a statistical analysis on the upper limit of reasoning paths per question k with the subset data amount N. 4) Perform SFT on several subsets to analyze the impact of removing duplicates and keeping varied reasoning paths.

All detailed experiment settings are in C.

Initial set is formed by employing four pretrained models: ToRA-CODE 7B/13B/34B and ToRA 70B. For every question in the GSM8K dataset, these models sample 100 reasoning paths each with temperature 0.9. We then merge 400 reasoning paths and extract those whose code can be executed and have correct answers to obtain the initial training set E_{u400}.

Deduplication Algorithm 1 aim to extract the deduplicated subset D_{u400} from E_{u400} by codes which share the same calculation process.

We iterate all n data with following steps:

1) Extract the code block c_i from raw data d_i, including query q_i, completion a_i and source s_i .

2) Employ the Abstract Syntax Tree (AST) method to parse the code into the tree t_i.

3) Normalize the tree by replacing variable names v with lowercase letters and function names

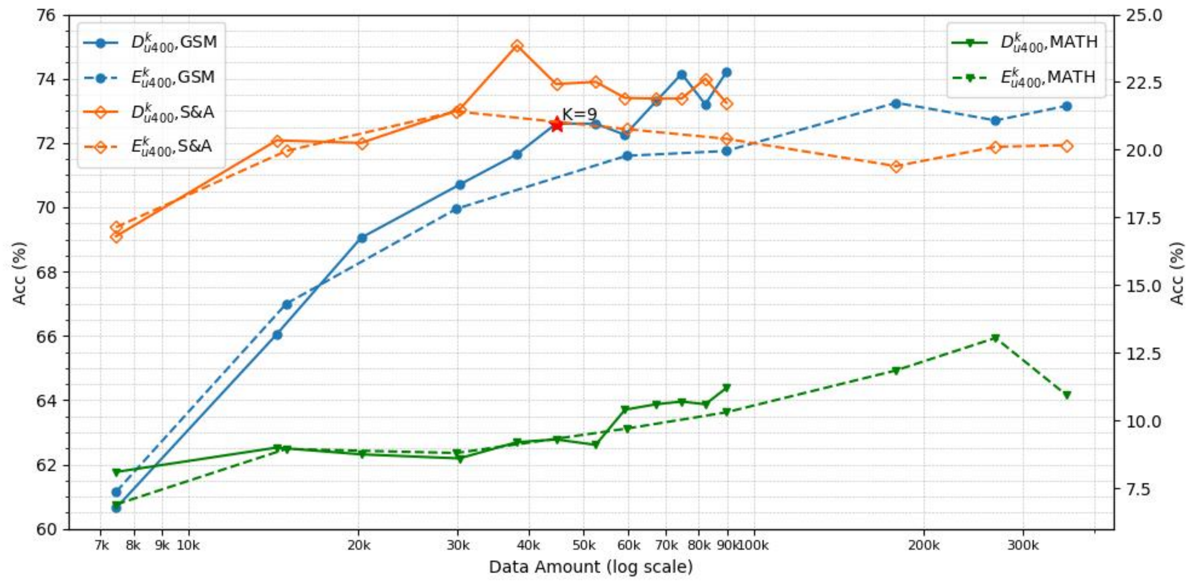

Figure 3: Comparison of test set accuracy on GSM8K, S&A and MATH for models after SFT on Code LLaMA 7B using series subsets of D^k_{u400} and E^k_{u400} with different data amount.

Algorithm 1 Deduplicate Data by Codes

Require: data d, extract $\xi(\cdot)$, recovery $\widetilde{\xi}(\cdot)$, astparse $\mathbb{P}(\cdot)$, astunparse $\widetilde{\mathbb{P}}(\cdot)$, deduplicate $\widetilde{\mathbb{D}}(\cdot)$

for $i = 1$ **to** n **do**
  $c_i \leftarrow \xi(d_i | q_i \oplus a_i \oplus s_i)$ ▷ Code Extraction
  $t_i \sim \mathbb{P}(c_i)$ ▷ Code Astparse
  $t_i^{'} \leftarrow \pi(t_i | v \oplus f)$ ▷ Code Substitution
  $c_i^{'} \sim \widetilde{\mathbb{P}}(t_i^{'})$ ▷ Code Astunparse
end for
$c^{'} \leftarrow \mathbb{D}(c^{'})$ ▷ Code Deduplication
$d^{'} \leftarrow \widetilde{\xi}(c^{'} \oplus q \oplus a \oplus s)$ ▷ Data Recovery

f with uppercase letters, resulting in t'_i.

4) Convert the normalized tree back into normalized code, denoted as $c_i^{'}$.

After completing the iteration, the normalized codes are duplicated through plain text matching. Finally deduplicated data $d^{'}$ is recovered with the deduplicated code, query, completion and source.

The k-N relation can be regarded as an estimation of the relationship between the number of reasoning paths per question k and the corresponding subset data amount N. This relation is obtained by implementing an upper limit on the reasoning paths per question in the initial set.

As shown in Appendix B, the k-N curve demonstrates a linear relationship on E_{u400} with a median of $k = 400$ and a mean of $k = 392.14$. In contrast, on D_{u400}, it exhibits a log-linear relationship with a median of 7 and a mean of 12.01. This indicates that the deduplication method is effective but still leaves room for improvement.

Comparative experiment includes two aspects. Firstly, to verify the effectiveness of adding varied paths, we conduct random selection of k paths for each question within D_{u400} to obtain twelve D^k_{u400} subsets with k ∈ {1,2,3,5,7,9,12,15,20,27,40,∞}, N ∈ {7.5,15,20,30,38,45,53,60,67,75,82,90}K.

Secondly, to better assess the impact of duplicate removal, we maintain a consistent order of magnitude in terms of data amount on E_{u400} and obtain E^k_{u400} with k∈{1,2,4,8,12,24,36,48} and N∈{7.5,15,30,60,90,180,270,360}K.

Evaluation & Conclusion. We conduct SFT on Code LLaMA 7B using a series of subsets D^k_{u400} and E^k_{u400}, and then inference on the test split of GSM8K, S&A, and MATH.

Results are shown in Figure 3. On the IND dataset GSM8K, as indicated by the blue solid line, the model's ability maintains a linear relationship with the logarithm of data amount before $k = 9$, $N = 45K$. In contrast, the blue dashed line representing the initial set data aligns with this trend only when k is small and duplicate paths are less likely to be selected. Beyond this point, further increasing the data amount sharply diminishes the marginal improvement in model ability. This suggests that enhancing the model's ability stems from adding varied reasoning paths, rather than merely

	k	5	7	9	15	27	-
$D_{u400}^{cluster,k}$	GSM8K	71.4(+0.7)	70.9(-0.7)	72.6(-1.2)	73.4(+0.8)	74(-0.1)	-
	S&A	73.4(+0.5)	73.4(-0.9)	73.1(-0.9)	74.2(+0.6)	73.4(+0.0)	-
	k	2	4	8	12	24	36
$E_{u400}^{cluster,k}$	GSM8K	67.6(+0.6)	70.5(+0.6)	72.1(+0.5)	74.0(+2.3)	73.2(+0.0)	73.5(+0.8)
	S&A	72.0(+0.3)	71.8(-1.1)	74.4(+2.0)	72.3(+0.2)	73.0(+2.0)	73.3(+1.4)

Table 1: Comparison of test set accuracy on GSM8K and S&A for models after SFT on Code LLaMA 7B using series subsets of D_{u400}^k and E_{u400}^k through clustering.

increasing the data amount.

We also observe that with the same data amount, beyond $N = 30K$, the performance on D_{u400} consistently surpasses that on E_{u400}. This reflects that removing duplicates can not only diminish the training duration but also enhance the model's ability.

On the Similar-Domain Datasets S&A, potentially due to the inherently easier nature of the questions, the models achieve high effectiveness even at k=1. The other conclusions are similar to those observed on GSM8K.

However, on the OOD dataset MATH, the models consistently exhibit weaker ability. This may be, as shown in Section 3.2, due to the differing types of questions presented in the dataset.

Thus far, we have essentially reached the conclusion that providing varied, deduplicated, and correct reasoning paths can improve math reasoning ability in both IND and Similar-Domain data.

Finally, we conduct a case study, as shown in Appendix D, where our example problem has 10 different solutions which is similar to the previously inflection point of k=9. Therefore, we consider $D_{u400}^{k=9}$ as the minimal optimal set. From this, we draw another conclusion: the ability boundary is reached, that is, we identify the minimal optimal set, when the number of reasoning paths is similar to the number of potential problem solutions.

3.4 Cluster as a Filter

Our deduplication algorithm, as an extension of a template method, is not flawless and can fail to eliminate similar paths. The example problem shown in Appendix D has only 10 distinct solutions. However, in D_{u400}, 43 paths are still retained. When we implement random selection to obtain $D_{u400}^{k=9}$, it only includes 6 distinct solutions.

We attempt to use clustering as a filter, replacing random selection, in order to ensure that the resulting $D_{u400}^{k=9}$ subset contains a greater number of distinct solutions. Specifically, we first obtain the embedding vectors of the codes. Then, we apply Latent Semantic Analysis (LSA) for dimensionality reduction, followed by k-means clustering. We extract and retain the central data points from these clusters. On the same example problem, the new $D_{u400}^{k=9}$ contains 7 distinct solutions.

Dataset	k	N	GSM8K	S&A
D_{u400}^k	9	44771	71.4	73.6
$D_{u400}^{total,k}$	9	46740	69.8(-1.6)	73.7(+0.1)
D_{u400}^k	∞	89530	74.2	73.3
$D_{u400}^{total,k}$	∞	126391	71.7(-2.5)	73.0(-0.3)

Table 2: Comparison of test set accuracy on GSM8K and S&A for models after SFT on Code LLaMA 7B using D_{u400} and D_{u400}^{total}.

In the comparative experiment, we replace random selection with clustering to obtain new subsets, $D_{u400}^{cluster,k}$ and $E_{u400}^{cluster,k}$. We then conduct SFT on Code LLaMA 7B using these subsets.

As shown in Table 1, the results on $E_{u400}^{cluster,k}$ exhibited a consistent improvement, suggesting that using clustering as a filter is viable. However, this is not the case for $D_{u400}^{cluster,k}$. We speculate that the remaining similar paths after deduplication have only a minor impact.

3.5 Correct Reasoning Ablation

While ensuring the correctness of paths is intuitively sound, we also observe that some methods, despite not guaranteeing correct answers for created problems, still yield reasonably good results. Therefore, we aim to ablate the effect of ensuring the correctness of paths.

During the acquisition of the initial set E_{u400}, we retain all data, including those with incorrect answers, resulting in $E_{u400}^{total,k}$. After deduplicating this set, we obtain $D_{u400}^{total,k}$. Subsequently, we generated subsets for $k = 9$ and $k = \infty$ through random selection from these sets and conduct comparative experiments with these subsets.

As illustrated in Table 2, on GSM8K, not filter-

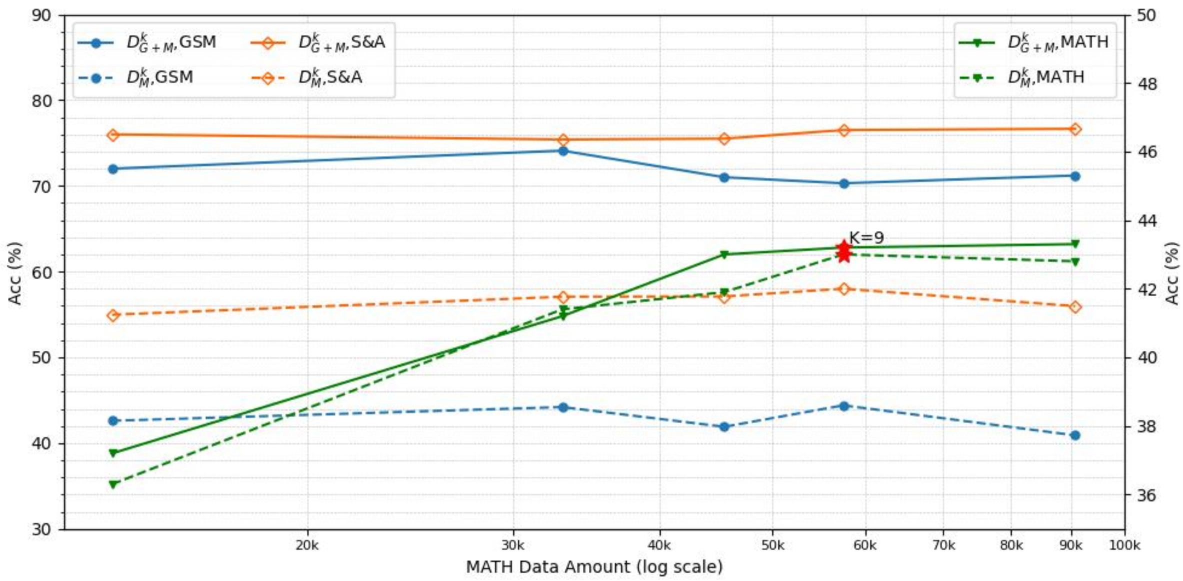

Figure 4: Comparison of test set accuracy on GSM8K, S&A and MATH for models after SFT on Code LLaMA 7B using series subsets of D^k_{G+M} and D^k_M with different MATH data amount.

ing out incorrect paths leads to a noticeable decline in performance. However, this effect is not observed on S&A, which could be attributed to the lower difficulty level of S&A.

4 Expand Boundary with Problems

4.1 Overview

In the previous section, we determine the ability boundary based on generating varied reasoning paths from the same problems. In this section, we consider expanding this ability boundary by introducing additional problems.

In Section 4.2, we examine whether the model's weak ability can be enhanced by providing corresponding data. Section 4.3 delves into the robustness of the model's numerical abilities and the issues present in a dataset, GSM-HARD. In Section 4.4, we develop an automated, high-accuracy problem generator for constructing numerically perturbed data, demonstrating its practical application value. Finally, in Section 4.5, we strive to achieve a state-of-the-art model and discuss the potential for further extending the model's existing ability.

4.2 Enhance Weak Ability

To address the issue of the weak ability of models trained with the minimal optimal set of GSM8K when applied to MATH, a straightforward and intuitive solution is to provide corresponding data.

Initially, following the same method described in Section 3.2, we obtain a series of deduplicated subsets D^k_M constructed using the MATH dataset and subsequently conduct SFT on them. And, as indicated by the green dashed line in Figure 4, we identify the minimal optimal set $D^{k=9}_M$ on MATH. As expected, compared to the models trained on D^k_{u400} originating from GSM8K, there is a significant improvement in ability on MATH, and the abilities on GSM8K and S&A, represented by the blue and yellow dashed lines, are weaker.

Subsequently, we merge the subsets D^k_M from the MATH dataset with the minimal optimal set of GSM8K $D^{k=9}_{u400}$, denoted as D^k_{G+M}. The experimental results on D^k_{G+M}, as shown by the various solid lines, indicate that compared to D^k_M, which provides the same amount of data from MATH, there is a slight improvement in performance on MATH, and a significant improvement on GSM8K and S&A. Additionally, the local optimum point for D^k_{G+M}, similar to D^k_M, is also achieved at k=9. Similarly, compared to $D^{k=9}_{u400}$, $D^{k=9}_{G+M}$ shows a slight decrease in performance on GSM8K, dropping from 72.6% to 70.3%, and a marginal decline on S&A, going from 73.6% to 76.5%. However, there is a significant improvement on MATH, with a rise from 10.4% to 43.2%. Overall, $D^{k=9}_{G+M}$ (102K) effectively combines the strengths of $D^{k=9}_{u400}$ (45K) and $D^{k=9}_M$ (57K), showcasing enhanced abilities on GSM8K, S&A, and MATH datasets.

We arrive at a fundamental conclusion: different abilities of the model can be cumulatively enhanced by mixing minimal optimal sets of corresponding types of data. This finding provides a simple yet effective method for enhancing the model's weak abilities by acquiring the corresponding datasets.

4.3 Is GSM-HARD Really Hard?

Another 'weak ability' of the $D_{G+M}^{k=9}$ model is demonstrated on GSM-HARD (54.8% vs 70.3% on GSM8K). This dataset is created by replacing the numbers in GSM8K with larger ones (Gao et al., 2023). Given that only the numerical values are altered, the distribution of problems in Figure 2 remains almost identical. Based on the conclusions from Section 3.3, such a significant discrepancy should not occur, whether we consider it as IND or Similar-Domain data. This leads us to questions: Is GSM-HARD really hard? Is the model's numerical robustness indeed weak?

The first source of discrepancy arises from the standards of ground truth. Due to the lack of meticulous design in the numerical values of the questions, some answers are not impractical, such as receiving answers with decimals when asking about quantities, or negative numbers when asking about the amount decreased. In practice, these initial calculation results should be rounded or converted to absolute values when providing answers, but GSM-HARD directly annotates these initial calculation results as the ground truth. We do not consider this to be indicative of a gap in ability. Therefore, using the standards of GSM-HARD, and evaluating based on the initial calculation results, the accuracy rate increases to 63.3(+8.5)%.

The second source of the discrepancy is due to errors in the ground truth annotation, stemming from an imperfect automated annotation process in GSM-HARD after modifying the problems. The corresponding values in the code are not updated in line with the changes in the numerical values in the problems, thus leading to execution with retained incorrect results as ground truth. We review the first 50 samples where the $D_{G+M}^{k=9}$ model make incorrect inferences and discover 25 errors in the ground truth annotations. We can estimate that the remaining gap, 70.3% - 63.3% = 7% < (1 - 63.3%) * (25/50) * 63.3% can be covered by these annotation errors.

Finally, we conjecture that GSM-HARD is not really hard and the numerical robustness issue is no longer prevalent in today's LLMs.

4.4 Auto Problem Generator

Considering this, developing an Auto Problem Generator capable of reliably producing data similar to GSM-HARD is meaningful. Such a generator can be used to test the numerical robustness of models. Additionally, it can also be utilized in educational applications to assess students' abilities.

Auto Problem Generator follows these steps:

1) Generate the deduplicated subset $D_{test,u400}$ from the seed dataset, the test split of GSM8K, following the method in Section 3.3.

2) For each question, extract the reasoning path with the highest repetition as the main path and separate the remaining path as the remain paths.

3) Extract numbers from questions using template matching and modify them with function $f(\cdot)$.

4) Modify the corresponding numbers in the code of the main path and execute it to obtain the answer A_{main}.

5) If the code execution fails or $A_{main} < 0$, modify the numbers again with 50 times limit.

6) Repeat step 4 on the remaining paths and obtain the answer set $A_{remains}$.

7) If all elements in $A_{remains}$ are identical to A_{main}, then we believe A_{main} is correct.

8) Combine the correct A_{main} with the modified questions to form the generated dataset P.

We apply the Distribution Perturbation (Xu et al., 2022) on numerical values with the following function $f(n)$ with μ=5, σ=1 and μ=1000, σ=300 to create datasets P_5 and P_{1000},

$$f(n) = n + \lfloor X \rfloor, X \sim \mathcal{N}(\mu, \sigma^2)$$

that $\mathcal{N}$ represents normal distribution. We manually review the first 100 QA pairs in P_5 and achieve a 98% accuracy rate, with only two questions having incorrectly annotated answers. A detailed analysis of these errors and their reasons can be found in the Appendix E.

Thus, we have successfully developed a high-quality Auto Problem Generator, which can be used for testing the numerical robustness of models as well as for educational application.

Numerical Robustness represents a model's consistent ability to handle different types of numerical values. Distribution Perturbation, as applied in GSM-HARD, P_5, and P_{1000}, is one such example. We evaluate P_5 and P_{1000} with the model trained on $D_{u400}^{k=9}$ with only GSM8K data. The experimental results show 73.8% on GSM8K, 72.1(-1.7)% on P_5 and 70.1(-3.7)% on P_{1000}.

Model	GSM8K	SVAMP	ASDiv	MATH	GSM8K	SVAMP	ASDiv	MATH
	7B				13B			
LLaMA-2	13.3	38.0	50.7	4.1	24.3	43.1	56.3	6.3
LLaMA-2 SFT	41.3	31.9	47.4	7.2	51.1	46.3	58.6	9.2
LLaMA-2 RFT	50.3	-	-	-	55.4	-	-	-
WizardMath	54.9	57.3	59.1	10.7	63.9	64.3	65.8	14.0
MAmmoTH	53.6	67.7	-	31.5	62.0	72.4	-	34.2
MetaMath	66.5	-	-	19.8	72.3	-	-	22.4
MathCoder-L	64.2	71.5	-	23.3	72.6	76.9	-	29.9
ToRA	68.8	68.2	73.9	40.1	72.7	72.9	77.2	43.0
MMOS	**69.9**	**73.4**	**76.8**	**40.2**	**74.8**	**77.0**	**80.0**	**43.2**

Table 3: Comparison of test set accuracy on 4 datasets for LLaMA-2 7B/13B based models.

Then, employing the approach used for creating P_{1000}, we produce P'_{1000} using the train split of GSM8K and include it in our training data. However, the results show tiny improvement, achieving 73.2% on GSM8K, 72.6(-0.6)% on P_5 and 70.4(-2.8)% on P_{1000}. Considering the results of both sets of experiments, since providing corresponding data does not enhance ability, we infer that the discrepancies in P_{1000} are more likely due to annotation issues caused by the inclusion of large numbers.

We also experiment with other numerical perturbation approaches including Language Perturbation and Noise Perturbation. Language Perturbation does not entail changes to the answers and simply involves converting numerical values into their English word representations. This has led to a slight improvement in the model's performance. Noise Perturbation introduces noise by adding decimal parts to the numerical values. The conclusions drawn from this method are similar to those from Distribution Perturbation.

Overall, we conclude that current LLMs no longer face significant issues with numerical robustness.

4.5 Expand Existing Ability

As mentioned in Section 4.2, the model $D^{k=9}_{G+M}$ obtained by SFT on Code LLaMA 7B achieve an accuracy of 70.3% on GSM8K, 76.5% on S&A, and 43.2% on MATH. Despite utilizing all available data from GSM8K and MATH, there remains a gap in accuracy compared to students who have undergone training. Therefore, we aim to explore how to further enhance the model's ability in the absence of corresponding data.

The dataset TAL-SCQ, as shown in Figure 2, displays query embeddings that overlap with GSM8K and MATH. We generate its minimal optimal set and merge it with $D^{k=9}_{G+M}$, denoted as D_{G+M+T}. Similarly, we conduct SFT on Code LLaMA 7B and achieve an accuracy of 73.9(+3.6)% on GSM8K, 77.5(+1.0)% on S&A, and 44.3(+1.1)% on MATH. We conclude that an overlapping dataset can continue to enhance the model's ability in the absence of corresponding data.

Due to the varying abilities of different base models, we conduct SFT on LLaMA-2 7B/13B for a horizontal comparison. The results, as shown in Table 3, indicate that our model (MMOS) using **M**ix of **M**inimal **O**ptimal **S**ets D_{G+M+T} achieves SOTA performance on all 4 datasets.

5 Conclusion

We explore the ability boundary of data in LLMs' math reasoning, with the goal of assisting people in effectively utilizing data. Firstly, we ascertain the ability boundary related to the augmentation of reasoning paths by identifying the minimal optimal set of these paths, with a focus on maximizing the data's potential. Secondly, we corroborate the premise that different abilities of the model can be collectively enhanced by amalgamating minimal optimal sets of data, each corresponding to specific types of information. Our models achieve SOTA performance in horizontal comparisons among LLaMA-2 7B/13B based models. Additionally, we uncover that LLMs currently do not exhibit a significant lack of numerical robustness. Moreover, we introduce an Auto Problem Generator, designed for testing the robustness of models and for use in educational applications.

Limitations

The limitations of our paper include the following aspects:

Datasets and Models. In our research, we use only three datasets to create a mix of minimal optimal sets as training data. However, we are uncertain whether the two conclusions drawn in Section 4 – that different abilities of the model can be cumulatively enhanced by mixing minimal optimal sets of corresponding types of data, and that an overlapping dataset can continue to enhance the model's ability in the absence of corresponding data – would still hold true with the introduction of more and larger datasets. Additionally, we are also unsure if these conclusions would apply to larger-scale models, such as the 70B model.

Sampling Bias. Our conclusions regarding the numerical robustness of the model, the GSM-HARD dataset and the Auto Problem Generator are based on our numerical analysis of accuracy and results from sample checks. This approach may introduce bias.

Ethical Statements

We claim from various aspects that our work is free of ethical risks:

1) Our research utilizes open-source models like LLaMA-2 and Code LLaMA and open datasets, and we strictly adhere to their licensing protocols.

2) Despite providing a new auto problem generator, its functionality is confined to numerical perturbation derived from open-source datasets. We endeavour to prevent the generation of illogical problems and the dissemination of inappropriate information resulting from numerical perturbations.

3) During the writing process, we used GPT4 to translate and correct grammatical errors, and the text was human-checked and rewritten to ensure that there were no ethical issues.

4) Our experiments are designed to be resource-efficient, requiring minimal compute time and power.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2022a. Program of Thoughts Prompting: Disentangling Computation from Reasoning for Numerical Reasoning Tasks.

Yanda Chen, Ruiqi Zhong, Sheng Zha, George Karypis, and He He. 2022b. Meta-learning via Language Model In-context Tuning.

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

Ronen Eldan and Yuanzhi Li. 2023. TinyStories: How Small Can Language Models Be and Still Speak Coherent English?

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. PAL: Program-aided Language Models.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. 2023. Textbooks Are All You Need.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring Mathematical Problem Solving With the MATH Dataset.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2020. The Curious Case of Neural Text Degeneration.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to Automatically Solve Algebra Word Problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281, Baltimore, Maryland. Association for Computational Linguistics.

Chengpeng Li, Zheng Yuan, Guanting Dong, Keming Lu, Jiancan Wu, Chuanqi Tan, Xiang Wang, and Chang Zhou. 2023. Query and response augmentation cannot help out-of-domain math reasoning generalization.

Jing Li, Aixin Sun, Jianglei Han, and Chenliang Li. 2022. A Survey on Deep Learning for Named Entity Recognition. *IEEE Transactions on Knowledge and Data Engineering*, 34(1):50–70.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2022. A survey of deep learning for mathematical reasoning.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. WizardMath: Empowering Mathematical Reasoning for Large Language Models via Reinforced Evol-Instruct.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020a. A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020b. A Diverse Corpus for Evaluating and Developing English Math Word Problem Solvers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 975–984, Online. Association for Computational Linguistics.

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Oleksandr Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. 2022. Learning Math Reasoning from Self-Sampled Correct and Partially-Correct Solutions.

OpenAI. 2023. GPT-4 Technical Report.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP Models really able to Solve Simple Math Word Problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. LLaMA: Open and Efficient Foundation Language Models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open Foundation and Fine-Tuned Chat Models.

Shyam Upadhyay and Ming-Wei Chang. 2017. Annotating Derivations: A New Evaluation Strategy and Dataset for Algebra Word Problems. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 494–504, Valencia, Spain. Association for Computational Linguistics.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2023a. MathCoder: Seamless Code Integration in LLMs for Enhanced Mathematical Reasoning.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023b. Self-Consistency Improves Chain of Thought Reasoning in Language Models.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent Abilities of Large Language Models.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H Chi, Quoc V Le, and Denny Zhou. 2022b. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models.

Yuxi Xie, Kenji Kawaguchi, Yiran Zhao, Xu Zhao, Min-Yen Kan, Junxian He, and Qizhe Xie. 2023. Self-evaluation guided beam search for reasoning.

Jialiang Xu, Mengyu Zhou, Xinyi He, Shi Han, and Dongmei Zhang. 2022. Towards Robust Numerical Question Answering: Diagnosing Numerical Capabilities of NLP Systems.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Keming Lu, Chuanqi Tan, Chang Zhou, and Jingren Zhou. 2023. Scaling Relationship on Learning Mathematical Reasoning with Large Language Models.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2023. MAmmoTH: Building Math Generalist Models through Hybrid Instruction Tuning.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D Goodman. 2022. Star: Bootstrapping reasoning with reasoning.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. 2023. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Yongfeng Huang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. 2023. Solving Math Word Problems via Cooperative Reasoning induced Language Models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4471–4485, Toronto, Canada. Association for Computational Linguistics.

A Datasets

In this paper, we have used 6 datasets, including: GSM8K (Cobbe et al., 2021), MATH (Hendrycks et al., 2021), GSM-HARD (Gao et al., 2023), SVAMP (Patel et al., 2021), ASDiv (Miao et al., 2020b) and TAL-SCQ5K.

In terms of difficulty, by rough estimation:

$$\text{SVAMP} \approx \text{ASDiV} < \text{GSM8K} \approx \text{GSM-HARD} < \text{TAL-SCQ5k} < \text{MATH}$$

with ASDiV as a diversed dataset covering problem types taught in elementary school; SVAMP as a structural modified version of a subset of ASDiv; GSM8K being an immense dataset covering grade school problems, with 2-8 steps; GSM-HARD built upon GSM8K, replacing numbers with less-common large numbers; TAL-SCQ5K containing primary, junior high and high school level mathematical topics; MATH full of challenging competition mathematics problems which requires a strong mathematical background to perform well on. Among which, MATH dataset and TAL-SCQ5K dataset further process notations of difficulty levels.

Dataset	Num	Example Q&A
GSM8K	Train: 7473 Test: 1319	**question:** In a dance class of 20 students, 20% enrolled in contemporary dance, 25% of the remaining enrolled in jazz dance, and the rest enrolled in hip-hop dance. What percentage of the entire students enrolled in hip-hop dance? **answer:** There are 20 x 20/100 = «20*20/100=4»4 students who enrolled in contemporary dance. So, 20 - 4 = «20-4=16»16 students are enrolled in either jazz or hip-hop dance. There are 16 x 25/100 = «16*25/100=4»4 students who enrolled in jazz dance. Hence, 16 - 4 = «16-4=12»12 students enrolled in hip-hop dance. This is 12/20 x 100% = 60% of the entire students. #### 60
MATH	Train: 7500 Test: 5000	**question:** How many vertical asymptotes does the graph of $ y=\frac {2}{x^2+x-6}$ have? **answer:** The denominator of the rational function factors into $x^2+x-6=(x-2)(x+3)$. Since the numerator is always nonzero, there is a vertical asymptote whenever the denominator is 0, which occurs for $x = 2$ and $x = -3$. Therefore, the graph has $\boxed{2}$ vertical asymptotes.
GSM-HARD	Test: 1319	**input:** A robe takes 2287720 bolts of blue fiber and half that much white fiber. How many bolts in total does it take? **code:** def solution(): """A robe takes 2 bolts of blue fiber and half that much white fiber. How many bolts in total does it take?""" blue_fiber = 2287720 white_fiber = blue_fiber / 2 total_fiber = blue_fiber + white_fiber result = total_fiber return result **target:** 3431580.0

Dataset	Num	Example Q&A
SVAMP	Test: 1000	**Body:** The Razorback t-shirt shop makes $ 78 dollars off each t-shirt sold. During the Arkansas game and the Texas tech game they sold a total of 186 t-shirts. If they sold 172 t-shirts during the Arkansas game **Question:** How much money did they make from selling the t-shirts during the Texas tech game? **Equation:** (78.0 * (186.0 - 172.0)) **Answer:** 1092.0
ASDiv	Test: 2215	**body:** Robert wants to practice goal kicks for soccer. He decides to have 98 kicks before going home from the park. He takes 43 kicks before taking a break to get a drink of water. He then takes another 36 kicks. **question:** How many more kicks does he need to make before he goes home? **equation:** 98-43-36=19 **answer:** 19 (kicks)
TAL-SCQ	5000	**problem:** If n is an even positive integer, the double factorial notation $n!!$ represents the product of all the even integers from 2 to n. For example, $8!!=2\\cdot4\\cdot6\\cdot8$. What is the units digit of the following sum? $2!!+4!!+6!!+\\cdot\\cdot\\cdot+2018!!+2020!!+2022!!$ **solution:** Answer: $$2$$

Table 4: Examples of datasets in their original format.

B Relationships of k & N

Figure 5 illustrates the relationships of the number of reasoning paths and the data amounts of the respective D_{u400} and E_{u400}.

We select multiple points from D_{u400} at regular intervals based on data amount. Simultaneously, we choose corresponding points from E_{u400} with similar data amounts to ensure consistence. The statistic that the relationships of the number of reasoning paths and the data amount is detailed in Table 5 and 6.

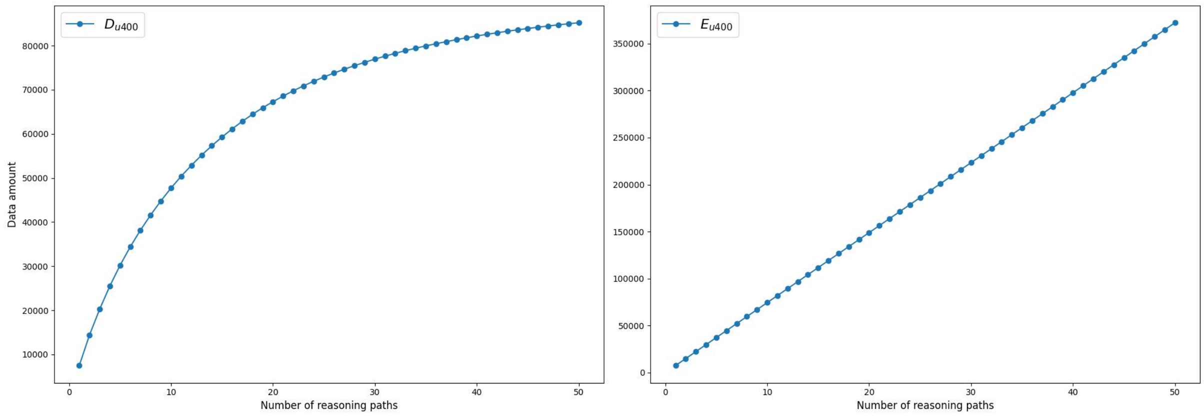

Figure 5: The relationships of k & N.

k	1	2	3	5	7	9	12	15	20	27	40	∞
N	7457	14344	20225	30179	38150	44771	52857	59261	67281	74643	82180	89530

Table 5: Extract subsets from relationships of k & N of D_{u400} for experiments.

k	1	2	4	8	12	24	36	48
N	7457	14911	29810	59603	89386	178707	268003	357295

Table 6: Extract subsets from relationships of k & N of E_{u400} for experiments.

C Detailed Experiment Setting

Generate Deduplicated Datasets

We spent 4 days generating both D_{u400}, D_M and the deduplicated dataset of TAL-SCQ in Section 3.3, 4.2 and 4.5 which is formed by employing four pre-trained models: ToRA-CODE 7B/13B/34B and ToRA 70B on the GSM8K, MATH and TAL-SCQ seperately, these models sample 100 reasoning paths each with temperature 0.9.

Training Models

We conducted SFT on Code LLaMA 7B using various deduplicated dataset and their subsets in Section 3.4, 3.5, 4.3 and 4.4. Addtionally we conducted SFT on LLaMA-2 7B/13B for a horizontal comparison in Section 4.5.

We used a learning rate of 2e-5 with a 3% warm-up period for 1 epoch and a global batch size of 128 on NVIDIA A100 40G GPUs. We trained all models with DeepSpeed ZeRO Stage3 and Flash-Attention 2.

Apart from validating the effectiveness of the deduplication algorithm, where the random selection process with seeds set to 0 and 42 and then averaging the inference results, all other training and inference processes used a seed of 0.

The training sessions were completed within 1 day, with an average training duration of approximately 5 hours. The average evaluation time is less than 10 minutes.

D Case Study: Actual Distinct Solutions

To validate the effectiveness of deduplication and using clustering as a filter, we conduct a case study focusing on the relationship of reasoning paths and their problems' actual distinct solutions.

In the deduplicated subset D_{u400} of the GSM8K dataset, we select the first question that has more than 15 reasoning paths, which has 43 reasoning paths for this problem in fact. Next, we utilize random selection and clustering as a filter to derive the subsets $D_{u400}^{k=15}$ and $D_{u400}^{cluster,k=15}$. We then separately analyze the 15 reasoning paths in these two subsets for the corresponding problem to categorize their actual distinct solutions on Table 7 and 8.

The question is formulated as follows:

Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?

Upon human analysis of this question, 10 distinct solutions have been summarized. These solutions are categorized based on whether the default daily salary is the same, whether the default working hours exceed the regular working hours, and how the total salary is calculated.

A Assuming that the daily wage is different, calculate the total wage by iterating over each day's wage and summing them in a loop.

B Assuming that the daily wage remains the same and that the working hours exceed the regular hours. Calculate the regular wage and the overtime wage for five days separately, then sum them up.

C Assuming that the daily wage remains the same and that the working hours exceed the regular hours. Calculate the regular hours worked and the overtime hours worked for five days separately, then sum the total wages.

D Assuming that the daily wage remains the same and that the working hours exceed the regular hours. Calculate daily wages, then sum them up.

E Assuming that the daily wage remains the same and that the Min and Max functions are used to avoid situations where the working hours are smaller than the regular hours. Calculate the regular wage and the overtime wage for five days separately, then sum them up.

F Assuming that the daily wage remains the same and that the Min and Max functions are used to avoid situations where the working hours are smaller than the regular hours. Calculate the regular hours worked and the overtime hours worked for five days separately, then sum the total wages.

G Assuming that the daily wage remains the same and that the Min and Max functions are used to avoid situations where the working hours are smaller than the regular hours. Calculate daily wages, then sum them up.

H Assuming that the daily wage remains the same and determine if the working hours exceed the regular hours. Calculate the regular wage and the overtime wage for five days separately, then sum them up.

I Assuming that the daily wage remains the same and determine if the working hours exceed the regular hours. Calculate the regular hours worked and the overtime hours worked for five days separately, then sum the total wages.

J Assuming that the daily wage remains the same and determine if the working hours exceed the regular hours. Calculate daily wages, then sum them up.

Table 7: Examples for Clustering as a Filter's Actual Solutions.

ID	Completion	Type
1	```python def total_salary(): """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?""" hourly_wage = 18.00 hours_per_shift = 10 num_days = 5 overtime_hours = hours_per_shift - 8 overtime_wage = hourly_wage + 0.5 * hourly_wage salary_per_day = hourly_wage * 8 + overtime_wage * overtime_hours total_salary = salary_per_day * num_days return total_salary total_salary = total_salary() print(total_salary) ``` ``` output 990.0 ``` Tina makes $\boxed{990.0}$ dollars.	J
2	```python def total_pay(): """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?""" hourly_wage = 18.00 hours_per_shift = 10 num_shifts = 5 overtime_hours = hours_per_shift - 8 overtime_wage = hourly_wage + 0.5 * hourly_wage pay_for_overtime = overtime_hours * overtime_wage * num_shifts pay_for_regular_hours = (8 * hourly_wage) * num_shifts total_pay = pay_for_overtime + pay_for_regular_hours return total_pay total_pay = total_pay() print(total_pay) ``` ``` output 990.0 ``` Tina makes $\boxed{990.0}$ dollars.	B
	``python def calculate_earnings(): """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""	

~~~
    hourly_wage = 18.00
    overtime_hours = 10 - 8 # hours worked in excess of 8 hours
    overtime_wage = hourly_wage + 0.5 * hourly_wage # overtime wage =
hourly wage + half hourly wage
3   daily_pay = 8 * hourly_wage + overtime_hours * overtime_wage    D
    total_pay = daily_pay * 5 # total pay for 5 days
    return total_pay
total_pay = calculate_earnings()
print(total_pay)
```
``` output
990.0
```
Tina makes 990.0 dollars.
~~~

~~~
```python
def total_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she
is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly
wage. If she works 10 hours every day for 5 days, how much money does she
make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    overtime_threshold = 8
    num_shifts = 5
    total_pay = 0
    for i in range(num_shifts):
        if hours_per_shift <= overtime_threshold:
4           total_pay += hourly_wage * hours_per_shift    A
        else:
            overtime_hours = hours_per_shift - overtime_threshold
            regular_hours = overtime_threshold
            total_pay += (hourly_wage * regular_hours) + (hourly_wage * over-
time_hours * 1.5)
    return total_pay
total_pay = total_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes 990.0 dollars.
~~~

~~~
```python
def calculate_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she
is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly
wage. If she works 10 hours every day for 5 days, how much money does she
make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_days = 5
    overtime_hours = hours_per_shift - 8
~~~

overtime_pay = hourly_wage + 0.5 * hourly_wage
5 regular_pay = hourly_wage * 8 D
total_pay = (overtime_hours * overtime_pay + regular_pay) * num_days
return total_pay
total_pay = calculate_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes 990.0 dollars.

```python
def total_earnings():
"""Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
hourly_wage = 18.00
hours_per_shift = 10
num_days = 5
overtime_hours = hours_per_shift - 8 if hours_per_shift > 8 else 0
6 earnings = (hourly_wage * 8) + (hourly_wage * overtime_hours * 1.5) J
total_earnings = earnings * num_days
return total_earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output
990.0
```
Tina makes 990.0 dollars.

```python
total_earnings():
"""Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
hourly_wage = 18.00
overtime_wage = hourly_wage + 0.5 * hourly_wage
hours_per_shift = 10
num_days = 5
7 total_hours = hours_per_shift * num_days C
overtime_hours = total_hours - 8 * num_days
earnings = (8 * num_days * hourly_wage) + (overtime_hours * overtime_wage)
return earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output
990.0

```
Tina makes 990.0 dollars.

```python
def total_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_day = 10
    num_days = 5
    overtime_hours = hours_per_day - 8 if hours_per_day > 8 else 0
8    overtime_pay = overtime_hours * (hourly_wage + 0.5 * hourly_wage)    J
    regular_pay = (8 * hourly_wage) + overtime_pay
    total_pay = regular_pay * num_days
    return total_pay
total_pay = total_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes 990.0 dollars.

```python
def total_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18
    hours_per_shift = 10
    num_days = 5
    overtime_hours = hours_per_shift - 8
9    overtime_wage = hourly_wage + 0.5 * hourly_wage    D
    earnings = (8 * hourly_wage + overtime_hours * overtime_wage) * num_days
    return earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output
990.0
```
Tina makes 990.0 dollars.

```python
def calculate_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18

    hours_per_shift = 10
    days_worked = 5
    overtime_hours = hours_per_shift - 8
    overtime_wage = hourly_wage + 0.5 * hourly_wage
    regular_pay = hourly_wage * 8
10    overtime_pay = overtime_wage * overtime_hours    D
    total_pay = (regular_pay + overtime_pay) * days_worked
    return total_pay
total_pay = calculate_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes $\boxed{990.0}$ dollars.

```python
def total_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    overtime_wage = hourly_wage + 0.5 * hourly_wage
    hours_per_shift = 10
    num_shifts = 5
11    earnings_straight_time = hourly_wage * 8 * num_shifts    B
    earnings_overtime = overtime_wage * (hours_per_shift - 8) * num_shifts
    total_earnings = earnings_straight_time + earnings_overtime
    return total_earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output
990.0
```
Tina makes $\boxed{990.0}$ dollars.

```python
def calculate_salary():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    overtime_wage = hourly_wage + 0.5 * hourly_wage
    hours_per_shift = 10
    days = 5
    regular_hours = min(hours_per_shift, 8)
12    overtime_hours = max(hours_per_shift - 8, 0)    E
    regular_pay = regular_hours * hourly_wage * days
    overtime_pay = overtime_hours * overtime_wage * days
    total_pay = regular_pay + overtime_pay

````
    return total_pay
total_pay = calculate_salary()
print(total_pay)
```
``` output
990.0
```
Tina makes 990.0 dollars.
````

---

````
```python
def total_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    overtime_wage = hourly_wage + 0.5 * hourly_wage
    hours_per_day = 10
    num_days = 5
    overtime_hours = max(hours_per_day - 8, 0)
13  regular_hours = min(hours_per_day, 8)
    total_pay = (regular_hours * hourly_wage + overtime_hours * overtime_wage) * num_days    G
    return total_pay
total_pay = total_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes 990.0 dollars.
````

---

````
```python
def total_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    overtime_hours = 10
    days = 5
    regular_hours = 8
    regular_earnings = hourly_wage * regular_hours * days
14  overtime_earnings = (hourly_wage + 0.5 * hourly_wage) * (overtime_hours - regular_hours) * days    B
    total_earnings = regular_earnings + overtime_earnings
    return total_earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output
990.0
```
````
````

	Tina makes $\boxed{990.0}$ dollars.	
15	```python def calculate_pay(): """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?""" hourly_wage = 18.00 hours_per_shift = 10 num_shifts = 5 overtime_hours = hours_per_shift - 8 overtime_wage = hourly_wage + 0.5 * hourly_wage pay = (8 * hourly_wage + overtime_hours * overtime_wage) * num_shifts return pay total_pay = calculate_pay() print(total_pay) ``` ``` output 990.0 ``` Tina makes $\boxed{990.0}$ dollars.	D

Table 8: Examples for Random Selection's Actual Solutions.

ID	Completion	Type
1	```python def total_salary(): """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?""" hourly_wage = 18.00 hours_per_shift = 10 num_days = 5 overtime_hours = hours_per_shift - 8 overtime_wage = hourly_wage + 0.5 * hourly_wage salary_per_day = hourly_wage * 8 + overtime_wage * overtime_hours total_salary = salary_per_day * num_days return total_salary total_salary = total_salary() print(total_salary) ``` ``` output 990.0 ``` Tina makes $\boxed{990.0}$ dollars.	D
	``python def total_pay():	

2

~~~
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    overtime_threshold = 8
    num_shifts = 5
    total_pay = 0
    for i in range(num_shifts):
        if hours_per_shift <= overtime_threshold:
            total_pay += hourly_wage * hours_per_shift
        else:
            overtime_hours = hours_per_shift - overtime_threshold
            regular_hours = overtime_threshold
            total_pay += (hourly_wage * regular_hours) +
            (hourly_wage * overtime_hours * 1.5)
    return total_pay
total_pay = total_pay()
print(total_pay)
```
``` output
990.0
```
~~~

Tina makes $\boxed{990.0}$ dollars.

A

3

~~~
```python
def calculate_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    overtime_wage = 0.5 * hourly_wage
    hours_per_shift = 10
    num_shifts = 5
    overtime_hours = max(hours_per_shift - 8, 0)
    regular_pay = hourly_wage * 8
    overtime_pay = (hours_per_shift - 8) * (hourly_wage + overtime_wage)
    total_pay = (regular_pay + overtime_pay) * num_shifts
    return total_pay
total_pay = calculate_pay()
print(total_pay)
```
``` output
990.0
```
~~~

Tina makes $\boxed{990.0}$ dollars.

G

~~~
```python
def calculate_pay():
~~~

4

```
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_shifts = 5
    overtime_hours = hours_per_shift - 8
    overtime_wage = hourly_wage + 0.5 * hourly_wage
    pay_for_shifts = (8 * hourly_wage) + (overtime_hours * overtime_wage)
    total_pay = num_shifts * pay_for_shifts
    return total_pay
total_pay = calculate_pay()
print(total_pay)
```

``` output
990.0
```

Tina makes $\boxed{990.0}$ dollars.

D

5

```python
def total_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_days = 5
    overtime_hours = hours_per_shift - 8
    if overtime_hours > 0:
        overtime_wage = hourly_wage + 0.5 * hourly_wage
        overtime_earnings = overtime_hours * overtime_wage * num_days
    else:
        overtime_earnings = 0
    regular_hours = 8
    regular_wage = hourly_wage
    regular_earnings = regular_hours * regular_wage * num_days
    total_earnings = regular_earnings + overtime_earnings
    return total_earnings
total_earnings = total_earnings()
print(total_earnings)
```

``` output
990.0
```

Tina makes $\boxed{990.0}$ dollars.

H

```python
def calculate_earnings():
```

```
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    overtime_wage = hourly_wage + 0.5 * hourly_wage
    hours_per_day = 10
    days_per_week = 5
    overtime_hours = hours_per_day - 8 # overtime is any hours over 8
6   base_earnings = hourly_wage * 8 * days_per_week                    B
    overtime_earnings = overtime_wage * overtime_hours * days_per_week
    total_earnings = base_earnings + overtime_earnings
    return total_earnings
total_earnings = calculate_earnings()
print(total_earnings)
```

``` output
990.0
```

Tina makes $\boxed{990.0}$ dollars.

```python
def total_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_day = 10
    num_days = 5
    overtime_hours = hours_per_day - 8 if hours_per_day > 8 else 0
7   overtime_pay = overtime_hours * (hourly_wage + 0.5 * hourly_wage)   J
    regular_pay = (8 * hourly_wage) + overtime_pay
    total_pay = regular_pay * num_days
    return total_pay
total_pay = total_pay()
print(total_pay)
```

``` output
990.0
```

Tina makes $\boxed{990.0}$ dollars.

```python
def calculate_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_days = 5
    overtime_hours = hours_per_shift - 8
```

    overtime_wage = hourly_wage + 0.5 * hourly_wage
8 earnings_regular = 8 * hourly_wage * num_days B
    earnings_overtime = overtime_hours * overtime_wage * num_days
    total_earnings = earnings_regular + earnings_overtime
    return total_earnings
total_earnings = calculate_earnings()
print(total_earnings)
```
``` output
990.0
```
Tina makes $\boxed{990.0}$ dollars.

```python
def calculate_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_shifts = 5
    overtime_hours = hours_per_shift - 8
9 overtime_wage = hourly_wage + 0.5 * hourly_wage D
    pay = (8 * hourly_wage + overtime_hours * overtime_wage) * num_shifts
    return pay
total_pay = calculate_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes $\boxed{990.0}$ dollars.

```python
def total_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    overtime_hours = hours_per_shift - 8
    num_shifts = 5
10 overtime_wage = hourly_wage + 0.5 * hourly_wage B
    regular_earnings = 8 * hourly_wage * num_shifts
    overtime_earnings = overtime_hours * overtime_wage * num_shifts
    total_earnings = regular_earnings + overtime_earnings
    return total_earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output

990.0
```

Tina makes $\boxed{990.0}$ dollars.

```python
def total_pay():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_shifts = 5
    overtime_hours = hours_per_shift - 8
    overtime_wage = hourly_wage + 0.5 * hourly_wage
11    regular_pay = hourly_wage * 8    D
    overtime_pay = overtime_wage * overtime_hours
    total_pay = (regular_pay + overtime_pay) * num_shifts
    return total_pay
total_pay = total_pay()
print(total_pay)
```

``` output
990.0
```

Tina makes $\boxed{990.0}$ dollars.

```python
def calculate_earnings():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
    hourly_wage = 18.00
    hours_per_shift = 10
    num_shifts = 5
    overtime_hours = hours_per_shift - 8
12    overtime_wage = hourly_wage + 0.5 * hourly_wage    D
    earnings = (8 * hourly_wage) + (overtime_hours * overtime_wage)
    return earnings * num_shifts
total_earnings = calculate_earnings()
print(total_earnings)
```

``` output
990.0
```

Tina makes $\boxed{990.0}$ dollars."

```python
def calculate_salary():
    """Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""

hourly_wage = 18.00
overtime_wage = hourly_wage + 0.5 * hourly_wage
hours_per_shift = 10
num_shifts = 5
13 overtime_hours = hours_per_shift - 8 if hours_per_shift > 8 else 0 J
salary = (8 * hourly_wage + overtime_hours * overtime_wage) * num_shifts
return salary
salary = calculate_salary()
print(salary)
```
``` output
990.0
```
Tina makes $\boxed{990.0}$ dollars.

```python
def calculate_pay():
"""Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
hourly_wage = 18.00
overtime_wage = hourly_wage + 0.5 * hourly_wage
hours_per_shift = 10
num_shifts = 5
overtime_hours = hours_per_shift - 8
14 regular_hours = 8 D
total_pay = (regular_hours * hourly_wage + overtime_hours * overtime_wage) * num_shifts
return total_pay

total_pay = calculate_pay()
print(total_pay)
```
``` output
990.0
```
Tina makes $\boxed{990.0}$ dollars.

```python
def total_earnings():
"""Tina makes $18.00 an hour. If she works more than 8 hours per shift, she is eligible for overtime, which is paid by your hourly wage + 1/2 your hourly wage. If she works 10 hours every day for 5 days, how much money does she make?"""
hourly_wage = 18.00
hours_per_day = 10
num_days = 5
overtime_hours = hours_per_day - 8 if hours_per_day > 8 else 0
15 overtime_wage = hourly_wage + 0.5 * hourly_wage H
regular_earnings = hourly_wage * 8 * num_days
overtime_earnings = overtime_wage * overtime_hours * num_days
total_earnings = regular_earnings + overtime_earnings

~~~
    return total_earnings
total_earnings = total_earnings()
print(total_earnings)
```
``` output
990.0
```
~~~
Tina makes $\boxed{990.0}$ dollars.

E Error Analysis: Numerical Perturb to create P_5

Table 9: Wrong case for P_5: Semantic Ambiguity

Problem:
Grandma Jones baked 5 apple pies for the fireman's luncheon. She cut each pie into 8 pieces and set the five pies out on the buffet table for the guests to serve themselves. At the end of the evening, after the guests had taken and eaten their pieces of pie, there were 14 pieces of pie remaining. How many pieces were taken by the guests?
Completion:
```python
def pie_pieces_taken():
    """Grandma Jones baked 5 apple pies for the fireman's luncheon. She cut each pie into 8 pieces and set the five pies out on the buffet table for the guests to serve themselves. At the end of the evening, after the guests had taken and eaten their pieces of pie, there were 14 pieces of pie remaining. How many pieces were taken by the guests?"""
    pies_initial = 5
    pieces_per_pie = 8
    total_pie_pieces = pies_initial * pieces_per_pie
    remaining_pie_pieces = 14
    taken_pie_pieces = total_pie_pieces - remaining_pie_pieces
    return taken_pie_pieces

taken_pie_pieces = pie_pieces_taken()
print(taken_pie_pieces)
```
```output
26
```
The guests took $\boxed{26}$ pieces of pie.

Perturbed Problem:
Grandma Jones baked 10 apple pies for the fireman's luncheon. She cut each pie into 13 pieces and set the five pies out on the buffet table for the guests to serve themselves. At the end of the evening, after the guests had taken and eaten their pieces of pie, there were 19 pieces of pie remaining. How many pieces were taken by the guests?
Perturbed Completion:
```python
def pie_pieces_taken():
    """Grandma Jones baked 10 apple pies for the fireman's luncheon. She cut each pie into 13 pieces and set the five pies out on the buffet table for the guests to serve themselves. At the end of the evening, after the guests had taken and eaten their pieces of pie, there were 19 pieces of pie remaining. How many pieces were taken by the guests?"""
```

```
    pies_initial = 10
    pieces_per_pie = 13
    total_pie_pieces = pies_initial * pieces_per_pie
    remaining_pie_pieces = 19
    taken_pie_pieces = total_pie_pieces - remaining_pie_pieces
    return taken_pie_pieces

taken_pie_pieces = pie_pieces_taken()
print(taken_pie_pieces)
```
```output
111
```
The guests took $\boxed{26}$ pieces of pie.

Table 10: Wrong case for P_5: Illogical Numerical Perturb

Problem:
A mechanic charges different rates to repair the tires of trucks and cars. For each truck tire that is repaired, the mechanic will charge $60 and for each car tire that is repaired, the mechanic will charge $40. On Thursday, the mechanic repairs 6 truck tires and 4 car tires. On Friday, the mechanic repairs 12 car tries and doesn't repair any truck tires. How much more revenue did the mechanic earn on the day with higher revenue?
Completion:
```python
def revenue]_difference():
    """A mechanic charges different rates to repair the tires of trucks and cars. For each truck tire that is repaired, the mechanic will charge $60 and for each car tire that is repaired, the mechanic will charge $40. On Thursday, the mechanic repairs 6 truck tires and 4 car tires. On Friday, the mechanic repairs 12 car tries and doesn't repair any truck tires. How much more revenue did the mechanic earn on the day with higher revenue?"""
    truck_tire_cost = 60
    car_tire_cost = 40
    truck_tires_repaired_thursday = 6
    car_tires_repaired_thursday = 4
    car_tires_repaired_friday = 12
    truck_tires_repaired_friday = 0
    revenue_thursday = (truck_tire_cost * truck_tires_repaired_thursday) + (car_tire_cost * car_tires_repaired_thursday)
    revenue_friday = (truck_tire_cost * truck_tires_repaired_friday) + (car_tire_cost * car_tires_repaired_friday)
revenue_difference = revenue_friday - revenue_thursday
    return revenue_difference
revenue_difference = revenue_difference()
print(revenue_difference)
```
```output
-40
```
The mechanic earned $\boxed{40}$ dollars less on Friday than on Thursday.

Perturbed Problem:
A mechanic charges different rates to repair the tires of trucks and cars. For each truck tire that is repaired, the mechanic will charge $65 and for each car tire that is repaired, the mechanic will charge $45. On Thursday, the mechanic repairs 11 truck tires and 9 car tires. On Friday, the mechanic repairs 17 car tries and doesn't repair any truck tires. How much more revenue did the mechanic earn on the day with higher revenue?
Perturbed Completion:
```python
def revenue]_difference():
    """A mechanic charges different rates to repair the tires of trucks and cars. For each truck tire that is repaired, the mechanic will charge $65 and for each car tire that is repaired, the mechanic will charge $45. On Thursday, the mechanic repairs 11 truck tires and 9 car tires. On Friday, the mechanic repairs 17 car tries and doesn't repair any truck tires. How much more revenue did the mechanic earn on the day with higher revenue?"""
    truck_tire_cost = 65
    car_tire_cost = 45
    truck_tires_repaired_thursday = 11
    car_tires_repaired_thursday = 9
    car_tires_repaired_friday = 17
    truck_tires_repaired_friday = 0
    revenue_thursday = (truck_tire_cost * truck_tires_repaired_thursday) + (car_tire_cost * car_tires_repaired_thursday)
    revenue_friday = (truck_tire_cost * truck_tires_repaired_friday) + (car_tire_cost * car_tires_repaired_friday)
revenue_difference = revenue_friday - revenue_thursday
    return revenue_difference
revenue_difference = revenue_difference()
print(revenue_difference)
```
```output
355
```
The mechanic earned $\boxed{-355}$ dollars less on Friday than on Thursday.