RevOrder: A Novel Method for Enhanced Arithmetic in Language Models

Anonymous ACL submission

Abstract

This paper presents RevOrder, a novel technique aimed at improving arithmetic operations in large language models (LLMs) by reversing the output digits in addition, subtraction, and n-digit by 1-digit (nD by 1D) multiplication tasks. Our method significantly reduces the Count of Sequential Intermediate Digits (CSID) to $\mathcal{O}(1)$, a new metric we introduce to assess equation complexity. Through comprehensive testing, RevOrder not only achieves perfect accuracy in basic arithmetic operations but also substantially boosts LLM performance in division tasks, particularly with large numbers where traditional models struggle. Implementation of RevOrder is cost-effective for both training and inference phases. Moreover, applying RevOrder to fine-tune the LLaMA2-7B model on the GSM8K math task results in a considerable improvement, reducing equation calculation errors by 46% and increasing overall scores from 41.6 to 44.4. [1]

1 Introduction

Large language models (LLMs) have gained significant attention in recent years, excelling in natural language understanding and generation tasks (Zhao et al., 2023). Despite their advancements, the leading models like ChatGPT (OpenAI, 2022) and GPT-4 (OpenAI, 2023) struggle with basic arithmetic, particularly with large digits. The GPT-4 website service[2] addresses this by switching to external Python tools, as depicted in Fig. 1(a). This shift not only adds a cumbersome step but also leads to excessive token usage, significantly disrupting the language processing flow and efficiency.

Arithmetic reasoning has long focused on solving arithmetic problems with LMs (Lu et al., 2022). Typically, LMs generate the solutions step-by-step in a chain-of-thought (COT) manner, common in reasoning tasks (Wei et al., 2022b; Kojima et al., 2022; Zhou et al., 2022). For instance, Nye et al. (2021) used a 'Scratchpad' to generate intermediate steps, achieving high accuracy in 8D addition tasks, as shown in Fig. 1(b). Similar methods are applied for subtraction, multiplication, division, and other arithmetic operations (Liu and Low, 2023; Yang et al., 2023).

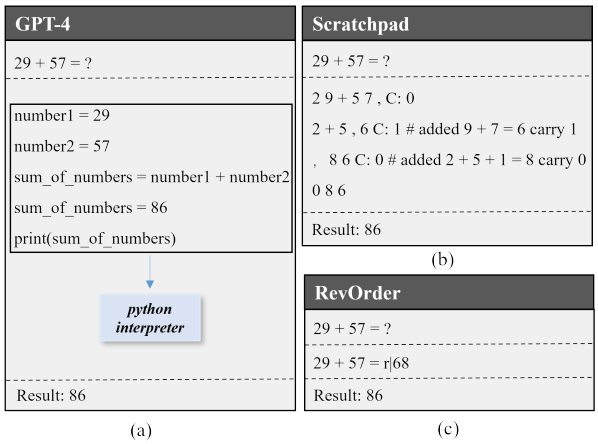

Figure 1: An illustration of performing addition using various methods. In the RevOrder method, the 'rl' symbol indicates that the subsequent digits are presented in reverse order.

However, practical application of arithmetic reasoning in LMs faces significant challenges. Firstly, LMs lack consistency in providing accurate results, and there is no established theory to measure equation complexity or to determine if an equation is within an LM's capabilities. For example, Liu and Low (2023) posited that addition is learnable by LLMs, but their experiments with large-digit addition contained minor errors. Secondly, current decomposition methods are token-intensive, making them more expensive than tool-based solutions during inference. Even for a simple 2D addition, the Scratchpad method (Nye et al., 2021), shown in Fig. 1(b), is not more token-efficient than Python tools (Fig. 1(a)).

[1] The data and code for this paper are available on Github.
[2] https://chat.openai.com/, 2024-1-26

To address these challenges, we introduce two novel concepts. First, we propose the Count of Sequential Digits (CSID) as an indicator to measure the difficulty of arithmetic equations. A larger CSID suggests more omitted reasoning steps, indicating a more complex equation. We demonstrate that the CSID complexity grows at $\mathcal{O}(n)$ for addition and subtraction, where n is the digit count. Empirical evidence suggests that advanced language models struggle considerably with high-CSID problems. This indicates a notable limitation: LLMs are unreliable in directly producing results for even basic arithmetic tasks, such as single additions or subtractions, when the digits involved are large.

Second, we propose RevOrder, a technique that reduces the CSID to a constant 1 for addition, subtraction, and nD by 1D multiplication operations. Illustrated in Fig. 1(c), RevOrder reverses the output order of addition. This approach aligns with the natural human reasoning sequence, where higher-order digits are resolved after the lower ones. Unlike previous methods such as Scratchpad (Nye et al., 2021), RevOrder requires virtually no additional tokens for these basic operations. Building upon these, we can construct more complex operations with significantly reduced token usage.

RevOrder, evaluated on the Big-bench arithmetic task (Srivastava et al., 2022) and an expanded set with larger digits, achieved 100% accuracy in addition, subtraction, multiplication, and low-digit division tasks, and nearly 100% in large-digit division, outperforming baseline methods. The experimental section highlights its training and inference efficiency. Finetuning LLAMA2 (Touvron et al., 2023) with RevOrder on the GSM8K dataset (Cobbe et al., 2021) significantly improved equation accuracy and overall scores (from 88.9% to 94.1%, and 41.6 to 44.4, respectively). These results affirm RevOrder's effectiveness and token economy in a range of arithmetic tasks, especially in addition and subtraction.

Section 2 reviews related work, Section 3 introduces the CSID metric, Section 4 details the RevOrder technique, Section 5 reports on experiments on arithmetic calculation, Section 6 discusses finetuning on GSM8K, and Section 7 concludes the paper.

2 Related Works

Arithmetic ability, a cornerstone of mathematics, has long served as a benchmark for assessing model capabilities, evolving from statistical methods (Hosseini et al., 2014) through machine learning techniques (Kushman et al., 2014), deep learning approaches (Wang et al., 2017) to LLM methods (Wei et al., 2022a).

While scaling laws for LLMs suggest that model capacity increases with model size, compute resources, and training data (Kaplan et al., 2020; Hoffmann et al., 2022), LLMs often struggle to directly generate arithmetic results. Consequently, step-by-step arithmetic reasoning methods have been developed. ScratchPad (Nye et al., 2021) introduces this concept for additions, achieving near-perfect accuracy on 8D addition tasks. This idea has since been expanded to more complex operations, such as multiplication and division (Liu and Low, 2023; Yang et al., 2023). These complex operations depend on the assumption that LLMs can efficiently perform basic operations such as addition and subtraction. Otherwise, token usage quickly becomes unsustainable. However, these so-called basic operations often fail to achieve 100% accuracy with large digits, making the more complex operations built upon them even more prone to error. Our CSID theory provides a framework to assess the complexity of equations, showing that LLMs' ability to perform basic operations diminishes as digit size grows. Conversely, RevOrder introduces an efficient method to keep equations' CSID low, ensuring their manageability within constrained token budgets.

Given the limitations and high token consumption of previous arithmetic reasoning methods, more pragmatic solutions have emerged, such as utilizing external tools or programming (Schick et al., 2023; Chen et al., 2022; Gao et al., 2023). RevOrder stands out by offering reliability and efficiency in addition and subtraction, positioning itself as a resource-saving alternative to these methods.

3 Sequential Intermediate Digits in Arithmetic Computation

Arithmetic reasoning in language models (LMs) is challenging, mainly due to the sequential prediction of digits. This complexity is exacerbated when contextual digits required for accurate predictions are not inferred from previous steps. For example, in addition, LMs may predict higher-order digits before lower-order ones, contradicting the logical computation order. This paper introduces a novel

metric to quantify and understand this complexity.

3.1 Definition of Sequential Intermediate Digits (SIDs)

A *Sequential Intermediate Digit* (SID) is a numeral crucial for the accurate prediction of the next digit in a sequence, yet not present in the preceding sequence. Within the framework of chain-of-thought reasoning, SIDs represent indispensable steps that, despite being missing, are vital for the computational process. Consequently, the Count of SIDs (CSIDs) is employed as a metric to assess the complexity of a generation step, with a higher CSID denoting a more demanding and intricate task. The CSID of an equation is thus defined as the maximum CSID required for generating each step of the result.

The primary types of SIDs include:

- Carry-over or borrow digits in addition and subtraction. For example, in $123 + 179 = 302$, the digit 3 in the hundreds place requires the carry-over from the tens and units places, resulting in a maximum CSID of 2.

- Digits from omitted reasoning steps, such as the intermediate sum 3 in $1 + 2 + 4 = 7$.

It is postulated that basic operations like 1D by 1D addition, subtraction, multiplication, division, counting, and copying do not require SIDs, as their straightforward nature falls within the capabilities of modern LMs. Directly generating results for complex operations, such as multi-digit multiplication and division, requires more SIDs due to the omitted steps for decomposing these into multiple basic operations.

Reducing an equation's CSIDs, thereby lowering its solving difficulty, can be achieved by expanding the equation step-by-step in a chain-of-thought manner. For instance, the CSID for the calculation $1+2+4 = 3+4 = 7$ is lower than for $1+2+4 = 7$ because the intermediate sum 3 is included in the reasoning process, effectively reducing the number of SIDs.

3.2 The CSIDs for Arithmetic Operations

In our CSID analysis of standard arithmetic operations, which is akin to analyzing space or time complexity in algorithms, we focus on the worst-case scenario. Consider two numbers $a = a_n a_{n-1} \ldots a_2 a_1$ and $b = b_m b_{m-1} \ldots b_2 b_1$, resulting in $c = c_t c_{t-1} \ldots c_2 c_1$, with $m \leq n$. When involving negative numbers, the minus sign '-' is also treated as a digit.

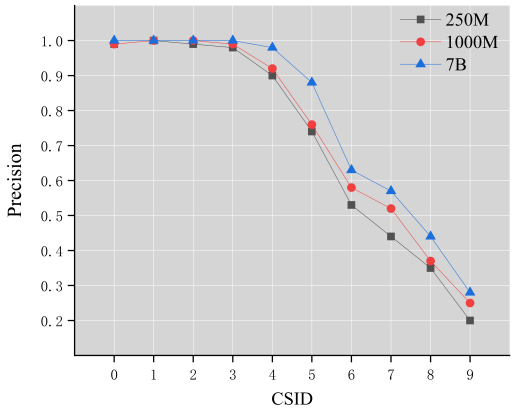

Figure 2: LLM performance on equations with varying CSIDs.

- In addition and subtraction, the computation sequence $a_n a_{n-1} \ldots a_2 a_1 \pm b_m b_{m-1} \ldots b_2 b_1 = c_t c_{t-1} \ldots c_2 c_1$ depends on each c_i involving a_i, b_i, and possibly c_{i-1} for carry-overs or borrows. Hence, the CSID for c_t includes all lower digits as SIDs, indicating a complexity of $\mathcal{O}(n)$.

- For multiplication and division, the CSIDs are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2 - m^2)$ respectively, as detailed in Appendix A.

3.3 LLM Performance on Large CSID Equations

We trained various models on arithmetic tasks involving 15D+15D calculations, maintaining identical hyper-parameters, training data, and training steps across all models to ensure a fair comparison. The test equations, strictly in 15D+15D format, were classified into various CSID levels according to the maximum number of continuous carry-over digits. The findings, as depicted in Fig. 2, demonstrate that:

- CSID effectively measures the complexity of arithmetic equations, where the performance consistently declines with increasing CSIDs.

- Larger models exhibit improved performance on equations with higher CSIDs.

- The benefit of increasing model size diminishes on high CSID equations. For instance, a 7B model shows more significant improvement on equations with CSIDs of 4 and 5

than on those with 6-9. This trend suggests that even advanced LLMs, like GPT-4, encounter difficulties with large digit addition tasks. Given that CSIDs have a complexity of at least $\mathcal{O}(n)$, arithmetic problems quickly surpass the capacity of LLMs when dealing with large digits. Therefore, **LLMs cannot serve as reliable calculators for immediate result generation in complex arithmetic tasks**.

4 RevOrder: Reversing the Order of Output Digits

We introduce RevOrder, an innovative technique devised to maintain low CSID in equations, thereby ensuring their solvability by LMs. Additionally, RevOrder is designed to minimize token usage, enhancing overall efficiency.

4.1 Addition and Subtraction

For addition and subtraction, we reverse the output digits' order:

$$\begin{aligned} a \pm b &= r|c_1 c_2 \dots c_t \\ &= c_t \dots c_2 c_1 \end{aligned}$$

Here, $r|$ is a special token indicating that the followed digits are in a reversed order. To generate each c_i in $r|c_1 c_2 \dots c_t$, only a_i, b_i, and at most a SID for the carry-over or borrow number from c_{i-1} are required. Thus, both addition and subtraction only consume at most 1 SID regardless of number length. Therefore, the complexity of CSID drop to $\mathcal{O}(1)$ from $\mathcal{O}(n)$, by applying RevOrder.

The cost of RevOrder for addition and subtraction is quite cheap during both training and inference. In training, RevOrder simply reverses the result digit orders. During inference, almost no additional tokens are required since the recovery of the result sequence can be done with rules.

4.2 Multiplication and Division

More complex operations like multiplication and division, can be decomposed to basic operations.

4.2.1 Multiplication

Firstly, consider the simplest form of multiplication, nD by 1D, e.g, 12*7=r|48, which consistently requires only 1 SID. This efficiency originates from the definition that 1D by 1D multiplication does not incur any SIDs, with the only one SID being the carry-over number in the addition.

Next, let's examine a more general multiplication example.

$$\begin{aligned} & 12 \times 4567 \\ =& 12 \times 4000 + 12 \times 500 + 12 \times 60 + 12 \times 7 \quad (1) \\ =& r|00084 + r|0006 + r|027 + r|48 \quad (2) \\ =& (r|00084 + r|0006) + (r|027 + r|48) \quad (3) \\ =& r|00045 + r|408 \quad (4) \\ =& r|40845 \\ =& 54804 \end{aligned}$$

First, decompose the multiplication as shown in Eqn. (1), which does not require any SIDs (require only count and copy operations that does not use SID in our definition). Second, output the results of each sub-multiplication in reverse order, as demonstrated in Eqn. (2). The zeros in these results can be efficiently generated through a copy operation from previous sequences. The nD by 1D multiplication in reverse order has a CSID of 1. Finally, iteratively combine the adjacent addition results until the final outcome is achieved, as illustrated in Eqn. (3) and (4). As each addition operation involves only two numbers, the CSID remains constant at 1 throughout the process.

In conclusion, the CSID in this multiplication process never exceeds 1, with a complexity of $\mathcal{O}(1)$.

4.2.2 Division

Consider the division $948 \div 12 = 79$:

$$\begin{aligned} & 948 \div 12 \\ =& 7 \text{ Rem } (948 - 12 \times 70) \quad (5) \\ =& 7 \text{ Rem } (948 - r|048) \\ =& 7 \text{ Rem } r|801 \\ =& 79 \text{ Rem } (r|801 - 12 * 9) \quad (6) \\ =& 79 \text{ Rem } (r|801 - r|801) \\ =& 79 \text{ Rem } (0) \\ =& 79 \end{aligned}$$

Utilizing traditional long division alongside RevOrder, the CSID typically remains at 1, with the exception of quotient estimation, as noted in Eqn. (5) and Eqn. (6). Since the CSID analysis here is similar to that of multiplication, we omit it for brevity, . However, it's important to note that quotient estimation often involves heuristic guesswork, making precise CSID measurement challenging. In

practice, we observed instances where the language model incorrectly estimated the quotient. To address this challenge, we implemented a rollback mechanism. If an incorrect quotient is detected, as illustrated in Eqn. (7), we insert a symbol 'W' after the line. This serves as a signal to adjust the process and re-estimate the quotient, as demonstrated in Eqn. (8). This method ensures more accurate quotient estimations in the long division process. In practice, a proportion of rollback scenarios are included in training to enhance the model's capability to correct such errors.

$$
\begin{aligned}
& 948 \div 12 \\
=& 8 \text{ Rem } (948 - 12 \times 80) \\
=& 8 \text{ Rem } (948 - r|069) \\
=& 8 \text{ Rem } (-r|21)W \qquad (7) \\
=& 7 \text{ Rem } (948 - 12 \times 70) \qquad (8) \\
& \ldots
\end{aligned}
$$

However, the quotient estimation in division is inherently unpredictable, rendering the CSID of this operation less controllable. Consequently, unlike other arithmetic operations, the CSID for division cannot be consistently maintained at $\mathcal{O}(1)$. This limitation makes division with RevOrder less robust compared to addition, subtraction, and multiplication, as will be evidenced in our experimental results.

4.3 Towards More Compact Forms

To reduce token usage, we propose compact forms while maintaining CSID unchangeable.

For the multiplication example, it can be succinctly rewritten as: '12×4567 = 12×4000 + 12×500 + 12×60+ 12×7=r|00084 + r|0006 + r|027 + r|48 = r|00045 + r|408 = r|40845 = 54804'.

Similarly, the division example can be condensed to: '948÷12 = 7R - (12×70)(r|048)(r|801) # 9R - (12×9)(r|801)(0) = 79', where R denotes REM and # denotes a new quotient estimation.

Two principles guide these simplifications: 1. Maintaining CSID: No digits essential for generating subsequent tokens are removed, ensuring the CSID remains unchanged. 2. Eliminating Redundancy: Duplicated digits are removed, but care is taken to avoid introducing ambiguities that might confuse the LM.

5 Experiments on Arithmetic Problems

In this section, we aim to address two key research questions (RQs):

- RQ1: Does RevOrder enable a language model to function as a reliable calculator? (Section 5.2)
- RQ2: Is RevOrder a cost-effective solution for practical using? (Section 5.4)

5.1 Setup

5.1.1 Dataset

Our training dataset is synthetically generated using a Python script, with each sample being an equation formatted with RevOrder, e.g., '123+46=r|961'. The dataset comprises positive integers, except in subtraction where negative numbers may result. Each division equation is assigned a probability of 0.5 to be selected for generating a rollback version. This involves intentionally misestimating a quotient step by a number ± 1, followed by a correction through the rollback process to the accurate estimation. The detailed of the training data is shown in Appendix B.

5.1.2 Training and evaluation protocol

We train a model named RevOrder-1B, which has 1.1 billion parameters. This model is trained on the TinyLLaMA 1.1B framework (Zhang et al., 2024), utilizing their released finetuning script. Specifically, the learning rate is set to 1e-4 for first 2 epochs and 1e-5 for the last epoch. The batch size is 500.

For evaluation, we employ the BIG-bench Arithmetic sub-task (Srivastava et al., 2022) and additional challenging tasks proposed in the GOAT-7B paper (Liu and Low, 2023). Each task has 1000 equations. We meticulously ensure that there is no overlap between the evaluation datasets and our training dataset, except for unavoidable overlaps in small digits tasks. The evaluation metric is exact match precision.

5.1.3 Baselines

As baselines, we compare against three methods:

- GOAT-7B (Liu and Low, 2023): This model, finetuned with 1 million instruction data on LLAMA-7B (Touvron et al., 2023), decomposes multiplication and division similarly to our approach. However, it relies on direct result generation for subtraction and addition.
- MathGLM-2B (Yang et al., 2023): Finetuned on the GLM-2B model for various arithmetic tasks, MATHGLM-2B claims that extensive

Correct result: 939007

429661432990/457570=

9R -9*45757000000(r|000000318114)(r|09923484871)#

3R -3*4575700000(r|00000172731(r|0992331214)#

8R -8*457570000(r|0000650663)(r|099277064)#

9R -9*45757000(r|000318114)(r|09995984)W#

0R -0*45757000(r|0)(r|09927706)#

9R -9*4575700(r|00318116)(r|013804-)W#

...

=93808325498165

Figure 3: An error example of division by RevOrder.

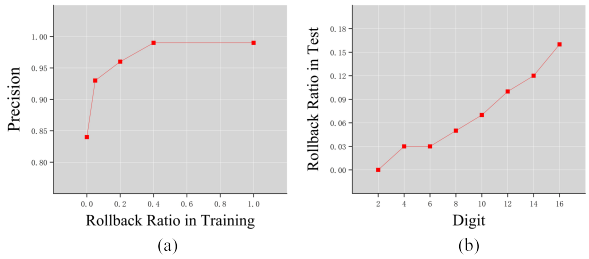

Figure 4: Analysis of the rollback ratio in division. (a) Test precision vs. rollback ratio for $12D \div 6D$ division. (b) Probability of rollbacks during testing across different digit sizes.

step-by-step training data (1m-50m instances) enables GPT models to solve math problems without external calculators.

- GPT-4 (OpenAI, 2023): Currently one of the most powerful LMs, GPT-4's results are based on direct mathematical problem-solving, without auxiliary tools or equation decomposition.

5.2 Main Results (RQ1)

The results, as presented in Table 1, demonstrate several key findings. Firstly, RevOrder-1B proves to be a reliable method for addition, subtraction, multiplication, and low-digit division tasks, achieving 100% accuracy across all corresponding tasks. In contrast, the accuracy of all baseline methods decreases with the increase in digit size. Secondly, while RevOrder-1B shows slight imperfections in large-digit division tasks, it still significantly outperforms baseline models. For instance, RevOrder-1B attains a 99.4% accuracy on the challenging 12D ÷ 6D tasks, with an increasing of 10.1% than that of the best-performing baseline, GOAT-7B.

The major success of RevOrder in multiplication and division can be attributed to its precise execution of basic operations, including addition, subtraction, and nD-1D multiplication. While GOAT-7B and MathGLM-2B also decompose these operations into basic ones, minor errors in these fundamental steps are amplified in subsequent composite operations, leading to a rapid decline in accuracy with larger digits.

In summary, RevOrder emerges as an effective technique, enabling language models to perform exact arithmetic calculations in addition, subtraction, multiplication, and low-digit division tasks.

5.3 In-Depth Analysis on Division

Large-digit division represents the sole operation where RevOrder encounters notable difficulties, warranting additional focus.

Upon examining division errors case by case, we discovered that all errors stemmed from incorrect quotient estimations. Fig. 3 illustrates such an error, where RevOrder-1B erroneously estimated the 3rd quotient as 8 (marked in red) instead of 9, without triggering the 'W' symbol for a rollback. Consequently, this led to a series of nonsensical outputs. It's notable that when a constant CSID of 1 is maintained in all four arithmetic operations, no errors occur. Errors only arise during quotient estimation, where CSID is unmeasurable. These results validate our theory regarding CSID.

We also assessed the effectiveness of the rollback mechanism. Fig. 4(a) presents the test precision for $12D \div 6D$ division across varying rollback ratios. A stark precision decline to 0.84 is observed with no rollback (ratio = 0). Precision does not significantly improve when the ratio exceeds 0.4, though this is partly due to the high baseline precision of 0.99. Fig. 4(b) illustrates the frequency of rollbacks during testing, indicating a higher incidence of rollbacks with larger digits. This trend underscores the importance of the rollback technique, particularly as it compensates for the increased likelihood of errors in quotient estimation with larger numbers.

5.4 The Cost of RevOrder (RQ2)

5.4.1 Cost of Training

By maintaining a low CSID, RevOrder simplifies the learning process for arithmetic problems, thereby reducing the volume of training data required. Table 2 compares the number of training equations needed for various methods. Despite being a smaller model, RevOrder-1B achieves perfect precision with at most half the training equations compared to other methods. Recent studies indicate that larger models often require less training data for task mastery (Hoffmann et al., 2022; Xia et al.,

Task	BIG-bench					Extra Tasks		
ADD	1D	2D	3D	4D	5D	8D+8D	16D+8D	16D+16D
GPT-4	100	100	99.6	98.8	94.1	92.1	9.4	94.1
GOAT-7B	100	100	99.4	98.3	98.1	97.8	97.1	97.6
MathGLM-2B	100	100	100	100	99.4	-	-	-
RevOrder-1B	100	100	100	100	100	100	100	100
SUB	1D	2D	3D	4D	5D	8D-8D	16D-8D	16D-16D
GPT-4	100	100	99.2	98.9	92.4	70.5	10.6	59.6
GOAT-7B	100	100	99.7	98.6	98.4	96.8	95.8	96.3
MathGLM-2B	100	100	99.9	99.8	98.9	-	-	-
RevOrder-1B	100	100	100	100	100	100	100	100
MUL	1D	2D	3D	4D	5D	16D $\times$ 1D	8D $\times$ 4D	6D$\times$6D
GPT-4	100	99.4	30.3	5.3	0.0	61.5	0.0	0.0
GOAT-7B	100	100	97.8	96.9	96.7	99.7	88.1	96.8
MathGLM-2B	100	99.9	98.3	94.9	89.9	-	-	-
RevOrder-1B	100	100	100	100	100	100	100	100
DIV	1D	2D	3D	4D	5D	16D$\div$1D	6D$\div$3D	12D$\div$6D
GPT-4	100	100	94.5	90.9	53.4	54	6.4	0.0
GOAT-7B	100	100	99.5	99	96.5	99	94.1	89.3
MathGLM-2B	100	100	99.4	100	94.9	-	-	-
RevOrder-1B	100	100	100	100	100	99.2	100	99.4

Table 1: Performance comparison on various arithmetic tasks. The results of the baseline methods are taken from their original paper, while the result of GPT-4 is taken from Liu and Low (2023).

Model	# Equations	100% ACC
RevOrder-1B	0.5m	Yes
MathGLM-2B	1m-50m	No
GOAT-7B	1.7m	No

Table 2: Number of training equations for different methods. This table reports the dataset size required for RevOrder-1B to achieve 100% accuracy on all Big-bench arithmetic sub-tasks. # Equations denotes the number of training equations.

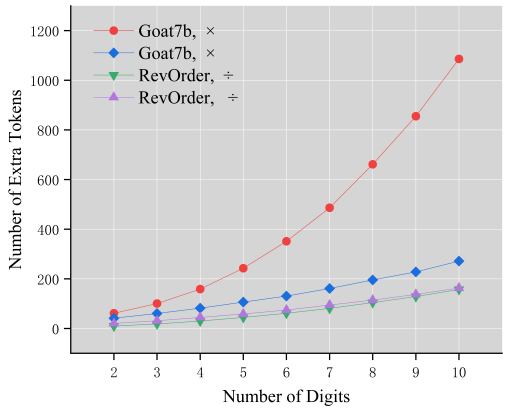

Figure 5: The number of extra tokens required for multiplication and division.

2022). Consequently, the training cost advantage of RevOrder is likely to be even more pronounced with larger LLMs.

5.4.2 Cost of Inference

The inference cost is assessed based on the number of additional tokens required for performing arithmetic calculations with RevOrder. We make two assumptions: 1) Each character (digit, symbol, etc.) is counted as one token, and 2) if the final result is output in reverse, the recovery process is handled by the tokenizer's decode function.

For addition and subtraction equations, only one extra token ('rl') is required. For multiplication and division equations, the number of extra tokens used is illustrated in Fig. 5. RevOrder is more token-efficient in both types of equations. Firstly, the compact form introduced in Section 3.3 significantly reduces the token requirement for division, approximately halving the number of extra tokens. Secondly, the iterative combination approach in multiplication, as exemplified in Eqn. (3), also notably reduces token usage in multiplication.

However, it must be acknowledged that for large-digit multiplication and division tasks, the token consumption of RevOrder increases polynomially and may eventually exceed the cost of using external tools. LLM service providers can set a threshold of digit number to decide between RevOrder

	Baseline	RevOrder
Score	41.6	44.4 (+2.8)
Equation Acc	88.9	94.1 (+5.2)
Acc of +	96.7	99.8 (+2.1)
Acc of -	97.0	99.6 (+2.6)
Acc of *	95.8	98.8 (+3)

Table 3: Fine-tuning results on GSM8K Dataset. This table compares the performance of models fine-tuned with the original GSM8K dataset (baseline) against those finetuned using the RevOrder-modified GSM8K dataset. The Score is measured by the correctness ratio of final results.

and tool-based solutions.

6 Additional Experiments on Math Word Problems

In this section, we delve into finetuning scenarios to address the research question:

- RQ3: How does applying RevOrder affect finetuning performance on mathematical tasks?

6.1 Setup

The experiment is conducted on GSM8K (Cobbe et al., 2021), a dataset of 8.5K high quality linguistically diverse grade school math word problems created by human problem writers. Our experiments utilize LLAMA2-7B (Touvron et al., 2023) as the foundational model. We modified the equations in the GSM8K training set to adopt the RevOrder format. This adaptation involved two major updates: Firstly, we presented the outcomes for addition, subtraction, and multiplication in reverse order. Secondly, polynomial equations were expanded and solved iteratively, in pairs. Noted that we did not decompose multi-digit multiplications and divisions, as these cases are infrequent in the GSM8K dataset. To further enhance the model's proficiency with RevOrder, we supplemented the training set with a small, synthetically generated dataset using a Python script. The comprehensive details of the dataset and the training parameters are provided in Appendix C.

6.2 Results

From the analysis, it is evident that RevOrder significantly reduces calculation errors, by 94% for addition, 87% for subtraction, and 46% for overall equation errors, thereby enhancing the final score. This improvement underscores the potential of seamlessly integrating RevOrder into fine-tuning processes to achieve substantial performance gains.

We also observe the errors, and find most of the errors are due to lack of enough training. Therefor, the model cannot well follow the instructions of RevOrder. Some examples are presented in Appendix C.

Hence, integrating RevOrder effectively into LMs is ideally conducted during the pretraining stage rather than the fine-tuning stage. The primary rationale is that excessive fine-tuning can lead to catastrophic forgetting, thereby impairing the general capabilities of LMs (Luo et al., 2023; Ramasesh et al., 2021).

7 Conclusion

In this paper, we introduce the CSID as a metric to evaluate the complexity of arithmetic equations and demonstrate that even large-scale LLMs struggle with high-CSID equations. We propose RevOrder, an innovative technique that ensures accurate arithmetic calculations by minimizing CSID, thereby enhancing precision while reducing both training and inference costs. Our experiments confirm that RevOrder significantly outperforms previous methods in terms of accuracy and efficiency.

For future work, we identify two possible paths: Firstly, developing token-efficient decomposition algorithms suitable for larger LLMs, which can handle higher CSIDs for complex arithmetic operations. Secondly, integrating RevOrder into LLMs' pretraining could enhance arithmetic capabilities more fundamentally than finetuning, reducing the risk of catastrophic forgetting and ensuring broader model proficiency.

8 Limitations

Firstly, RevOrder struggles with large-digit division, requiring significantly more training samples for this operation than others. An alternative algorithm that bypasses traditional quotient estimation may mitigate this issue.

Secondly, improvements in finetuning accuracy on the GSM8K dataset through RevOrder have not met our expectations. Increasing the dataset with arithmetic equations risks diminishing the LLM's overall performance. Finding an effective method to enhance arithmetic accuracy with minimal training data remains an unresolved challenge.

References

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

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

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Pal: Program-aided language models. In *International Conference on Machine Learning*, pages 10764–10799. PMLR.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 523–533.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. 2020. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

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.

Tiedong Liu and Bryan Kian Hsiang Low. 2023. Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks. *arXiv preprint arXiv:2305.14201*.

Pan Lu, Liang Qiu, Wenhao Yu, Sean Welleck, and Kai-Wei Chang. 2022. A survey of deep learning for mathematical reasoning. *arXiv preprint arXiv:2212.10535*.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*.

Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bieber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, et al. 2021. Show your work: Scratchpads for intermediate computation with language models. *arXiv preprint arXiv:2112.00114*.

OpenAI. 2022. Introducing chatgpt. *https://openai.com/blog/chatgpt*.

OpenAI. 2023. Gpt-4 technical report. *https://openai.com/research/gpt-4*.

Vinay Venkatesh Ramasesh, Aitor Lewkowycz, and Ethan Dyer. 2021. Effect of scale on catastrophic forgetting in neural networks. In *International Conference on Learning Representations*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. 2022. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*.

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

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 conference on empirical methods in natural language processing*, pages 845–854.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022b. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Mengzhou Xia, Mikel Artetxe, Chunting Zhou, Xi Victoria Lin, Ramakanth Pasunuru, Danqi Chen, Luke Zettlemoyer, and Ves Stoyanov. 2022. Training trajectories of language models across scales. *arXiv preprint arXiv:2212.09803*.

Zhen Yang, Ming Ding, Qingsong Lv, Zhihuan Jiang, Zehai He, Yuyi Guo, Jinfeng Bai, and Jie Tang. 2023. Gpt can solve mathematical problems without a calculator. *arXiv preprint arXiv:2309.03241*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024. Tinyllama: An open-source small language model.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, et al. 2022. Least-to-most prompting enables complex reasoning in large language models. *arXiv preprint arXiv:2205.10625*.

A The CSID Analysis of Multiplication and Division

This section extends the CSID analysis to nD by nD multiplication and nD by mD division, following the algorithmic approach outlined in Section 4.2 but excluding the RevOrder technique.

A.1 Multiplication

The decomposition of an nD by nD multiplication into n sub-multiplications, each an nD by 1D operation, serves as the initial step. This phase does not generate SIDs, as all required digits for $a \times b$ are immediately accessible.

Addressing these sub-multiplications yields up to $n^2 + n \times (n+1) = 2n^2 + n$ SIDs, with n^2 SIDs allocated for the sub-multiplications and $n \times (n+1)$ SIDs dedicated to storing the outcomes.

Aggregating the results of these sub-multiplications necessitates a maximum of $4n^2$ SIDs, with each addition consuming $4n$ SIDs, $2n$ for carry-overs and another $2n$ for storing the results.

Consequently, directly generating an nD by nD multiplication outcome requires a maximum of $6n^2 + n$ SIDs, indicating a complexity of $\mathcal{O}(n^2)$. This substantial complexity explains the difficulty models face with even 2D by 2D multiplications.

Decomposition methods, as applied in models like GOAT-7B and MathGLM-2B, reduce the CSID to $\mathcal{O}(n)$, by omitting intermediate results from the SID count, though carry-overs are still considered.

A.2 Division

For an nD by mD division, typically $n - m$ iterations are needed, each estimating a quotient digit.

Each iteration involves an nD by 1D multiplication and a subtraction, with the multiplication incurring 2m SIDs for result and carry-over digit storage, and the subtraction using up to 2n SIDs for result storage and borrow digits.

Thus, the total CSID for an nD by mD division reaches $(2m+2n)*(n-m) = 2n^2-2m^2$, amounting to a complexity of $\mathcal{O}(n^2 - m^2)$.

This estimation excludes the quotient estimation step's complexity, which could further complicate large number divisions, potentially surpassing the $\mathcal{O}(n^2 - m^2)$ complexity.

In models like GOAT-7B and MathGLM-2B, using decomposition methods keeps the CSID at $\mathcal{O}(n)$, with the subtraction's borrow digits being the primary complexity factor.

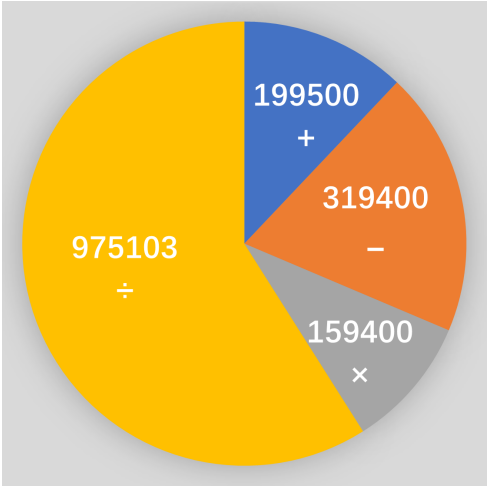

Figure 6: The distribution of the equations in training set.

B Training Data for Arithmetic Experiments

The training dataset comprises 1.7 million equations. For addition and subtraction tasks, equations involve numbers as large as 16D on both sides. Multiplication tasks are capped at 8D by 8D, supplemented by 16D by 1D equations to enhance generalization in the test set. Division tasks feature dividends up to 16D. Fig. 6 illustrates the distribution of these equations. The major training samples are division, since the quotient estimation steps require more training samples to achieve a high precision.

C Settings for Math Word Experiments

C.1 Training Data

Our approach involved two types of instructional data to train models on arithmetic tasks using RevOrder.

Firstly, we modified the original GSM8K dataset to reflect RevOrder formatting. An example of this adaptation is illustrated in Fig. 7.

Secondly, to further bolster the model's proficiency in RevOrder calculations, we compiled an additional enhancement dataset. A sample from this dataset is depicted in Fig. 8.

Given the limited size of the training data, the 7B model faced challenges in mastering the use of the reverse symbol r|. To address this, we introduced a notation where all numbers enclosed by @@, signify reverse order.

C.2 Training Details

The models were trained with a batch size of 32 and a learning rate of 5e-5, employing a warm-up ratio of 0.08 over 3 epochs. During each epoch, the model was exposed to both the additional datasets and the GSM8K datasets sequentially.

C.3 Equation Errors

Fig. 9 showcases representative errors encountered in the GSM8K test set, attributable to difficulties in adhering to RevOrder instructions. For instance, while the model successfully solved the second equation in reverse order, it faltered in performing the simple task of reversing the solution to arrive at the final result.

Instruction: “ Now it's time to put the reverse order rules into practice to solve some mathematical problems. Recall the key points from the examples we've previously gone through: 18+51=@96@=69, 69-51=@81@=18, 48*13=@426@=624, 48/4=12, 154+446+375=@006@+375=600+375=@579@=975, 64+784+645+78=@848@+@327@=848+723=@1751@=1571, 43+85+6+45+48=@821@+@15@+48=128+51+48=@971@+48=179+48=@1751@=1571,and 34*73*4=@2842@*4=2482*4=@8299@=9928. In these examples, the '@' symbols enclose the reverse order result.
Your task is to answer the following math questions using the same methodology.”

Question: “ Mitchell is trying to chew as many pieces of gum at once as he can. He has 8 packets of gum, There are 7 pieces in each. If he chews all the gum except for 2 pieces, how many pieces does he chew at once? ”

Response: “ The packets have 56 pieces in total because 8*7 = << 8*7=@65@=56 >>56
He chews 54 pieces because 56 - 2 = << 56-2=@45@=54 >>54#### 54 ”

Figure 7: A data sample from the GSM8K dataset formatted in RevOrder.

Instruction: “As a math aficionado, let's dive into the fascinating concept of reverse order calculations. This rule dictates: (1) Calculations should be performed in reverse order, starting with lower level operations followed by higher ones. For instance, in 35+44=@97@=79, we first add the lower digits 5 and 4, getting 9, then we add the higher digits 3 and 4, getting 7. The number '97' is framed within '@', signifying the reverse order calculation result, which we then reverse again to get '79’.
(2) For polynomials, we use the union method for easier operations.
For instance, 7+11+13=@81@+13=18+13=@13@=31,
63+68+44+25=@131@+@96@=131+69=@002@=200,and
15+46+35+2+4=@16@+@73@+4=61+37+4=@89@+4=98+4=@201@=102, calculations involving multiple additions should be done in pairs for accuracy.
(3) In a polynomial with both multiplication and division, to avoid inaccuracies due to endless decimals, we use the exchange method, performing multiplication before division.
For example, in 36/6*4=(36*4)/6=@441@/6=144/6=24, the exchange method is used.
Please familiarize yourself with these reverse rules. Here are a few additional examples for further practice. ”

Response: “<<10+7=@71@=17>>, <<562-231=@133@=331>>, <<100-7=@39@=93>>, <<63+55=@811@=118>>, <<20*5=@001@=100>>, <<54*3=@261@=162>>, <<200-86=@411@=114>>, <<27+26=@35@=53>>, <<40*4=@061@=160>>, <<16*8=@821@=128>>, << 46*2=@29@=92 >>, <<16+15=@13@=31>>, <<4953*7=@17643@=34671>>, << 76-28=@84@=48 >>, <<121+48=@961@=169>>, <<47*5=@532@=235>>, << 285+89=@473@=374 >>, <<2514-1983=@135@=531>>, <<43974+11978=@25955@=55952>>, <<29+41+26+6=@07@+@23@=70+32=@201@=102 >>”

Figure 8: A sample from the additional enhancement dataset for RevOrder calculations.

33.48+16.64+40.04=@20.05@+40.04=50.02+40.04=@60.09@=90.06 ×

256-392=@661-@=-166 ×

24*125=@0023@=3200 ×

162/7=23.142857 ×

Figure 9: Illustrative errors from the GSM8K test set encountered by the model trained with RevOrder.