RevOrder: A Novel Equation Format for Arithmetic Operations in Language Models

Anonymous ACL submission

Abstract

This paper proposes to understand arithmetic operations in Language Models (LM) by framing them as digit-based reasoning challenges. We introduce a metric called the Count of Sequential Intermediate Digits (CSID), which measures the complexity of arithmetic equations by counting the missing steps in digit reasoning. Our empirical findings suggest that increasing the model size does little to improve the handling of equations with high CSID values.

We propose RevOrder, a method that incorporates techniques such as reversing the output order, step-by-step decomposition, and rollback mechanisms to maintain a low CSID, thereby enhancing the solvability of arithmetic equations in LMs. RevOrder also introduces a more compact reasoning process, which reduces the token requirements without affecting the CSID, significantly enhancing token efficiency.

Comprehensive testing shows that RevOrder achieves perfect accuracy in operations such as addition, subtraction, and multiplication, and substantially improves performance in division tasks, especially with large numbers where traditional models falter. Additionally, applying RevOrder to fine-tune the LLaMA2-7B model on the GSM8K math task led to a significant reduction in equation calculation errors by 46% and increased the overall score from 41.6 to 44.4. [1]

1 Introduction

Arithmetic reasoning has long been a focus for improving the capabilities of Language Models (LMs) in solving arithmetic problems (Lu et al., 2022). A popular alternative involves generating solutions step-by-step in a chain-of-thought (COT) manner, which have been applied to a range of operations including subtraction, multiplication and division(Liu and Low (2023)). Interestingly, recent findings by Lee et al. (2023) have shown that simply reversing the output order of digits significantly enhances performance in addition, subtraction, and 2D multiplication, aligning the problem-solving approach more closely with human methods, which typically proceed from lower to higher digits.

In this paper, we conceptualize arithmetic problems as digit-based reasoning tasks, where each digit represents a step in the reasoning process. From this perspective, reversing the output digits effectively reorders these reasoning steps into a more logical sequence. This understanding bridges the reversing technique and COT solutions, aiming to reduce missing reasoning steps and simplify equation complexity.

We introduce a new metric, the Count of Sequential Intermediate Digit (CSID), to gauge the complexity of an equation. A higher CSID indicates more missing reasoning steps. Our empirical evidence suggests that simply enlarging LLMs does not substantially improve their performance on equations with high CSID values.

Guided by these insights, we propose RevOrder, a novel equation format designed to enhance the precision of arithmetic operations while minimizing the use of extra tokens. RevOrder fundamentally reverses the order of all output digits in intermediate steps, keeping the CSID low and ensuring that equations remain solvable by LMs. Figure 1 shows an example of mutiplication using different methods.

For division tasks, where the CSID for quotient estimation remains high with large digits, we introduce a 'Rollback' technique that enables LMs to detect and correct errors automatically. Additionally, we present a compact equation format that maintains the same CSID while eliminating unnecessary tokens, further enhancing the efficiency of LMs in arithmetic tasks.

RevOrder is evaluated on the Big-bench arith-

[1] The data and code can be found at GitHub Repository

Simple Reverse (Lee et al.(2023))

$12 × 18 = 612$

COT (Liu and Low(2023))

12 × 18
= 12 × 10 + 12 × 8
= 120 + 96
= 216

Reverse + COT (RevOrder)

12 × 18
= 12 × 10 + 12 × 8
= 021 + 69
= 612

Figure 1: An example of multiplication using different methods. Digits enclosed by $ indicate reversed orders. The simple reverse method (Lee et al., 2023) omits the total reasoning steps required for decomposition, thus simplifying the process. In contrast, the GOAT-7b model (Liu and Low, 2023) does not reverse the output digits for basic operations such as addition and simple multiplication, which results in missing reasoning steps. RevOrder integrates the benefits of both approaches, minimizing the occurrence of missing reasoning steps while maintaining clarity in the solution process.

metic 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 with a large margin.

The remainder of this paper is organized as follows: 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

Equation complexity Dries and Moschovakis (2009) early obtain lower bounds on the cost of computing various arithmetic functions (Dries and Moschovakis, 2009). Gowers and Wolf (2010) focused on complex linear equations complexity (Gowers and Wolf, 2010). Few have attempted to evaluate the basic equations complexity. 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.

Decomposition of formulas For addition and subtraction, Lee et al. (2023) proposed reversing the output digits, significantly improving sampling efficiency. However, their methods are primarily effective for simpler addition and subtraction operations and do not extend to solving division or large-digit multiplication challenges.
GOAT-7b (Liu and Low, 2023) solves more complex arithmetic operations by decomposing them into a series of simpler operations (Liu and Low, 2023). Differently, our method incorporate the reverse method in the intermediate decomposition step, which greatly improves the computational efficiency and efficiently deal with the solving division or large-digit multiplication challenges. Notably, we employ unique rollback strategies in our approach when tackling division tasks.

Token economy RevOrder introduces an efficient method to keep equations' CSID low, ensuring their manageability within constrained token budgets. XVal presents another approach by directly encoding numerical values into LLMs, offering greater token efficiency compared to RevOrder (Golkar et al., 2023). However, integrating such a method with modern LLM architectures is challenging due to the requisite changes in network structure. Additionally, the current performance of XVal is falling far behind RevOrder.

Performance Almost all current methods of arithmetic computation fail to achieve 100% accuracy, especially when dealing with large numbers and division problems. In contrast, our method succeeds in achieving 100% accuracy.

3 Sequential Intermediate Digits in Arithmetic Computation

Arithmetic reasoning in language models (LMs) is challenging, mainly due to the sequential prediction of digits(Lee et al., 2023). 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 $125 + 179 = 304$, 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 Plain 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 $a = a_n a_{n-1} \dots a_2 a_1$ and $b = b_m b_{m-1} \dots b_2 b_1$, resulting in $c = c_t c_{t-1} \dots c_2 c_1$, with $m \leq n$. When involving negative numbers, the minus sign '-' is also treated as a digit.

- In addition and subtraction, the computation sequence $a_n a_{n-1} \dots a_2 a_1 \pm b_m b_{m-1} \dots b_2 b_1 = c_t c_{t-1} \dots c_2 c_1$ 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.

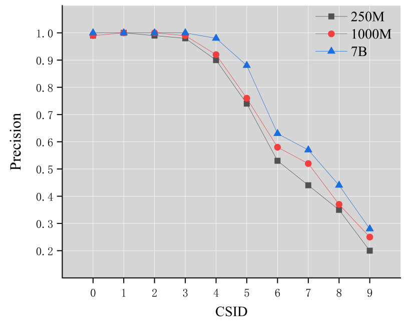


Figure 2: Performance of LLMs on equations with varying CSIDs. This graph illustrates how CSID values affect LLM accuracy, with data obtained under the training protocols outlined in Section 5.

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: Reducing the CSID for Equations

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

Following Lee et al. (2023), we reverse the output digits for addition and subtraction.

$$a \pm b = \$c_1 c_2 \ldots c_t\$$$

Numbers enclosed within \$ symbols are represented in reversed order. Lee et al. (2023) demonstrated that this formatting enables the model to generate the least significant digit (LSB) first, mimicking the typical human approach to addition and subtraction.
We present an analysis of the CSID for this method. To generate each c_i in $\$c_1 c_2 \ldots c_t\$$, 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 reversing the order of the output digits.
Note that we make a slight modification in our implementation compared to Lee et al. (2023). Their format is:

$$\$a \pm b = c_1 c_2 \ldots c_t\$$$

Enclosing the entire equation within \$ symbols complicates the use of addition and subtraction as basic components for more complex operations.

4.2 Multiplication and Division

RevOrder skillfully integrates the chain-of-thought (COT) technique (Liu and Low, 2023) with the reversing output digits technique (Lee et al., 2023), effectively maintaining a low CSID for both multiplication and division equations.

4.2.1 Multiplication

Firstly, consider the simplest form of multiplication, nD by 1D, e.g, 12*7=\$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.

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

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 contrast to the merge operation in Eqn. (3), which requires approximately $\log_2 m$ iterations, GOAT-7B (Liu and Low, 2023) combines numbers one at a time and requires about m iterations. 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 $948 \div 12 = 79$:

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

RevOrder utilizes traditional long division for step-by-step decomposition and reverses all output digits in intermediate addition, subtraction, and nD

by $1D$ multiplications. The overall CSID complexity remains $\mathcal{O}(m)$, primarily due to the quotient estimation steps, as noted in Eqn. (5) and Eqn. (6), while other components sustain a CSID complexity of $\mathcal{O}(1)$. The CSID analysis for quotient estimation is detailed in Appendix A, confirming that the CSID complexity for division within RevOrder is $\mathcal{O}(m)$.

Quotient estimation represents a bottleneck and accounts for the majority of errors in practice. To address this challenge, we have proposed a novel rollback mechanism. If an incorrect quotient is detected, as illustrated in Eqn. (7), we insert a symbol 'W' following 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. A proportion of rollback scenarios are included in the training data to teach the model how to correct such errors.

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

Although rollback technique can correct most of the errors, 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 further 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=00084 + 0006 + 027 + 48 = 00045 + 408 = 40845 = 54804'.

Similarly, the division example can be condensed to: '948÷12 = 7R - (12×70)(048)(801) # 9R - (12×9)(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.

Method	+	−	×	÷
Plain	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 - m^2)$
Simple Reverse	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$	$--$
GOAT-7b	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n+m)$
RevOrder	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(m)$

Table 1: The CSID complexity for different methods. "Plain" denotes the direct generation of results.

4.4 A Comparison of CSID Among Different Methods

Table 1 compares the CSID complexity of RevOrder with other methods. The complexities for Plain, Simple Reverse and GOAT-7b are detailed in Appendix A. It is evident that RevOrder offers advantages across all types of arithmetic operations.

5 Experiments on Arithmetic Operations

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 - 5.3)

- RQ2: Is RevOrder a token efficient format? (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=961'. Note this experiment aims at testing the LM's capability of doing arithmetic operations, hence no prompt engineering is included. 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 a huge amount training data (1m-50m instances) enables GPT models to solve math problems without external calculators.
- Simple Reverse (Lee et al., 2023): This method initially proposed reversing the order of output digits. It is important to note that the Simple Reverse method cannot be applied to division.

5.2 Main Results (RQ1)

The results, as presented in Table 2, 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

```
Correct result: 939007
------------------------------------------------------------
429661432990/457570=
9R -9×45757000000($000000318114$)($09923484871$)#
3R -3×4575700000($00000172731$)($0992331214$)#
8R -8×457570000($0000650663$)($099277064$)#
9R -9×45757000($000318114$)($09995984$)W#
0R -0×45757000($0$)($09927706$)#
9R -9×4575700($00318116$)($013804-$)W#
...
=93808325498165
```

Figure 3: An error example of division by RevOrder.

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 also decomposes 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

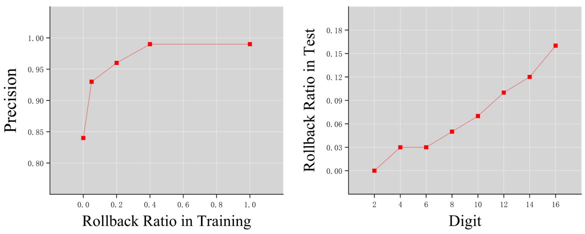
Large-digit division represents the sole operation where RevOrder encounters notable difficulties, warranting additional focus.

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 complexity is $\mathcal{O}(m)$. 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

Task	BIG-bench					Extra Tasks		
ADD	1D	2D	3D	4D	5D	8D+8D	16D+8D	16D+16D
Simple Reverse	100	100	100	100	100	100	100	100
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
Simple Reverse	100	100	100	100	100	100	100	100
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$6D
Simple Reverse	100	100	80.4	35.5	10.7	100	0.0	2.1
GOAT-7B	100	100	97.8	96.9	96.7	99.7	88.1	96.8
MathGLM-2B	100	99.9	98.3	94.9	89.9	-	-	-
RevOrder-1B	100	100	100	100	100	100	100	100
DIV	1D	2D	3D	4D	5D	16D$\div$1D	6D$\div$3D	12D$\div$6D
Simple Reverse	-	-	-	-	-	-	-	-
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 2: Performance comparison on various arithmetic operations. The results of the baseline methods are taken from their original paper, while the result of Simple Reverse is based on our implementation.



(a) (b)

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.

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

Table 3: 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.

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)

By maintaining a low CSID, RevOrder simplifies the learning process for arithmetic problems, thereby reducing the volume of training data required. Table 3 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., 2022). Consequently, the training cost advantage of RevOrder is likely to be even more pronounced with larger LLMs.

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 a pair of extra tokens ('$') 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,

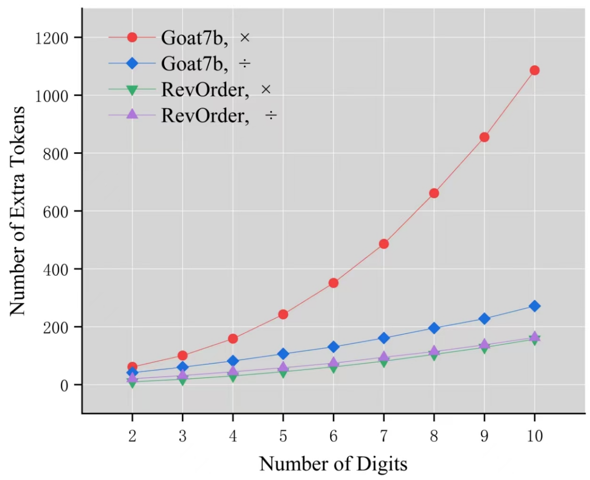


Figure 5: The number of extra tokens required for multiplication and division.

the compact form introduced in Section 4.3 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.

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). 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.

	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 4: 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.

6.2 Results

From Table 4, 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.

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

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*.

Lou Van Den Dries and Yiannis N Moschovakis. 2009. Arithmetic complexity. *ACM Transactions on Computational Logic (TOCL)*.

Siavash Golkar, Mariel Pettee, Michael Eickenberg, Alberto Bietti, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Parker, et al. 2023. xval: A continuous number encoding for large language models. *arXiv preprint arXiv:2310.02989*.

William T Gowers and Julia Wolf. 2010. The true complexity of a system of linear equations. *Proceedings of the London Mathematical Society*.

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*.

Nayoung Lee, Kartik Sreenivasan, Jason D Lee, Kangwook Lee, and Dimitris Papailiopoulos. 2023. Teaching arithmetic to small transformers. *arXiv preprint arXiv:2307.03381*.

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*.

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*.

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.

A The CSID Analysis for Multiplication and Division

This section extends the CSID analysis to nD by nD multiplication and nD by mD division.

A.1 Multiplication

A.1.1 The CSID for Plain Multiplication

We assume the plain method adopts a similar decomposition method in Section 4.2, but without reversing the output digits.

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.

A.1.2 The CSID for Multiplication in Simple Reverse

Simple Reverse (Lee et al., 2023) only omits n SID by the reversing operation, leaves the overall complexity being unchanged $\mathcal{O}(n^2)$.

A.1.3 The CSID for Multiplication in GOAT-7b

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

A.2 Division

A.2.1 The CSID for Quotient Estimation

Estimating a quotient c when dividing by a divisor $b = b_m b_{m-1} \ldots b_1$ typically requires only the first m or $m+1$ digits of the dividend a. We consider the scenario where the length of a is m and $a_m > b_m$. The case where the length of a is $m+1$ and $a_m < b_m$ is omitted for brevity, as the analysis and results are analogous.
In an optimal scenario where $a_m = 9$ and $b_m = 8$, c can be deterministically set to 1, and no SID is incurred. However, in the least favorable case where $a_m = 9$ and $b_m = 1$, c could potentially be any of $5, 6, 7, 8$, or 9. To accurately determine the quotient, it is necessary to evaluate each candidate quotient $\hat{c}$:

$$d = a - \hat{c} \times b$$

The candidate $\hat{c}$ is deemed correct if d is a non-negative number less than b. Calculating d requires approximately $2m$ SIDs when using RevOrder (m for storing the results of the multiplication and m for storing d), or $4m$ when not using RevOrder, making the total CSID in the worst scenario about $10m$ or $20m$. Therefore, the complexity of quotient estimation remains $\mathcal{O}(m)$.

A.2.2 The CSID for Plain 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, and $20m$ for quotient estimation.
Thus, the total CSID for an nD by mD division reaches $(22m + 2n) * (n - m)$, amounting to a complexity of $\mathcal{O}(n^2 - m^2)$.

A.2.3 The CSID for Division in GOAT-7b

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

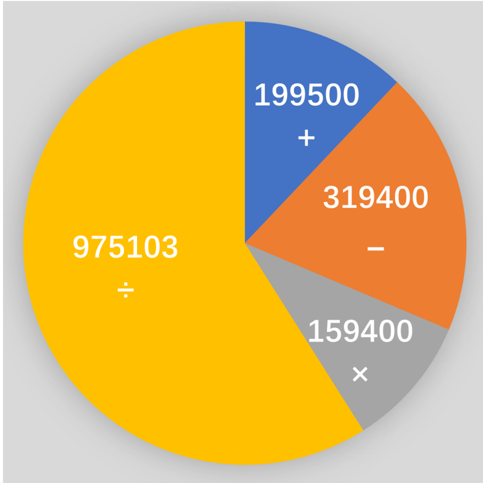


Figure 6: The distribution of the equations in training set.

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.

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

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.

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.

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.