NUMBER COOKBOOK: NUMBER UNDERSTANDING OF LANGUAGE MODELS AND HOW TO IMPROVE IT

Anonymous authors
Paper under double-blind review

ABSTRACT

Large language models (LLMs) can solve an increasing number of complex tasks while making surprising mistakes in basic numerical understanding and processing (such as $9.11 > 9.9$). The latter ability is essential for tackling complex arithmetic and mathematical problems and serves as a foundation for most reasoning tasks, but previous work paid less attention to it or only discussed several restricted tasks (like integer addition). In this paper, we comprehensively investigate the **numerical understanding and processing ability** (NUPA) of LLMs. Firstly, we introduce a test suite covering four common numerical representations and 17 distinct numerical tasks in four major categories, resulting in 41 meaningful combinations in total. These tasks are derived from primary and secondary education curricula, encompassing nearly all everyday numerical understanding and processing scenarios, and the rules of these tasks are very simple and clear. We find that current LLMs exhibit a considerable probability of error in many of them. To address this, we analyze and evaluate techniques designed to enhance NUPA, identifying three key factors that influence NUPA of LLMs. We also perform finetuning of practical-scale LLMs on our defined NUPA tasks and find that a naive finetuning improves the performance but these tricks cannot be directly used to finetune an already-trained model. We further explore the impact of chain-of-thought techniques on NUPA. Our work takes an initial step towards understanding and improving NUPA of LLMs.

1 INTRODUCTION

The mathematical and reasoning abilities of large language models (LLMs) are currently quite impressive (OpenAI, 2023; Meta, 2024; OpenAI, 2024; Yang et al., 2024), capable of solving problems at the level of a graduate student or even more difficult ones like olympiad-level problems (He et al., 2024), GAOKAO (a nationwide examination of high school students applying to universities in China) (Zhang et al., 2024) and college mathematics (Tang et al., 2024). However, upon closer examination of the models' outputs, we found that although the models demonstrate remarkable proficiency in problem-solving approaches, they often struggle with basic numerical understanding and processing — like a careless student who claims, "*I know how to do it, but I didn't get it right.*" Some of these errors are quite surprising, such as believing that $3.11 > 3.9$ or making mistakes in simple addition $8/7 + 3/5$. These types of errors are a major cause of hallucinations when dealing with math, reasoning, and data analysis tasks, as the model presents seemingly correct problem-solving approaches but ultimately produces incorrect results. Therefore, improving the fundamental "**numerical understanding and processing abilities**" (NUPA) of models is crucial.

However, as we mentioned earlier, in current tests and research, the tasks of "numerical processing" and "logical reasoning" are often mixed together, and much of the evaluation is based on solving various word problems or complex problems. And even the numerical content is often simplified in these datasets. For example, in various exam questions, to focus on assessing students' understanding of mathematical concepts — such as how to set up the correct equations and apply the right theorems — the numbers in both the questions and answers are often specially chosen to be integers. However, this is not the case in real-world scenarios.

Therefore, an accurate, detailed and comprehensive formalization, measurement, and analysis of this fundamental capability remains lacking. In this paper, we first try to formalize the NUPA of LLMs.

Table 1: Task overview of NUPA Test. ✓: 41 tasks included in our test. ✗: Not included, too complex. ◯: Not directly included but can be easily transferred from an included task. −: Not applicable. Detailed explanation for these non-included tasks refers to Appendix A.1.2

	Add	Sub	Multiply	Truediv	Floordiv	Mod	Max	Min	Digit Max	Digit Min	Digit Add	Get Digit	Length	Count	To Float	To Scientific	Sig. Fig.
Integer	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	−	✓	✓
Float	✓	✓	✓	✗	−	−	✓	✓	✓	✓	✓	✓	✓	◯	−	✓	✓
Fraction	✓	✓	✓	✓	−	−	✓	✓	−	−	−	−	−	◯	✓	◯	◯
Scientific	✓	✓	✓	✗	−	−	✓	✓	−	−	−	−	−	◯	✓	−	◯

By summarizing the concepts and operations involving numbers in primary and secondary education into four types of numerical representations: *integers*, *floating-point numbers* (finite decimals), *fractions*, and *scientific notation*, along with four categories comprising 17 tasks. Pairing these representations results in 41 meaningful tasks, forming our NUPA benchmark (Table 1). These tasks cover most common scenarios involving number understanding and processing, providing a more comprehensive measure of NUPA of LLMs. We test several state-of-the-art LLMs containing GPT-4o (OpenAI, 2024), Llama-3.1 (Meta, 2024) and Qwen2 (Qwen Team, 2024), which claim strong mathematical abilities. Although the latest LLMs performed well on most typical tasks, their performance declined significantly as tasks became slightly more complex or specialized (such as multiplication, modulo operations, or digit-based calculations), or when the representation of numbers extended beyond basic integers to floating-point numbers and other formats (Section 2.4, Figure 2). This performance decline may impede the models' reasoning capabilities in addressing more complex and practical problems.

To address this issue, we explore three possible approaches to enhance the NUPA of models. First, we summarize previous techniques aimed at improving models' numerical processing abilities, and evaluate and analyze them on our newly introduced tasks, representations, and metrics. These techniques contain different tokenization, specially-designed positional encoding (PE) (Haviv et al., 2022; Kazemnejad et al., 2023b; Zhou et al., 2024a), special format of numbers (like zero-padding, index-hint (Zhou et al., 2023) or reverse representation (Lee et al., 2023; Zhou et al., 2024b)). We find that past techniques can be broadly classified into three mechanisms: assisting mathematical reasoning, aiding digit alignment, and providing regularization. We also discuss the applicability of these mechanisms across different numerical processing scenarios.

Next, we investigate how to improve NUPA for an already trained model and discover that while simple direct finetuning can significantly enhance NUPA performance, introducing these tricks in this stage may have negative effects. We test with several different setting and finetuning configuration but no one can achieve similar or better NUPA performance than the original model. Our findings suggest that these modifications can significantly disrupt the model's original information flow or conflict with its pre-existing knowledge.

Finally, we discuss the potential of using chain-of-thought techniques for numerical processing. Although chain-of-thought methods can break down complex problems into simpler sub-tasks and significantly increase the likelihood of obtaining correct answers, their drawbacks—such as consuming large reasoning windows and requiring extended processing time—become particularly apparent in numerical tasks. We tested a general chain-of-thought method known as RFFT, and found that for more complex tasks (such as those with $O(n^2)$ complexity, including multiplication, division, and fraction simplification), chain-of-thought methods face scalability challenges, making them difficult to apply in practical scenarios.

In summary, we conclude that current research is insufficient to fully solve the basic NUPA task, despite it being a fundamental capability of models. We hope that by introducing a more comprehensive definition and evaluation of NUPA, we can bring greater attention to this area and encourage researchers to include numerical processing, alongside reasoning, in their evaluations of model performance.

2 NUPA TEST: A COMPREHENSIVE BENCHMARK FOR NUMBER UNDERSTANDING AND PROCESSING ABILITY

In this section, we will introduce our NUPA benchmark from the following four aspect: number representation, tasks, metrics, and the test performance analysis about current LLMs. We will explain the rationale behind the inclusion (or exclusion) of specific representations and tasks in our benchmark, highlighting their distinctive features. By comparing these representations and tasks

with previous benchmarks, we will demonstrate that several common tasks and abilities have been overlooked in prior evaluations.

2.1 NUMBER REPRESENTATION

As we discuss above, we believe that the educational curricula on the primary and secondary school stage is a good reference on what NUPAs are necessary and expected to learn by LLM. We identify four number formats that are both common and capable of covering most practical scenarios.

- **Integer**: The integer is the first set of numbers that we met in primary school and the most common representation of a number. Some experiments on animals (Davis & Memmott, 1982) and infants (Strauss & Curtis, 1981) have found that understanding of (simple) integers is somewhat innate because it is used naturally in counting various countable objects. Understanding and operations of more complex integers is also acquired very early in human civilization (Smith & Karpinski, 1911; Dauben, 2002). At the same time, integers serve as the foundation for other mathematical representations.
- **Fraction**: Specifically, the fractions whose numerators and denominators are integers. Once the "division" operation has been introduced in integers, the concept "*fraction*" has occurred. Extending from the integer to the fraction (i.e. rational numbers) is the first extension of our understanding of number set. In life, once it comes to distribution, fractions are inevitable. However, though the concept of fraction is intuitive, the operation on fractions is a little more complex. As we will discuss later, mastering fraction operations remains a significant challenge for current LLMs.
- **Floating-point numbers** (Float): Finite decimals are a subset of fractions whose denominators consist of powers of 2 and 5. Since this representation aligns with the decimal number system, it simplifies calculations—especially addition, subtraction, and comparison—allowing them to function similarly to integer operations. This simplicity makes finite decimals common in everyday use. In cases where exact precision isn't necessary, most fractions are eventually converted to floating-point representation. Floating-point numbers are also one of the most prevalent formats in computer-related tasks. However, not all fraction operations can be accurately converted into decimal form. Consider this problem solved by ChatGPT:

  > The bookstore sells newspapers and magazines. Bob bought 7 newspapers and 3 magazines for 13 dollars, and Alice bought 3 newspapers and 7 magazines for 23 dollars. How much are the newspapers and magazines, respectively?

  ChatGPT sets up a system of linear equations: $7x + 3y = 13$ and $3x + 7y = 23$. Using the elimination method, $x = \frac{13}{7} - \frac{3}{7}y$, we can easily find the integer solution $x = 1$, $y = 2$. In this case, the fractions $\frac{13}{7}$ and $\frac{3}{7}$ cannot be approximated as floats like 1.86 and 0.43, as this would prevent us from arriving at the exact integer solution.
- **Scientific Notation**: Scientific notation is a common way to represent floating-point numbers, characterized by separating a number's precise value from its order of magnitude. The magnitude is expressed using a base-10 exponent, while the float represents the precise value. This format introduces some complexity, as numbers with different exponents cannot be easily aligned by the decimal point. However, it is widely used in fields like physics, economics, and computer science because it efficiently handles a wide range of numbers and clearly conveys significant figures and precision. For LLMs, mastering scientific notation can significantly enhance their ability to handle practical tasks, such as interpreting financial reports or understanding scientific texts.

There are possible representations of numbers that are not included in these four formulas, like *complex numbers*, *infinite decimal representation* (repeating and non-repeating), *radical expression* (like $\sqrt{2}$), ... These representations either occur relatively infrequently in practical conversations (e.g., complex numbers) or present significant challenges for language models to process effectively without the aid of external tools (e.g., radicals). For these reasons, we have opted not to include them in our benchmark at this stage.

2.2 TASKS IN FOUR CATEGORIES

Another aspect of NUPA is defining the tasks that the model needs to handle. The tasks should have clear calculation rules — any student who has completed a full primary and secondary education should be able to accomplish them. Furthermore, a vast majority of practical numerical processing tasks are either included in these tasks or can be easily transformed into some of them. We propose

17 tasks across four categories. The complete task list is shown in Table 1 and for each task we provide an example in Appendix A.1.1.

- **Elementary arithmetic**: **addition**, **subtraction**, **multiplication**, and **division**. They are the most fundamental mathematical operations and the first branch of mathematics taught in schools. However, some operations can be complicated when different number representations are involved. For example, fraction addition is more complicated than multiplication because it needs to be reduced to a common denominator first. Additionally, multiplication of long integers or floating point numbers actually involves L^2 complexity (L is the length of digits), which poses a challenge to models.
  - **True division, floor division and modulus**: Division is somewhat unique because it is not closed for integers and float numbers. Here, we consider three common division-related calculations. **True division**: To maintain precision, we represent the division of two integers as a simplified fraction. Combined with the "significant digits" task we will mention later, this can approximate the result of dividing two integers as a floating-point number. **Integer division** and **modulus**: Represent approximate multiple relationships, frequently used in practical applications - such as dividing individuals into batches.
- **Compare**: **Max** and **min**. Another important aspect of understanding of numbers is the concept of "order". To truly comprehend a number, we must know how large it is and whether it is greater or smaller than another number. Moreover, comparison serves as the foundation for other significant operations. For instance, when adding negative and positive numbers, we actually determine the sign first and then subtract their absolute values - this involves identifying which of the two numbers has a greater absolute value.
- **Digit understanding**: The concept of a digit is fundamental. Unlike the "value" of a number, a digit is tied to its specific representation. When we care about a language model's understanding, processing (and generation) of numbers, digit is an unavoidable issue, as they are the units that our tokenizers and models directly handle. Numbers are not read and processed by the language model as a whole, but rather as a sequence of digits. Therefore, we specially designed some tasks that may not be so practical but are useful for exploring whether the language model correctly understands the concept of digits. Including:
  - **Digit compare**: compare and return the larger (smaller) digit one by one.
  - **Digit add**: Operate the normal addition but *ignore any carrying*. For example, $\mathrm{digit\_add}(12345, 34567) = 46802$. This calculation is independent of each digit. In general addition, it can test the model's understanding of digit alignment and its mastery of the unit operation of single-digit addition.
  - **Get digit**: Given a number and an integer, return the corresponding digit, like the subscribe operator of a string. **Length**: Return the total length (i.e., the number of digits) of the number.
  - **Count**: Count the number of times a particular digit occurs in an integer.

  Through these tasks, we can assess whether the model correctly understands the concepts of the number's digits (length), the position of a digit, and the alignment of the digits between two numbers.
- **Conversion between representations**: we designed tasks for converting to two representations: **to float** and **to scientific notation**, as they are frequently used to present final results. These two tasks also create transformations between different representations to test whether the model can understand the relationship between various numerical formats. In particular, since many tasks present answers as approximate values, we designed a "**significant digit**" (**sig. fig.**) task to evaluate the model's ability to round longer numbers to fixed-digit significant digits.

The combination of representations and tasks ultimately resulted in a total of 41 meaningful tasks. The remaining tasks were excluded due to being excessively complex, uncommon, inapplicable, or redundant with other tasks; for further details, see the discussion in the Appendix A.1.2.

The difficulty of each task depends not only on the nature of the task itself, but also on the length of the numbers to be processed — longer tasks involve more steps of internal operations, as well as longer inputs and outputs. Therefore, we want to evaluate model performance on different problem lengths. For tasks that are inherently more difficult, we limit the size of the problem to 1-20 digits, and for easier tasks to 1-100 digits. (For which tasks are considered difficult or easy, please refer to the Appendix A.1.3.) We generated 1,000 questions as the NUPA benchmark for each length and each task. Unlike some previous works that set the lengths of two numbers to be the same, in our

tests, the length L of a question is determined by the longer of the two numbers, while the length of the shorter number follows a uniform distribution between 1 and L. For some tasks, randomly generated numbers may lead to unusual answers. For instance, performing addition or subtraction on numbers in scientific notation with significantly different exponents may result in very long (and impractical) outcomes, like $1.1e10 - 2.1e1 = 1.0999999979e10$. To address such issues, we implemented additional handling to ensure the generated problems do not result in overly simple, overly complex, or meaningless results. For some tasks, we also split a task into a hard and an easy version. More details can be found in Appendix A.1.4.

2.3 METRICS ABOUT NUPA

Measuring the performance of NUPA benchmarks on these tasks is not trivial. "**Exact match**" accuracy is undoubtedly the golden standard of the performance where only the answer is considered as correct when it exactly match the groundtruth. For most practical tasks involving numbers (like arithmetic or math tests), all we care is whether the answer is right, and there is no significant difference between being almost right and being completely wrong. But having a smoother and more detailed metric is more important when we want to understand the capabilities of a model, especially focusing on current flaws in the model and how to make it better. Therefore, we also report the "**digit match**" and "**dlength**" metrics. For "digit match", we first align the generated answer with the groundtruth digit by digit. The integer parts (integer, integer part of a float, the numerator and denominator of a fraction, the exponential of a scientific notation) are aligned from the least significant digit; and for the decimal part of float (and the float in scientific notation), they are aligned from the most significant digit. We then measured the accuracy for each digit, with missing digits considered as errors, and averaged them for each digit. For "dlength", we report the absolute value of the difference in length between prediction and groundtruth. Figure 1 provides an illustration for these three metrics.

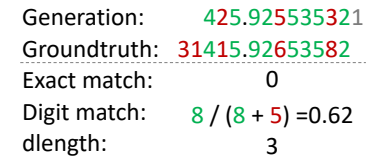


Figure 1: An example of metrics.

For each task, we divide the digits into four intervals (s, m, l, xl), where the total number of tasks from 1-20 corresponds to 1-4[1], 5-8, 9-14, 15-20; the total number of tasks from 1-100 corresponds to 1-10, 11-20, 21-60, 61-100. We average the results in each range for each task and metric. Besides, to demonstrate the model's upper and lower performance limits, we also present the following metrics: *well-learned digits* and *performance-preserving digits*. These represent the maximum number of digits that can maintain over 90% accuracy and the maximum number of digits that can maintain over 10% accuracy, respectively. (For digit match, the thresholds are set to 90% and 50%, and for dlength, where smaller is better, the thresholds are 0.1 and 1).

2.4 PERFORMANCE OF CURRENT LLMS

We test some common-used LLMs on our benchmark, including three Llama models: Llama-2-7b, Llama-3.1-8b and Llama-3.1-70b (Meta, 2024), because they are the most popular open-source model; Mixtral-8×7B (Jiang et al., 2024) as a strong MoE model and Qwen2-2B and -72B (Qwen Team, 2024) which are also open-source models and are believed to have strong math ability. We also test GPT-4o-2024-08-06 and GPT-4o-mini-2024-07-18 on our benchmark. The results on parts of tasks are shown in Figure 2 while the full results are shown in Appendix B.1 as well as the "digit-match" and "dlength" metrics. We have several observations regarding the results:

The optimal model performs well on typical tasks, but its performance declines on many less common tasks. We find that GPT-4o, GPT-4o-mini and Qwen2 can solve typical tasks such as integer addition, float addition, integer max, integer length with high accuracy among S and M ranges. This is also consistent with their excellent performance on various mathematical datasets. However, accuracy drops significantly when dealing with less common representations, such as fractions

[1]Because we also generate a larger training set (with $100,000$ different pairs of numbers for each digit and each tasks), we prioritized ensuring that certain basic operations appeared in the training set, which led to some lengths being absent in the test set. For example, there are no tests for the addition of 1- or 2-digit integers because $100 \times 100 < 100,000$.

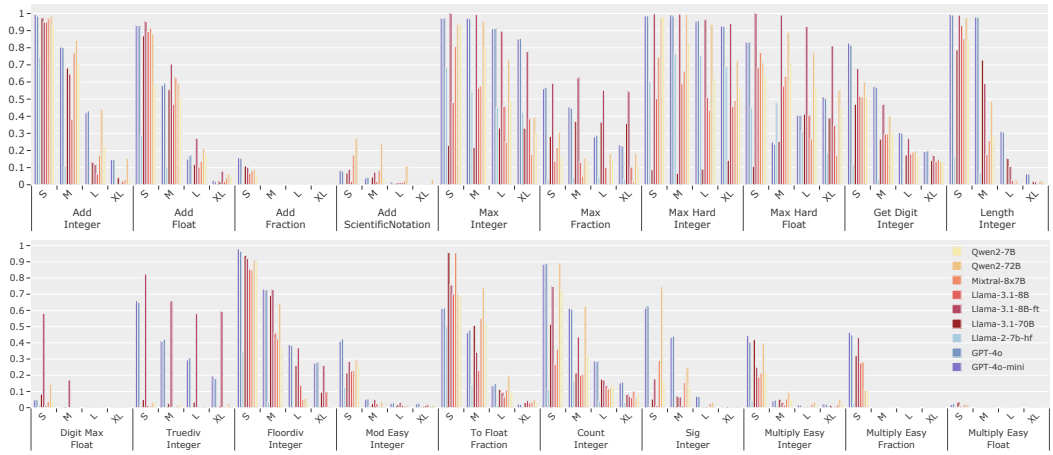


Figure 2: Parts of performance of state-of-the-art LLMs on NUPA benchmark.

and scientific notation, with average accuracy falling below 20%, even in the shortest S-range (1-4 length input). For tasks such as significant figure, modulo operations, or digit-based calculations, the model's performance was similarly unsatisfactory. This highlights the current limitations of LLMs in understanding numerical diversity and complexity — they only demonstrate proficiency with a small subset of numbers, while for many other tasks, they remain unfamiliar and unable to produce accurate answers.

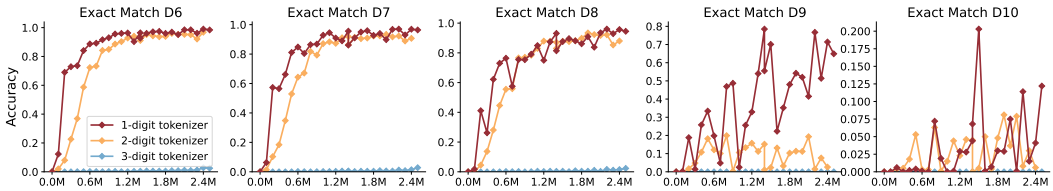
Length still remains an obstacle for NUPA of LLMs. We observe a clear decline in accuracy for even simple integer addition as problem length increases. The precision of GPT-4o drops from nearly 100% in the S range and 80% in the M range to around 40% in the L range and 15% in the XL range. For the more challenging floating-point addition task, the accuracy falls from 90% (S) and 60% (M) to just 15% (L) and under 5% (XL). The same trend is evident in other models and tasks. For instance, Qwen2's performance in the integer length task declines from almost 100% (S) to 50% (M) and below 5% (L and XL).

Concept "digit" is not as easy as we expected. We were surprised to discover that LLMs struggle to fully grasp the concept of a "digit". For instance, in the "get digit" task, where the model is given a long integer and an index, and asked to return the corresponding digit, performance is strong in the shorter S-range. This suggests that the model can at least understand the task requirements. However, as the length of the number increases, performance drops significantly, indicating that the model lacks a consistent understanding of what a digit represents. In the XL-range, GPT-4o manages only 20% accuracy, barely above random guessing, which would achieve 10% accuracy (since the answer is always a digit between 0 and 9). This fundamental limitation likely explains why current models struggle with numerical understanding and processing across a wider range of tasks and longer input lengths. If a model cannot reliably determine a certain digit in a given number, it casts doubt on its ability to learn and apply more complex arithmetic operations in a generalizable way, such as addition. The performance on these tasks may largely rely on case-based reasoning, as pointed out by Hu et al. (2024).

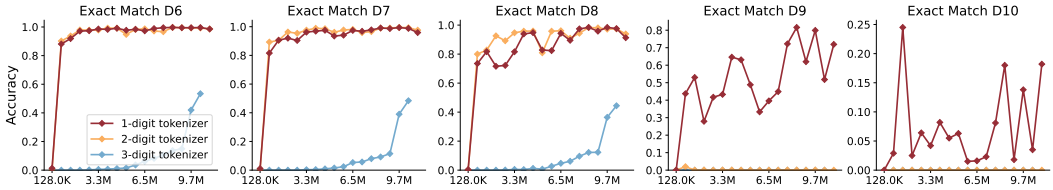
We also have some interesting observations: (1) LLMs find the "max-hard" task easier than "max" with integer inputs. The difference is that in the max task, the two numbers often differ in length, whereas in max-hard, they are always the same length and share some left-most digits, requiring more digits to be compared. While max-hard intuitively seems more difficult, models actually perform better on it, likely because they struggle to effectively use sequence length information, as evidenced by their weaker performance on the "length" tasks in the longer ranges. This suggests that models might process tasks in different ways from humans. They could have to compare two numbers digit-by-digit. In this situation, the "harder" subtasks is actually easier because the numbers have been aligned. (2) GPT-4o and GPT-4o-mini show nearly identical performance across most tasks, similar to the comparison between Qwen2-72B and Qwen2-7B. This suggests that after reaching a certain size, NUPA performance depends more on the training approach, such as data diversity and post-training, rather than simply on model size.

3 How do tokenizer and other tricks affect NUPA?

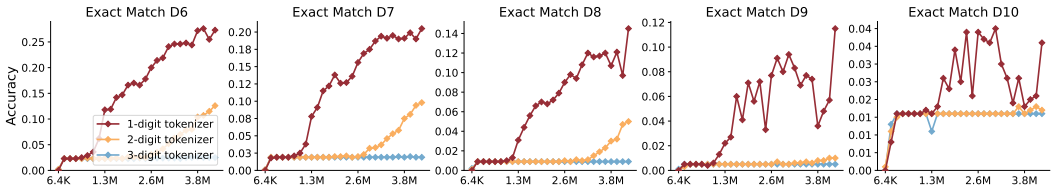
We use the architecture of decoder-only transformers and alter the number of attention heads and layers together with the hidden size and intermediate size to get models of various param-



(a) 0.9B model int add



(b) 0.9B model float add



(c) 0.9B model int multiply

Figure 4: Accuracy of 0.9B models trained with 1-3 digit tokenizer on three task of integer addition, float addition and integer multiplication.

eter sizes, like 0.1B, 0.9B and 3B on integer addition, float addition, and integer multiplication. We then train the models from scratch with a wide range of techniques that may affect NUPA. We confirm the effects of the various techniques through experiments, including tokenizers, positional encodings (PE) and data formats, such as reverse formatting, zero-padding and index hints.

3.1 TOKENIZER

LLMs interpret numbers as segmented tokens rather than whole numbers or digits like humans. With the development of language models, various tokenization strategies have emerged, including 1-digit tokenizers, mixed tokenizers, and 3-digit tokenizers, as shown in Figure 3. There has been a trend towards extending each numeric token. However, the question of which tokenizer best assists the model in understanding and processing numbers remains an open question. Longer tokenizers reduce training and inference costs per sample, as they decrease the number of tokens per sample. However, the trade-off is that long tokenizers require exponentially larger number vocabularies. For tokenizers with such large vocabularies, developing NUPA might demand the model to also see exponentially more examples, which is intuitively more challenging. In this section, we will mainly discuss tokenizers of 1 to 3 digits. The n-digit tokenizer greedily segments a number from left to right into n-digit tokens until a remainder shorter than n digits is left, which is then segmented into a single token.

(a) 31415.926535897932

(b) 31415.926535897932

(c) 31415.926535897932

Figure 3: Different tokenization of a long number in representative LLMs. (a) 1-digit tokenizer, used in Llama-2. (b) mixed digit tokenizer, used in GPT-2. (c) 3-digit tokenizer, used in GPT-3.5, GPT-4 and Llama-3.

We train 0.9B models on 1- to 8- digit length training samples of tasks including integer addition, float addition, and integer multiplication. We also discuss experiments on models of 3 different sizes, including 0.1B, 0.9B, and 3B in Appendix B.2.

Figure 4 illustrates the in-domain performance of these models in the first three columns and their out-of-domain (OOD) performance in the last two columns. Here we use the exact match metric. From the figure, we find that except for in-domain performance of integer addition, the 1-digit tokenizer consistently exceeds the others by large margins in both in-domain tests and length generalization. In contrast, the 3-digit tokenizer exhibits poor performance in both in-domain and out-of-domain evaluations. Tokenizers with an increasing number of digits significantly hinder subbillion models' NUPA. In the experiments of the 3B model in Appendix B.2, the 3-digit tokenizer shows the potential in length generalization for the first time, yet its performance remains inferior to that of

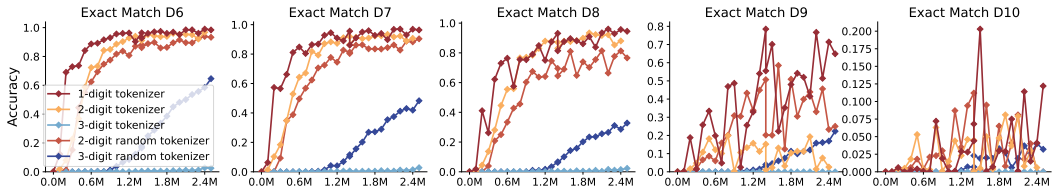


Figure 5: Accuracy of 0.9B models trained with 1- to 3- digit tokenizers and 2- to 3- digit random tokenizers on integer addition.

the smaller tokenizers. This indicates that scaling up the model size indeed alleviates the challenges in developing NUPA caused by larger tokenizers. Nevertheless, larger tokenizers do not present any distinct benefits in either in-domain or out-of-domain generalization in our models.

Random tokenizer. Additionally, we investigate the random tokenizer, as introduced in Sathe et al. (2024). They claim that this trick can improve the reasoning ability of LLMs by giving the same reasoning path with different generating processions. Here, we try this idea in the number domain, which means the tokenizer segments a number into tokens of digit lengths randomly chosen between 1 and a predefined maximum iteratively instead of segmenting numbers from left to right greedily. Unlike GPT-2 tokenizer shown in Figure 3 (b), random tokenizers can give different tokenizations for a fixed number while the GPT-2 tokenizer can not. We denote this random tokenizer with a maximum digit length of n as the n-digit random tokenizer. It should be noted that the 1-digit tokenizer is equivalent to the 1-digit random tokenizer.

We show the performance of 1- to 3-digit tokenizers and 2- to 3-digit random tokenizers in Figure 5. 2- and 3-digit random tokenizers outperform corresponding standard tokenizers respectively in length generalization. Regarding in-domain generalization performance, the 3-digit random tokenizer surpasses the 3-digit tokenizer, while the 2-digit random tokenizer shows a marginal decrease in performance relative to the 2-digit standard tokenizer. Nevertheless, it should be noted that even though random tokenizers show superior performance in length generalization, they still fail to match the 1-digit tokenizer.

3.2 Tricks to improve NUPA

Table 2: RoPE performance.

	Exact Match		Digit Match		Dlength	
	d8	d9	d8	d9	d8	d9
RoPE	**1.00**	0.00	**1.00**	0.53	0.00	0.67

PEs: In most tasks, there is an intrinsic calculating rule not related to the length of input numbers, we called it the "length-agnostic" rules. However, because the model only accesses a portion of the length, there can be a "length-related" rule that is not distinguishable by the model. For example, the model can learn a "length-related" addition rule that is the normal addition rules + the output should be of length ranging from 1 to 8. This phenomenon can be called length over-fitting. To investigate whether and why different PE will influence the length generalization, we train 100M models with different PEs: RoPE (Su et al., 2023), NoPE (Kazemnejad et al., 2023a) and Alibi (Press et al., 2022) on integer addition of 1-8 length (S and M range), then test them on full range (S to XL, 1-20). RoPE is the most classic relative PEs and Alibi is also a relative PE that claim more effective in length generalization, while NoPE denotes transformers without positional encoding. We find that (1) *RoPE encourages the model to rely on the length of the input*. The first evidence is that RoPE causes the model's predictive performance to plummet dramatically just beyond the training boundary; for instance, accuracy drops from nearly 100% to 0% when moving from 8 to 9 digits, while "dlength" rises from 0 to 0.7 (Table 2). This indicates that the model has a significant probability of generating shorter results, avoiding the generation of 9-digit answers. At the same time, RoPE not only constrains the model's output length but also affects the digit pairing. The performance of 100% for inputs of 8 digits indicates that the model performs calculations for each position unless it can successfully align the corresponding digits. However, when inputting 9-digit numbers, digit match drops significantly to 50%, suggesting that the model has a considerable probability of failing to align the digits. (2) On the other hand, we observe that the *length learning provided by RoPE appears to be a shortcut*. In cases where the model is extremely small or has been trained very little, we see the advantages of this "shortcut". In Table 3, we train a 2-layer transformer (1.3M parameters) on integer addition using 3 different PEs on 1- to 8- digit integer addition or the 0.1B model with only 1M samples, we find RoPE shows the best in-domain performance. However, in actual training, this shortcut, as a case-based learning mechanism for length, fails to achieve a generalization in length. In contrast, although NoPE and Alibi slow down learning, it enables the model to learn a generalizable behavior. Therefore, our conclusion is that RoPE causes over-fitting on length shortcuts while these specially-designed PE can be understood as a **length regularization**.

Table 4: Exact match of 0.1B models trained on integer addition and float addition respectively with various compositions of reverse formatting and zero padding.

	Integer Addition				Float Addition									
	rev	rev +pad	no	pad	rev total	rev total + pad	rev each	rev each + pad	rev dec	rev dec + pad	rev int	rev int + pad	no	pad
d9	**1.00**	**1.00**	**1.00**	**1.00**	0.12	0.24	0.12	0.24	0.12	0.24	**1.00**	**1.00**	**1.00**	**1.00**
d10	0.80	**0.92**	0.32	0.82	0.10	0.22	0.12	0.23	0.09	0.22	**0.98**	**0.98**	0.22	**0.98**

Table 3: 8-digit digit-match accuracy with small model or small dataset.

	1.3M Model	1M Samples
RoPE	**0.091**	**0.97**
NoPE	0.061	0.78
Alibi	0.056	0.23

Data Formats: A series of works focusing on length generalization, especially the addition of language models, has proposed specific data formats to enhance the model performance on the addition of long numbers. Typical techniques include reverse formatting, zero padding and index hints. **Reverse formatting** (Lee et al., 2023; Shen et al., 2023; Zhou et al., 2023; 2024b; Cho et al., 2024) presents numbers in reverse order from the least significant digit to the most significant digit to align with the models' autoregressive mechanism, simplifying the learning process for addition. **Zero padding** (Lee et al., 2023; Shen et al., 2023; Zhou et al., 2024b; Cho et al., 2024) adds leading zeros to numbers to standardize the lengths of operands, helping models align operands. **Index Hints** (Zhou et al., 2023) explicitly include positional information in input and output sequences by representing each digit with their indexes corresponding to the positions prefixing the digit.

While previous work mainly focuses on integer addition or multiplication, we extend the techniques to various tasks in NUPA Test of different number domains for the first time. Reverse formatting is believed to help models perform carry-over calculations and also assist in digit alignment, while zero padding and index hints serve primarily to help with digit alignment. To compare the effects of reverse formatting and zero padding, we demonstrate in Table 4 how the combination of reverse formatting and zero padding impacts length generalization. Reverse formatting, zero padding, and their combination show comparable performance to each other in integer and float addition, which indicates that the role of reverse formatting in assisting calculations is not apparent. To provide an intuitive explanation: regarding addition, examples where reverse formatting can make a difference through the effects of assisting carry-over calculations are quite rare. Most of the time, knowing the result of the next digit allows us to determine the answer for the current digit. When the next digit addition is not less than 10 (without considering further carrying from the following digit), there must be a carrying from that digit into the current one, no matter what the result of the later digits is. And when the next digit addition is not more than 8, there will never be a carrying. The only exception is the next digit addition is 9. In this situation, we must refer to the next two digits to determine the current digit results. Therefore, we point out that, although in the worst-case scenario, performing non-reversed addition requires $O(n)$-length looking forward for each digit, and reversing could solve this problem, such cases are extremely rare. ($44445 + 55556 = 100001$) In most instances, the task can be accomplished with a very limited local view. Assuming the model has the capability to focus on the next n digits to calculate the current digit, then reverse formatting would only aid in calculations for questions with consecutive $n + 1$ carry-overs, which are quite uncommon. Therefore, the role of reverse formatting in aiding calculation is very subtle, and its main functions are similar to padding, primarily helping with alignment tasks. As for index hints, it expands the vocabulary and thus causes an extra burden for models to achieve length generalization. On the contrary, reverse formatting raises the performance of length generalization dramatically even with 0.1B models in multiple tasks. The experiments about index hint are included in appendix B.4.

3.3 Does Finetuning improve NUPA performance of LLMs?

As shown in Figure 2, the current models still exhibit poor performance in some NUPA tests. Additionally, the existing techniques aimed at enhancing mathematical abilities have rarely been applied to practical LLMs, mostly remaining in toy models and limited to a few isolated tasks. We are curious whether it is possible to improve the NUPA capabilities of large models through post-training finetuning. Therefore, based on the NUPA test tasks, we also generate training (10^5 samples for each digit and each task) and validation sets, ensuring that they do not overlap with the previously proposed test set. We then used them to perform LoRA finetuning on a pre-trained model. We finetune a Meta-Llama-3.1-8B model with lora (rank 128, α=32) on a mixed-up training set of all our NUPA tasks. We find only 800 steps training (about 50M training samples, $\ll 1$ epoch) can improve the performance significantly, as shown in Figure 2 labeled as "`Llama-8B-ft`". Though Llama-

Table 6: Performance of RF CoT. "-" means exceeding context window limitation (2k tokens).

Exact Match	Add Float			Multiply Fraction			Max Scientific			Mod Integer		
# Digit	5	6	7	2	3	4	38	39	40	6	7	8
RF CoT	**1.00**	**1.00**	-	**0.90**	**0.85**	-	1.00	0.85	0.90	**0.70**	**0.35**	-
GPT-4o	0.78	0.66	0.49	0.53	0.20	0.00	0.37	0.46	0.36	0.01	0.00	0.00
Qwen2-72B	0.62	0.50	0.70	0.05	0.00	0.00	0.96	**0.98**	**0.95**	0.03	0.00	0.00

3.1-8B is not a strong baseline, this finetuned version achieves much better performance. In max, max-hard, add-float and turediv tasks, this model can beat or exceed GPT-4o. This confirms our hypothesis: for many NUPA tasks, the model's capabilities may not be the main limiting factor, but rather the lack of numerical diversity and task variety in the training data constrains performance. However, we also found that such finetuning does not provide much improvement on certain tasks, such as understanding digits.

However, when we attempted to incorporate various techniques, such as modifying the model's original PEs, tokenizer, or the representation of inputting numbers, into an already trained model, we find that these approaches were ineffective. Whether we altered the PE or adjusted the model's tokenization and representation, it significantly disrupted the model's original behavior, leading to a substantial drop in performance. In such cases, finetuning was unable to restore the model's original performance, let alone improve it. This suggests that enhancing a model's NUPA capabilities post-training may require more revolutionary innovations beyond the current research paradigms. The results of these attempts are presented in Appendix B.5.

4 Is CoT suitable and valid for NUPA?

Due to the task and representation diversity of our benchmark, it is difficult for one form of CoT to cover all issues. So here we adapt a special CoT form called Rule-Following CoT (Hu et al., 2024), where the LLMs are taught to follow a code or pseudo-code that describes the procedure to solve the task. Because all the steps can be converted into recurrences and some basic unit operations, the RF-CoT can be used to solve any problem solvable by code, which is suitable for our benchmark.

Table 5: Average inference time.

	batchsize	sec / sample
RF CoT	128	5.625
Direct	128	0.371
Direct	256	0.336

To evaluate the performance of the CoT method on the NUPA test, we finetuned the LLaMA 3.1-8B model on a subset of the NUPA test, using the rule-following finetuning (RFFT). During both training and testing, we set a context window of 2000 tokens, with any data exceeding this limit being ignored. The performance on selected tasks is presented in Table 6. Within the context length limit, the rule-following finetuned LLaMA 3.1-8B significantly outperformed GPT-4o and Qwen2-72B on average. However, although CoT methods are quite effective within a certain data range, their design requiring a longer context window for more complex operations leads to significantly slower reasoning times as input size increases. This is particularly crucial in mathematical computations. As shown in Table 6, with the 2000-token limit, CoT can only handle fraction addition involving numbers up to three digits. The more complex the logic required for the computation, the smaller the range of numbers CoT can process. We provide the maximal digit length within the 2k context window limitation for each task in Appendix C to show the context window limitation for complex tasks. As for inference time, Table 5 demonstrates the average inference time for generating each sample using "RF CoT" and "direct answer" during the NUPA test where both experiments are operated on an A800 GPU. The batchsize 128 is the same batchsize while 256 shares a similar CUDA memory. The RF CoT method is approximately 17 times slower than directly generating the answer, causing an unsustainable burden for such a basic operation that is frequently encountered in solving real-world problems.

5 Conclusion & Limitation

We investigate NUPA of LLMs, propose a comprehensive benchmark called NUPA test and reveal that numerical problems remain challenging for modern LLMs. Our comprehensive test suite, which includes a variety of numerical representations and tasks, has exposed the surprising vulnerability of LLMs in this fundamental area. Our analysis includes techniques proposed in previous work that significantly influence the NUPA of LLMs. finetuning models on NUPA Test do improve their performance. However, extra tricks harm NUPA in the finetuning process, which indicates that these methods are not readily transferable to enhance a pre-trained model. Additionally, we have explored the potential of chain-of-thought techniques to improve NUPA.

While our work marks a preliminary step towards enhancing NUPA in LLMs, a definitive solution to the challenges faced by LLMs has not yet been found, which is left for future work.

Reproducibility Statement

We have made every effort to ensure that the results presented in this paper are fully reproducible. Detailed descriptions of the number formats, construction and metrics of our NUPA dataset are provided in 2 and A.1.2, and examples for each task in B.4. To further facilitate reproducibility, we have incorporated the full source code, enabling the generation of the entire datasets and the training and assessment of models, within the supplementary materials. Researchers wishing to generate NUPA benchmark or replicate our experiments can refer to these resources for all necessary information.

Ethics Statement

In conducting this research, we have adhered to the highest ethical standards to ensure the integrity and fairness of our work. For source code releases, we have ensured compliance with applicable legal standards, ensuring that the code is anonymized and free from personally identifiable information. During the construction of the dataset, all data was entirely generated randomly, without including any personal identity information or other private data of individuals.

References

Hanseul Cho, Jaeyoung Cha, Pranjal Awasthi, Srinadh Bhojanapalli, Anupam Gupta, and Chulhee Yun. Position coupling: Leveraging task structure for improved length generalization of transformers, 2024. URL https://arxiv.org/abs/2405.20671.

Joseph Dauben. The universal history of numbers and the universal history of computing. *Notices of the AMS*, 49(1), 2002.

Hank Davis and John Memmott. Counting behavior in animals: A critical evaluation. *Psychological Bulletin*, 1982.

Adi Haviv, Ori Ram, Ofir Press, Peter Izsak, and Omer Levy. Transformer language models without positional encodings still learn positional information. In Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2022*, pp. 1382–1390, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-emnlp.99. URL https://aclanthology.org/2022.findings-emnlp.99.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Leng Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems, 2024. URL https://arxiv.org/abs/2402.14008.

Yi Hu, Xiaojuan Tang, Haotong Yang, and Muhan Zhang. Case-based or rule-based: How do transformers do the math?, 2024. URL https://arxiv.org/abs/2402.17709.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao, Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. Mixtral of experts, 2024. URL https://arxiv.org/abs/2401.04088.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers, 2023a. URL https://arxiv.org/abs/2305.19466.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers, 2023b. URL https://arxiv.org/abs/2305.19466.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. Teaching arithmetic to small transformers, 2023. URL https://arxiv.org/abs/2307.03381.

Meta. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

OpenAI. Gpt-4 technical report, 2023. URL https://arxiv.org/abs/2303.08774.

OpenAI. Gpt-4o system card, 2024.

Ofir Press, Noah A. Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation, 2022. URL https://arxiv.org/abs/2108.12409.

Alibaba Group Qwen Team. Qwen2 technical report, 2024. URL https://arxiv.org/abs/2407.10671.

Ashutosh Sathe, Divyanshu Aggarwal, and Sunayana Sitaram. Improving self consistency in llms through probabilistic tokenization, 2024. URL https://arxiv.org/abs/2407.03678.

Ruoqi Shen, Sébastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. Positional description matters for transformers arithmetic, 2023. URL https://arxiv.org/abs/2311.14737.

David Eugene Smith and Louis Charles Karpinski. *The hindu-arabic numerals*. Ginn, 1911.

Mark S. Strauss and Lynne E. Curtis. Infant perception of numerosity. *Child Development*, 1981.

Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2023. URL https://arxiv.org/abs/2104.09864.

Zhengyang Tang, Xingxing Zhang, Benyou Wang, and Furu Wei. Mathscale: Scaling instruction tuning for mathematical reasoning, 2024. URL https://arxiv.org/abs/2403.02884.

An Yang, Beichen Zhang, Binyuan Hui, Bofei Gao, Bowen Yu, Chengpeng Li, Dayiheng Liu, Jianhong Tu, Jingren Zhou, Junyang Lin, Keming Lu, Mingfeng Xue, Runji Lin, Tianyu Liu, Xingzhang Ren, and Zhenru Zhang. Qwen2.5-math technical report: Toward mathematical expert model via self-improvement, 2024. URL https://arxiv.org/abs/2409.12122.

Xiaotian Zhang, Chunyang Li, Yi Zong, Zhengyu Ying, Liang He, and Xipeng Qiu. Evaluating the performance of large language models on gaokao benchmark, 2024. URL https://arxiv.org/abs/2305.12474.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. What algorithms can transformers learn? a study in length generalization, 2023. URL https://arxiv.org/abs/2310.16028.

Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly, 2024a. URL https://arxiv.org/abs/2402.09371.

Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. Transformers can achieve length generalization but not robustly, 2024b. URL https://arxiv.org/abs/2402.09371.

A Appendix

A.1 NUPA Test

A.1.1 Examples for each task

We provide each tasks with an examples. To test the models, we also add some model specific system messages like "You are a helpful assistant to process numbers. Please directly answer the question after the =". The context before "=" is the question and the context after "=" is the groundtruth and be removed when testing.

- Add-Integer: Add two numbers: $744 + 543 = 1287$
- Add-Float: Add two numbers: $93.81 + 9.976 = 103.786$
- Add-Fraction: Add two numbers: $3/8 + 2/5 = 31/40$
- Add-Scientific: Add two numbers: $9.92e16 + 9.731e18 = 9.8302e18$
- Sub-Integer: Subtract two numbers: $744 - 543 = 201$
- Sub-Float: Subtract two numbers: $93.81 - 9.976 = 83.834$
- Sub-Fraction: Subtract two numbers: $2/5 - 3/8 = 1/40$
- Sub-Scientific: Subtract two numbers: $9.731e38 - 9.92e36 = 9.6318e38$
- Multiply-Integer: Multiply two numbers: $968 \times 8 = 7744$
- Multiply-Float: Multiply two numbers: $8.4 \times 9.555 = 80.262$
- Multiply-Fraction: Multiply two numbers: $8/7 \times 5/2 = 20/7$
- Multiply-Fraction: Multiply two numbers: $9.92e16 \times 9.731e38 = 9.653152e55$
- Truediv-Integer: Divide two numbers and return the result as a fraction. $744 \ / \ 543 = 248/181$
- Truediv-Fraction: Divide two numbers and return the result as a fraction. $(3/8) \ / \ (2/5) = 15/16$
- Floordiv-Integer: Divide two numbers and return the result as an integer. $845 \ // \ 152 = 5$
- Mod-Integer: Divide two numbers and return the remainder. $845 \ \% \ 152 = 85$
- Max-Integer: Get the maximal number: 50404 and $97871 = 97871$
- Max-Float: Get the maximal number: 44.418 and $65.669 = 65.669$
- Max-Fraction: Get the maximal number: $3/5$ and $3/8 = 3/5$
- Max-Scientific: Get the maximal number: $8.15e64$ and $1.063e73 = 1.063e73$
- Digit_max-Integer: Compare two numbers digit by digit and return the larger digit at each position, treating any missing digits as 0. 50194 and $14283 = 54294$
- Digit_max-Float: Compare two numbers digit by digit and return the larger digit at each position, treating any missing digits as 0. 35.905 and $8.4 = 38.905$
- Digit_add-Integer: The task is to add two given numbers digit by digit and return the result modulo 10 (ignoring carry), treating any missing digits as 0. 50404 digit add $97871 = 47275$
- Digit_add-Float: The task is to add two given numbers digit by digit and return the result modulo 10 (ignoring carry), treating any missing digits as 0. 44.418 digit add $65.669 = 9.077$
- Get_digit-Integer: Get the digit at the given position (from left to right, starting from 0). 50404 at position $4 = 4$
- Get_digit-Float: Get the digit at the given position (from left to right, starting from 0). 44.418 at position $3 = 1$
- Length-Integer: The total number of digits of $50404 = 5$
- Length-Float: The total number of digits of $262.534 = 6$
- Count-Integer: Count the number of the given digit in the given number: 27422 count the occurance time of digit $2 = 3$
- To_float-Fraction: Convert the number to float: $9/5 = 1.8$
- To_float-Scientific: Convert the number to float: $8.538e2 = 853.8$
- To_scientific-Integer: Convert the number to scientific notation: $50400 = 5.04e4$
- To_scientific-Float: Convert the number to scientific notation: $262.534 = 2.62534e2$
- Sig.Fig-Integer: Convert the number to scientific notation: 50194 and keep significant figures as $3 = 5.02e4$
- Sig.Fig-Float: Convert the number to scientific notation: 65.669 and keep significant figures as $2 = 6.6e1$

A.1.2 NON-INCLUDED TASKS

We exclude some compositions between number representations and tasks because of the following three reasons:

- ✗ too complex. We exclude the truediv between float and scientific. Division between float numbers is difficult to define accurately in our scenario. It is very common to divide two floating point numbers into an infinite decimal, which means that even very short decimals can still result in a very long and unpredictable result after division. And in this task we do not want to discuss the case of rounding the result. (This is another task of ours.) For the same reason, we also exclude division in scientific notation.
- ◯: can be easily transferred to from an included task.
  - Converting fractions to scientific notation can be done by first converting to a float. (Fraction-to_scientific = Fraction-to_float + Float-to_scientific). Fraction-SiginificantFigure is similar.
  - Scientific notation retains significant digits and is virtually identical to floating point numbers.
  - count is a special task where we just consider a nubmer as "a set of digits" so count in a float, fraction and scientific notation is as the same as in a integer.
- −: not applicable.
  - In fraction and scientific notation, the digit concept is not well-defined so the tasks about digit (digit-compare, digit-add, get-digit and length) are not applicable.
  - Floordiv and mod is only defined on integer.
  - Integer and float do not need to be further converted to float. Similarly, scientific has no need to converted to scientific.

A.1.3 EASY/HARD SPLIT OF NUPA TASKS

We divide the tasks into easy and hard as shown in Table 7, where the hard tasks marked as H with maximal test digit as 20 and the easy tasks marked as E with maximal test digit as 100.

Table 7: Tasks can be divided into Easy and Hard.

	Add	Sub	Multiply	Truediv	Floordiv	Mod	Max	Min	Digit Max	Digit Min	Digit Add	Get Digit	Length	Count	To Float	To Scientific	Sig. Fig.
Integer	H	H	H	H	H	H	E	E	E	E	E	E	E	E		E	E
Float	H	H	H				E	E	E	E	E	E	E			E	E
Fraction	H	H	H	H			H	H							H		
Scientific	H	H	H				E	E							E		

A.1.4 PREPROCESS FOR NUPA TASKS

After we select two random numbers, we have some pre-procession to generate the final questions:

- For "Multiply", the difficulty also affected by the shorter number severely, so we split the task into two sub-tasks as "Multiply-hard" and "multiply-easy". For hard subset, we require that the shorter number must be longer than half of the longer one. For easy subset, we require that the length of the shorter number is less than 3, so that the complexity is $O(n)$ instead of $O(n^2)$. And because the addition of fraction also involves multiplication, we also add a add-easy for this task in a same way.
- For "max" and "min" tasks, we additionally provide a harder version. For Integers and floats, we make the compared two numbers share the same length. At the same time, they should have more digit as the same like 12949 and 12961 to avoid models can solve the problem by only counting the length or compare the first digit. For scientific notation, we make sure 70% pairs of compared numbers with the same exponential part so that model cannot directly get the answer without compare the mantissa part. For fraction, we make sure the numbers are both less than one, avoiding model can just compare them with 1 to get more than 50% accuracy.

- For "to_float-Fraction", we require the fraction must can be convert into a finite decimal, that is the denominator contains only factors 2 and 5.

- For "add/sub-Scientific", we require the exponential part of each number has difference less than 5 that make sure the generated answer will not too long.

B TOKENIZER, PE AND DATA FORMATS

B.1 FULL TEST RESULTS OF LLMS

See Figures 6, 7, 8 and Table 8.

B.2 TOKENIZATION

We experiment on models of 3 different size, including 0.1B, 0.9B and 3B. For the 0.1B and 0.9B models, we train them on integer addition of 1-8 digits; for the 3B model, we train it on the same task of 1-40 digits.

Figure 9 illustrates the in-domain performance of these three models in the first three columns and their out-of-domain (OOD) performance in the last two columns. Here we use the exact match metric. In our experiments of the 0.1B and 0.9B models, the 1-digit and the 2-digit tokenizer demonstrate comparable performance in the in-domain test, while the 1-digit tokenizer exceeds the others to a large extent in length generalization. In contrast, the 3-digit tokenizer exhibits poor performance in both in-domain and out-of-domain evaluations. Tokenizers with an increasing number of digits significantly hinder subbillion models' NUPA. In the experiments of the 3B model, the 2-digit tokenizer matches the 1-digit tokenizer in both in-domain and OOD performance. In addition, the 3-digit tokenizer shows the potential in length generalization for the first time, yet its performance remains inferior to that of the smaller tokenizers. This indicates that scaling up the model size indeed alleviate the challenges in developing NUPA caused by larger tokenizers. Nevertheless, larger tokenizers do not present any distinct benefits in either in-domain or out-of-domain generalization in both small and large models.

B.3 PES

We show exact match, digit match and dlength of 100M models trained with various PE, including RoPE, NoPE and Alibi in Figure 10. The shadowed areas denote in-domain length range.

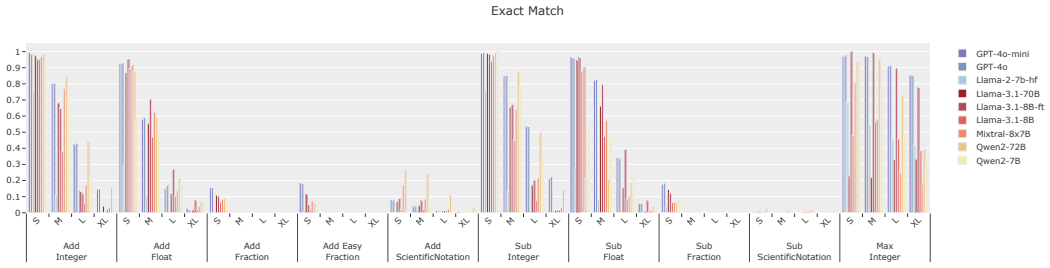
B.4 DATA FORMATS

We provide the evaluation curves of compositions of reverse formatting, zero padding and index hints in Figure 13, Figure 12, Figure 13, Figure 14 and Figure 15. We experiment on 0.1B models trained on 1- to 8- digit training samples. Here we all use the exact match metric.

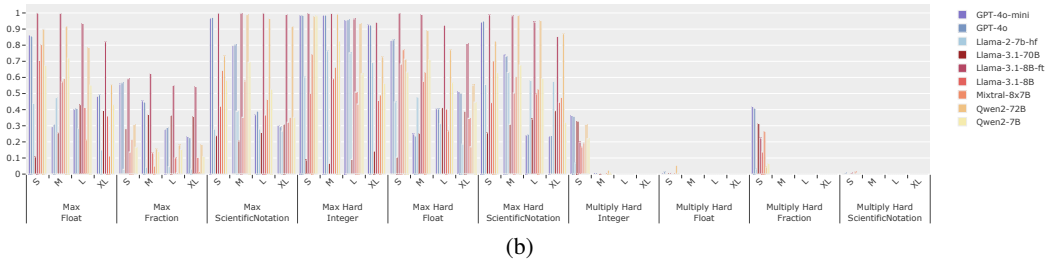
About the experiments of index hint, we show in Table 9.

B.5 NUPA FINTUNING WITH PE, TOKENIZER AND REPRESENTATION MODIFICATION

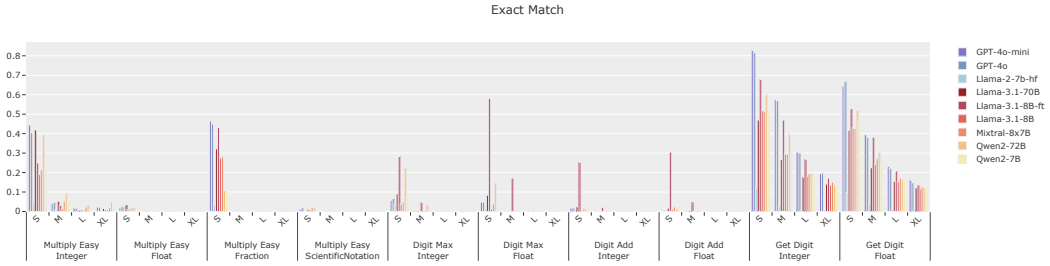
We show parts of results of our attempt to finetune a Llama-3.1-8B model with PE, tokenizer and data format modification in Table 10. All the checkpoint we select by the lowest valid loss. No one can outperform the naive finetuning or the original Llama.



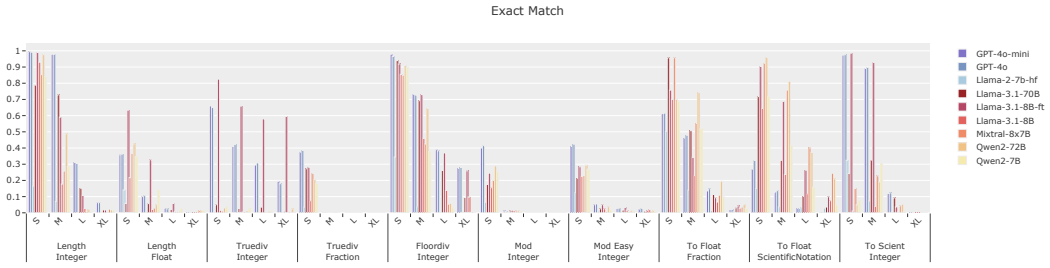
(a)



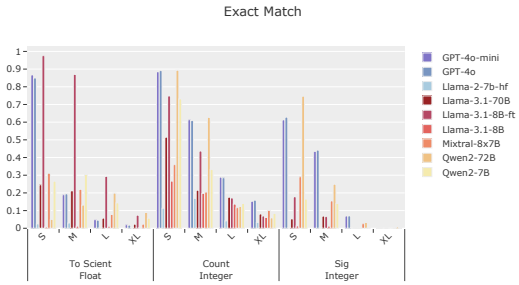
(b)



(c)

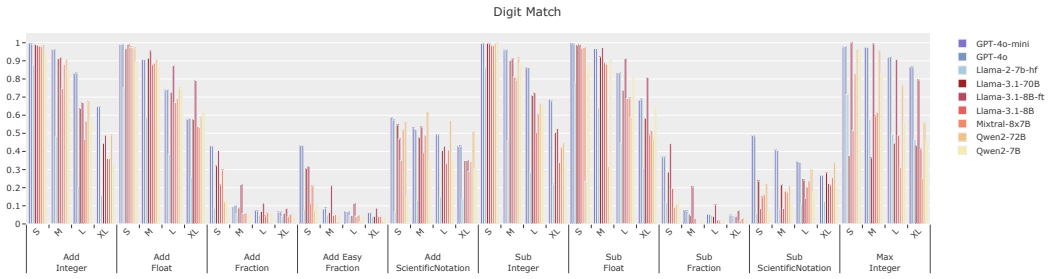


(d)

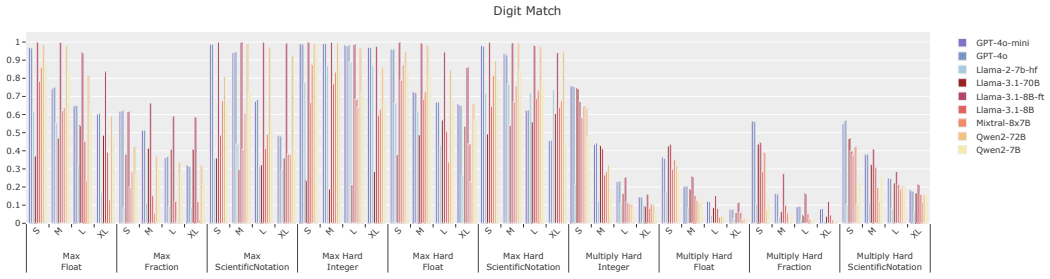


(e)

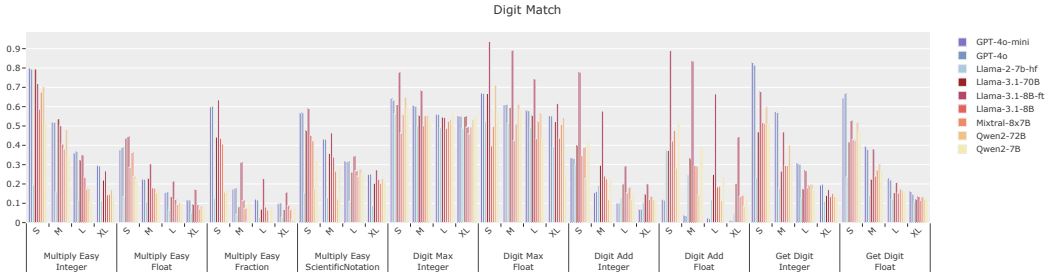
Figure 6: Exact match of models tested on NUPA Test.



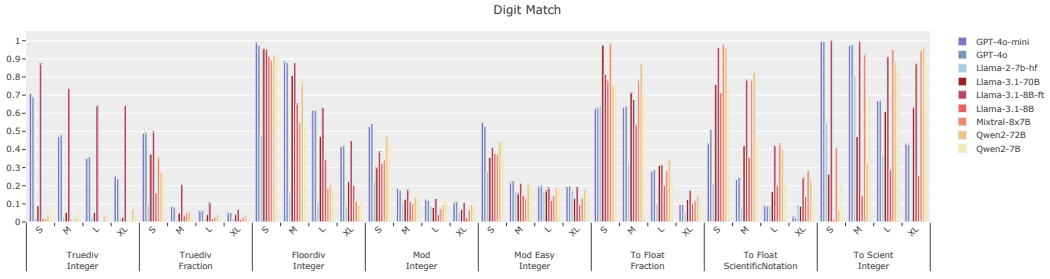
(a)



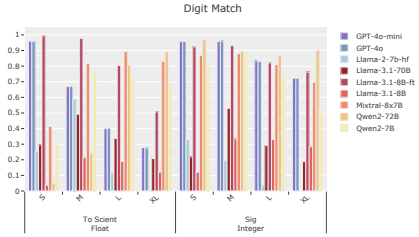
(b)



(c)



(d)



(e)

Figure 7: Digit match of models tested on NUPA Test.

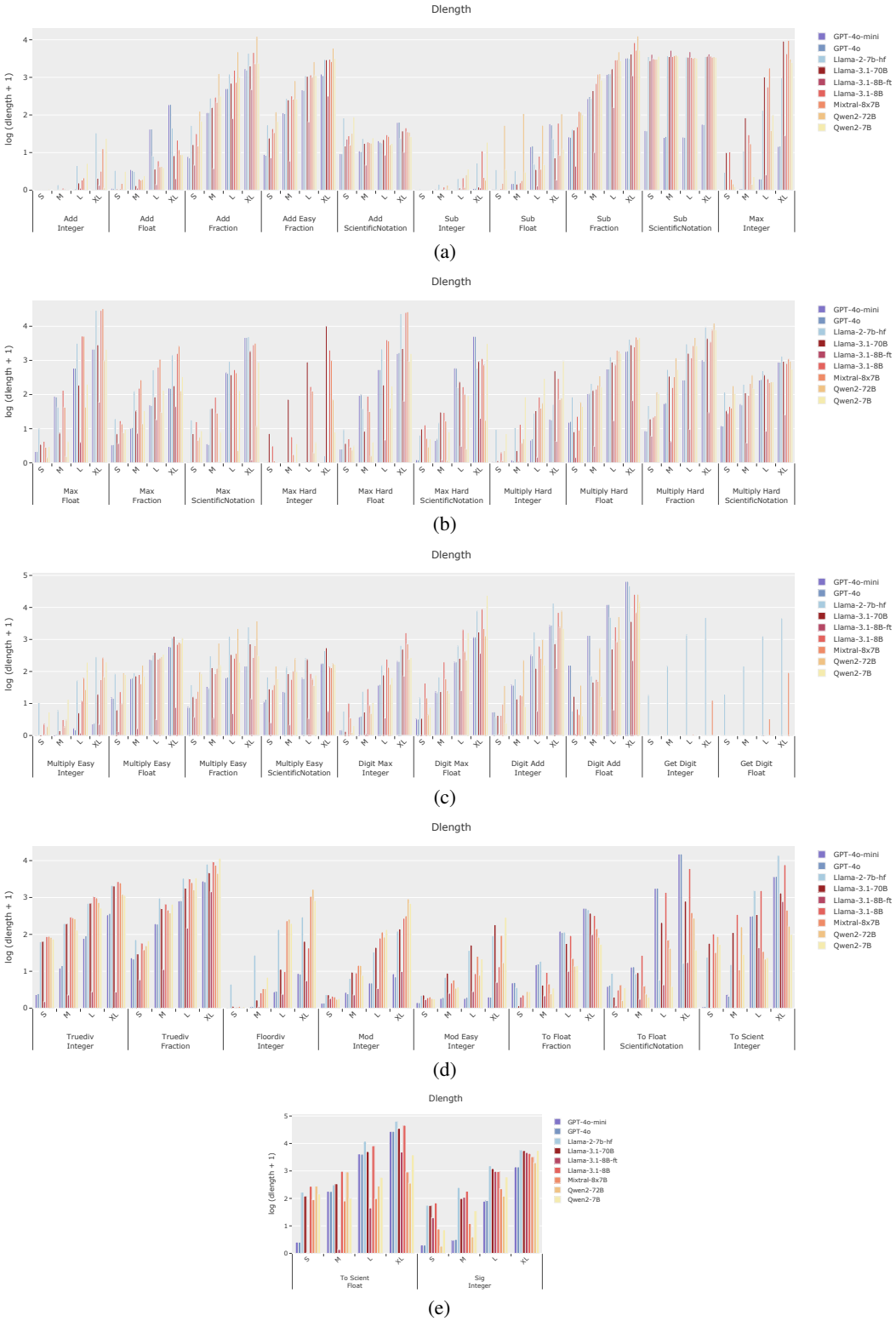


Figure 8: Dlength of models tested on NUPA Test. Note that we use $\log(\text{dlength} + 1)$ as the ylabel in the figure.

	Add Int	Add Float	Add Frac	Add Easy Frac	Add Sci	Sub Int	Sub Float	Sub Frac
GPT-4o-mini	5 / 20	4 / 11	0 / 1	0 / 2	0 / 0	6 / 20	5 / 15	0 / 1
GPT-4o	5 / 20	4 / 11	0 / 1	0 / 1	0 / 0	6 / 20	4 / 15	0 / 1
Qwen2-72B	6 / 20	0 / 15	0 / 1	0 / 1	0 / 11	6 / 20	0 / 15	0 / 1
Qwen2-7B	4 / 14	0 / 15	0 / 1	0 / 0	0 / 7	4 / 14	0 / 13	0 / 1
Llama-3.1-8B-ft	4 / 12	5 / 17	0 / 1	0 / 1	0 / 4	4 / 12	4 / 17	0 / 2
Llama-3.1-70B	4 / 15	0 / 11	0 / 1	0 / 1	0 / 0	6 / 11	4 / 11	0 / 1
Llama-3.1-8B	4 / 9	3 / 11	0 / 1	0 / 0	0 / 0	3 / 10	3 / 10	0 / 1
Mixtral-8x7B	5 / 10	4 / 11	0 / 1	0 / 1	0 / 6	4 / 15	3 / 11	0 / 1
Llama-2-7b-hf	0 / 6	0 / 7	0 / 0	0 / 0	0 / 0	0 / 6	0 / 5	0 / 1

	Sub Sci	Max Int	Max Float	Max Frac	Max Sci	Max Hard Int	Max Hard Float	Max Hard Sci
GPT-4o-mini	5 / 20	4 / 11	0 / 1	0 / 2	0 / 0	6 / 20	5 / 15	0 / 1
GPT-4o	5 / 20	4 / 11	0 / 1	0 / 1	0 / 0	6 / 20	4 / 15	0 / 1
Qwen2-72B	6 / 20	0 / 15	0 / 1	0 / 1	0 / 11	6 / 20	0 / 15	0 / 1
Qwen2-7B	4 / 14	0 / 15	0 / 1	0 / 0	0 / 7	4 / 14	0 / 13	0 / 1
Llama-3.1-8B-ft	4 / 12	5 / 17	0 / 1	0 / 1	0 / 4	4 / 12	4 / 17	0 / 2
Llama-3.1-70B	4 / 15	0 / 11	0 / 1	0 / 1	0 / 0	6 / 11	4 / 11	0 / 1
Llama-3.1-8B	4 / 9	3 / 11	0 / 1	0 / 0	0 / 0	3 / 10	3 / 10	0 / 1
Mixtral-8x7B	5 / 10	4 / 11	0 / 1	0 / 1	0 / 6	4 / 15	3 / 11	0 / 1
Llama-2-7b-hf	0 / 6	0 / 7	0 / 0	0 / 0	0 / 0	0 / 6	0 / 5	0 / 1

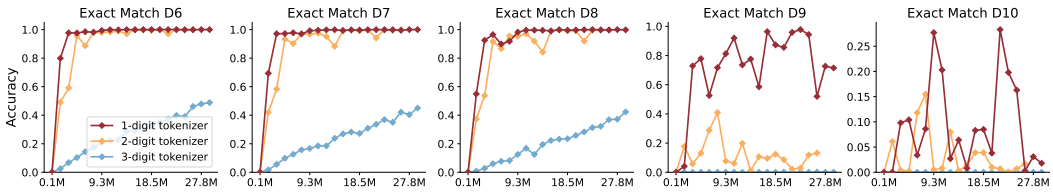
	Multiply Hard Int	Multiply Hard Float	Multiply Hard Frac	Multiply Hard Sci	Multiply Easy Int	Multiply Easy Float	Multiply Easy Frac	Multiply Easy Sci
GPT-4o-mini	5 / 20	4 / 11	0 / 1	0 / 2	0 / 0	6 / 20	5 / 15	0 / 1
GPT-4o	5 / 20	4 / 11	0 / 1	0 / 1	0 / 0	6 / 20	4 / 15	0 / 1
Qwen2-72B	6 / 20	0 / 15	0 / 1	0 / 1	0 / 11	6 / 20	0 / 15	0 / 1
Qwen2-7B	4 / 14	0 / 15	0 / 1	0 / 0	0 / 7	4 / 14	0 / 13	0 / 1
Llama-3.1-8B-ft	4 / 12	5 / 17	0 / 1	0 / 1	0 / 4	4 / 12	4 / 17	0 / 2
Llama-3.1-70B	4 / 15	0 / 11	0 / 1	0 / 1	0 / 0	6 / 11	4 / 11	0 / 1
Llama-3.1-8B	4 / 9	3 / 11	0 / 1	0 / 0	0 / 0	3 / 10	3 / 10	0 / 1
Mixtral-8x7B	5 / 10	4 / 11	0 / 1	0 / 1	0 / 6	4 / 15	3 / 11	0 / 1
Llama-2-7b-hf	0 / 6	0 / 7	0 / 0	0 / 0	0 / 0	0 / 6	0 / 5	0 / 1

	Digit Max Int	Digit Max Float	Digit Add Int	Digit Add Float	Get Digit Int	Get Digit Float	Length Int	Length Float
GPT-4o-mini	5 / 20	4 / 11	0 / 1	0 / 2	0 / 0	6 / 20	5 / 15	0 / 1
GPT-4o	5 / 20	4 / 11	0 / 1	0 / 1	0 / 0	6 / 20	4 / 15	0 / 1
Qwen2-72B	6 / 20	0 / 15	0 / 1	0 / 1	0 / 11	6 / 20	0 / 15	0 / 1
Qwen2-7B	4 / 14	0 / 15	0 / 1	0 / 0	0 / 7	4 / 14	0 / 13	0 / 1
Llama-3.1-8B-ft	4 / 12	5 / 17	0 / 1	0 / 1	0 / 4	4 / 12	4 / 17	0 / 2
Llama-3.1-70B	4 / 15	0 / 11	0 / 1	0 / 1	0 / 0	6 / 11	4 / 11	0 / 1
Llama-3.1-8B	4 / 9	3 / 11	0 / 1	0 / 0	0 / 0	3 / 10	3 / 10	0 / 1
Mixtral-8x7B	5 / 10	4 / 11	0 / 1	0 / 1	0 / 6	4 / 15	3 / 11	0 / 1
Llama-2-7b-hf	0 / 6	0 / 7	0 / 0	0 / 0	0 / 0	0 / 6	0 / 5	0 / 1

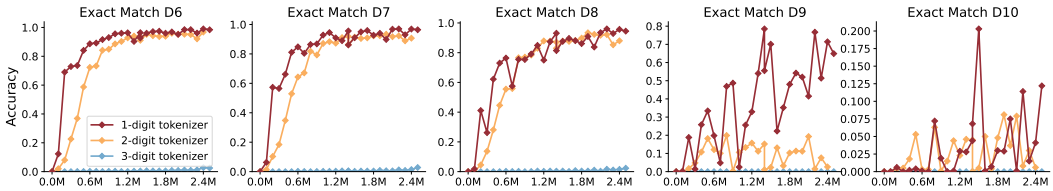
	Truediv Int	Truediv Frac	Floordiv Int	Mod Int	Mod Easy Int	To Float Frac	To Float Sci	To Scient Int
GPT-4o-mini	5 / 20	4 / 11	0 / 1	0 / 2	0 / 0	6 / 20	5 / 15	0 / 1
GPT-4o	5 / 20	4 / 11	0 / 1	0 / 1	0 / 0	6 / 20	4 / 15	0 / 1
Qwen2-72B	6 / 20	0 / 15	0 / 1	0 / 1	0 / 11	6 / 20	0 / 15	0 / 1
Qwen2-7B	4 / 14	0 / 15	0 / 1	0 / 0	0 / 7	4 / 14	0 / 13	0 / 1
Llama-3.1-8B-ft	4 / 12	5 / 17	0 / 1	0 / 1	0 / 4	4 / 12	4 / 17	0 / 2
Llama-3.1-70B	4 / 15	0 / 11	0 / 1	0 / 1	0 / 0	6 / 11	4 / 11	0 / 1
Llama-3.1-8B	4 / 9	3 / 11	0 / 1	0 / 0	0 / 0	3 / 10	3 / 10	0 / 1
Mixtral-8x7B	5 / 10	4 / 11	0 / 1	0 / 1	0 / 6	4 / 15	3 / 11	0 / 1
Llama-2-7b-hf	0 / 6	0 / 7	0 / 0	0 / 0	0 / 0	0 / 6	0 / 5	0 / 1

	To Scient Float	Count Int	Sig Int
GPT-4o-mini	5 / 20	4 / 11	0 / 1
GPT-4o	5 / 20	4 / 11	0 / 1
Qwen2-72B	6 / 20	0 / 15	0 / 1
Qwen2-7B	4 / 14	0 / 15	0 / 1
Llama-3.1-8B-ft	4 / 12	5 / 17	0 / 1
Llama-3.1-70B	4 / 15	0 / 11	0 / 1
Llama-3.1-8B	4 / 9	3 / 11	0 / 1
Mixtral-8x7B	5 / 10	4 / 11	0 / 1
Llama-2-7b-hf	0 / 6	0 / 7	0 / 0

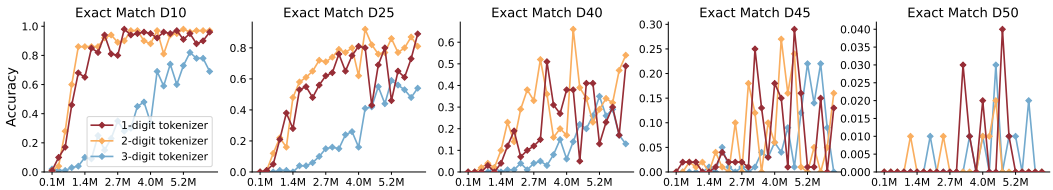
Table 8: Well-learned digits / performance-preserving digits of models tested on NUPA Test.



(a) 0.1B model int add



(b) 0.9B model int add



(c) 3B model int add

Figure 9: Accuracy of models of 0.1B, 0.9B and 3B parameters trained with 1-3 digit tokenizer on the task of integer addition.

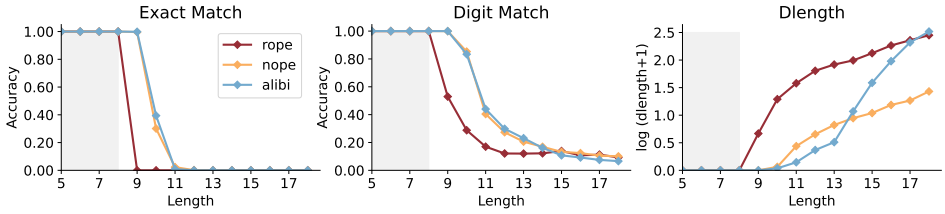


Figure 10: Exact match, digit match and dlength of 100M models trained with various PE, including RoPE, NoPE and Alibi. The shadowed areas denote in-domain length range.

Table 9: Exact match of 0.1B models trained on integer addition, multiply and maximum respectively with various compositions of reverse formatting and index hints.

	Integer Addition				Integer Multiply				Integer Max			
	rev	rev + idx	no	idx	rev	reverse + idx	no	idx	reverse only	reverse + idx	no	idx
d9	**1.00**	0.93	0.98	0.41	**0.43**	0.00	0.13	0.00	**1.00**	0.99	**1.00**	0.99
d10	**0.80**	0.06	0.32	0.01	**0.13**	0.02	0.04	0.02	**1.00**	0.97	**1.00**	0.98

Table 10: Finetining with PE, data format, and tokenizer modification will degrade the performance. The first two lines are a naive finetuned Llama and the original Llama without finetuning, which are the baseline. All the checkpoint we select by the lowest valid loss. "wl-digit" is used to denote well-learned digit; "pp-digit" is used to denote performance-preserving digit.

	Add Integer					
	S	M	L	XL	wl-digit	pp-digit
FT	**0.95**	**0.65**	**0.12**	0.01	**4**	**12**
without FT	**0.95**	0.38	0.06	**0.02**	**4**	9
NoPE +reverse	0.89	0.35	0.06	**0.02**	3	9
NoPE +reverse +pad	0.87	0.34	0.05	**0.02**	0	9
RoPE +reverse +pad	0.40	0.20	0.04	0.00	0	7

	Add Float					
	S	M	L	XL	wl-digit	pp-digit
FT	**0.96**	**0.71**	**0.27**	**0.08**	**5**	**17**
without FT	0.90	0.47	0.10	0.02	3	11
NoPE +reverse	0.81	0.38	0.09	0.01	0	11
NoPE +reverse +pad	0.74	0.38	0.06	0.01	0	9
RoPE +reverse +pad	0.35	0.30	0.09	0.02	0	11

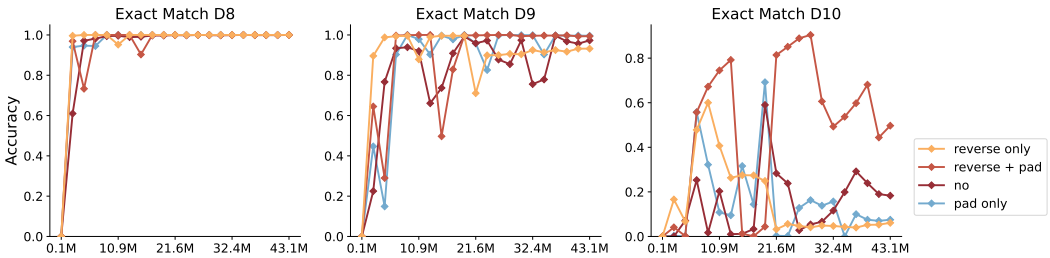


Figure 11: Exact match of 0.1B models trained on 1- to 8- digit integer addition with different compositions of reverse formatting and zero padding on 8- to 10- digit tests.

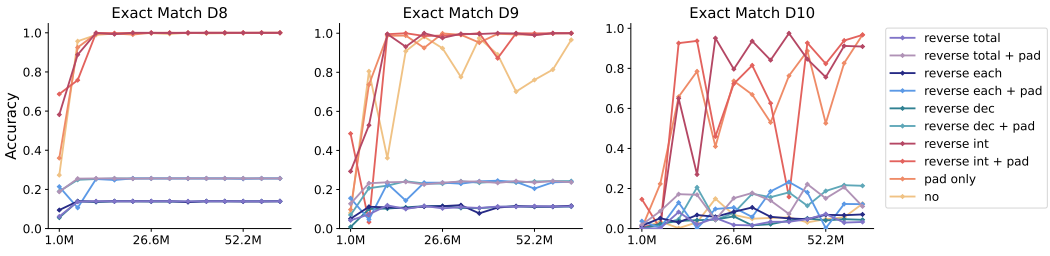


Figure 12: Exact match of 0.1B models trained on 1- to 8- digit float addition with different compositions of reverse formatting and zero padding on 8- to 10- digit tests.

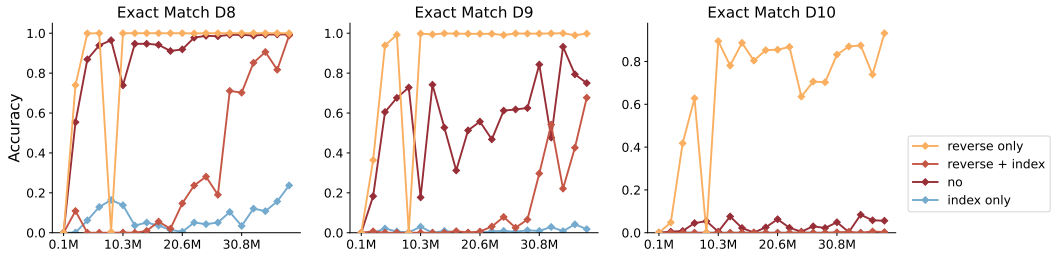


Figure 13: Exact match of 0.1B models trained on 1- to 8- digit integer addition with different compositions of reverse formatting and index hints on 8- to 10- digit tests.

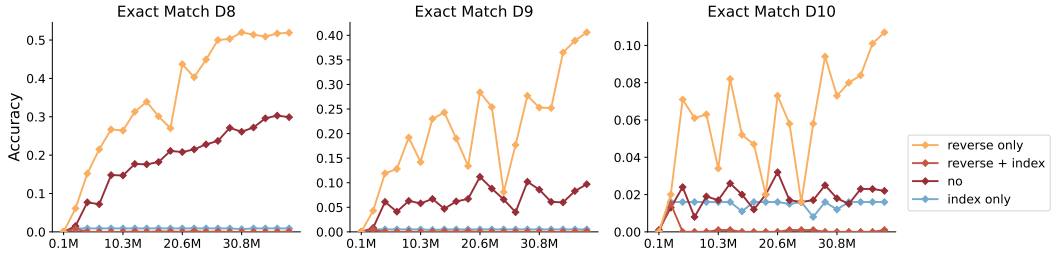


Figure 14: Exact match of 0.1B models trained on 1- to 8- digit integer multiplication with different compositions of reverse formatting and index hints on 8- to 10- digit tests.

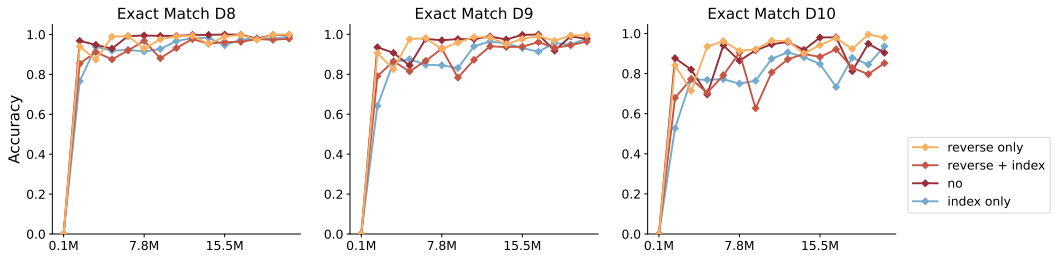


Figure 15: Exact match of 0.1B models trained on 1- to 8- digit integer maximum with different compositions of reverse formatting and index hints on 8- to 10- digit tests.

C RULE-FOLLOWING CHAIN-OF-THOUGHT

The selective tasks used to train the RFFT are shown in Table 11 and we also report the maximal length within 2k tokens context windows limitation. For the detailed prompt of these tasks we cannot put them into papers so we include them in supplementary.

Table 11: Maximum length of each task that 2k context window can afford with RF-CoT

	Add	Sub	Multiply	Floordiv	Mod	Max	DigitMax	GetDigit	Length
Integer	20	20	12	20	6	100	17	100	34
Float	6	5	4	-	-	50	-	100	-
Fraction	3	2	3	-	-	20	-	-	-
Scientific	3	3	3	-	-	100	-	-	-