How Numerical Precision Affects Mathematical Reasoning Capabilities of LLMs

Anonymous ACL submission

Abstract

Despite the remarkable success of Transformer-based Large Language Models (LLMs) across various domains, understanding and enhancing their mathematical capabilities remains a significant challenge. In this paper, we conduct a rigorous theoretical analysis of LLMs' mathematical abilities, with a specific focus on their arithmetic performances. We identify numerical precision as a key factor that influences their effectiveness in mathematical tasks. Our results show that Transformers operating with low numerical precision fail to address arithmetic tasks, such as iterated addition and integer multiplication, unless the model size grows super-polynomially with respect to the input length. In contrast, Transformers with standard numerical precision can efficiently handle these tasks with significantly smaller model sizes. We further support our theoretical findings through empirical experiments that explore the impact of varying numerical precision on arithmetic tasks, providing valuable insights for improving the mathematical reasoning capabilities of LLMs.

1 Introduction

Transformer-based Large Language Models (LLMs), such as GPT (OpenAI, 2023), Claude (Anthropic, 2024), and LLaMA (Dubey et al., 2024), have achieved impressive performance across a broad range of natural language tasks (Zhu et al., 2024; Basyal and Sanghvi, 2023; Shao et al., 2023). Despite the great success, significant challenges remain when applying LLMs to mathematical problem-solving. Unlike many typical NLP tasks, which often depend on pattern recognition and statistical correlations (Blei et al., 2003), mathematical reasoning requires rigorous logical deduction in a specific order (Frieder et al., 2024; Bubeck et al., 2023). To address these challenges, various strategies have been proposed, including carefully designed prompting strategies (Wei et al., 2022b; Yamauchi et al., 2023; Imani et al., 2023) and inference-based searching method (Kang et al., 2024; Wu et al., 2024a; Snell et al., 2024; Brown et al., 2024). However, a comprehensive understanding of the intrinsic limitations that restrict the mathematical reasoning capabilities of LLMs remains elusive.

In principle, mathematical reasoning, built on basic arithmetical operations, requires the accurate computation of intermediate results throughout the reasoning process (Bubeck et al., 2023; Lee et al., 2024). There exist works (Feng et al., 2023; Yang et al., 2024) exploring the arithmetic capabilities of LLMs with Chain of Thought prompting (Wei et al., 2022b). However, unlike the tokenization methods of modern LLMs (Dubey et al., 2024; OpenAI, 2023), where each digit in a number is tokenized individually, these works treat a whole number as one token. Under this assumption, each distinct number occupies a unique position in the vocabulary, leading to an essential mismatch with practical implementations. Moreover, recent studies have demonstrated that LLMs operating with reduced numerical precision (e.g., `int4`) exhibit a significant decline in performance on mathematical tasks (Jin et al., 2024; Marchisio et al., 2024).

In this paper, we provide a rigorous theoretical investigation of the arithmetical abilities of LLMs under the autoregressive paradigm. Specifically, we follow the tokenization method of modern LLMs, allowing the models to process and generate numbers digit by digit. Under these assumptions, we identify **numerical precision** as a key factor influencing their performance in arithmetical tasks. Our analysis focuses on three elementary arithmetic tasks: integer addition, iterated addition, and integer multiplication, which serve as elementary building blocks in solving complex real-world math problems.

To elucidate the role of numerical precision, we first examine the expressiveness of Transformers

Arithmetic Tasks	Standard Precision	Low Precision
Integer Addition $\mathrm{ADD}(n, p)$	Constant	$O(n^2)$
Iterated Addition $\mathrm{IterADD}(n, k, p)$	Constant	Super-polynomial
Integer Multiplication $\mathrm{Mul}(n, l, p)$	$O(n^2)$	Super-polynomial

Table 1: The model size *w.r.t.* the input size required for various arithmetic tasks on bounded-depth Transformers, under both standard and low numerical precision. Blue denotes the small model size, and red represents the large model size.

operating under low precision, such as `int8` and `int4`. We establish foundational impossibility results for low-precision Transformers, demonstrating that such models require super-polynomial size with respect to input length to solve iterated addition and integer multiplication (Theorems 4.2 and 4.3). Our proofs, grounded in complexity theory (Arora and Barak, 2009; Razborov, 1987), show that this limitation arises from the inability of individual neurons to store intermediate results during arithmetic computations. As a result, a significantly larger number of neurons is required to distribute the computation and avoid overflow.

We further demonstrate that increasing numerical precision is essential to overcoming this limitation: as the precision increases, the model size required to solve these arithmetic problems decreases dramatically. In particular, we prove that a bounded-depth Transformer with standard precision, such as `float32`, can solve all three tasks efficiently, requiring constant model size for integer and iterated addition (Theorems 5.1 and 5.2), and quadric size for integer multiplication (Theorem 5.3). At standard precision, it is sufficient for LLMs to accurately store relatively large numbers within individual neurons, enabling the Transformer to perform arithmetic tasks more effectively. These results not only underscore the critical role of numerical precision in enhancing the theoretical capabilities of LLMs for mathematical tasks but also offer practical insights. While low-precision models may provide computational efficiency, ensuring adequate numerical precision is essential for applications involving complex mathematical reasoning. A summary of our results is presented in Table 1.

In addition to theoretical analysis, we conduct extensive experiments to empirically validate our conclusions. First, we evaluate the performance of Transformers trained from scratch on the aforementioned arithmetic tasks, systematically examining how problem size and numerical precision impact their capabilities. Furthermore, we also conduct experiments on LLAMA 3.1 (Dubey et al., 2024) to evaluate the performance of these arithmetic tasks under different numerical precision. Our empirical results demonstrate that both low-precision and standard-precision Transformers perform adequately on the integer addition task. However, as task complexity increases—particularly in iterated addition and integer multiplication—the decrease in precision in Transformers results in significant performance degradation. These findings align with our theoretical predictions and offer practical guidance for enhancing LLM performance in mathematical reasoning tasks.

2 Preliminary

An autoregressive Transformer, or decoder-only Transformer (Radford et al., 2019; Dai et al., 2019), is a neural network designed to model sequence-to-sequence mappings. For an input sequence $\boldsymbol{s}$ of length n, each input token s_i (for $i \in [n]$) is transformed into a d-dimensional vector $\boldsymbol{x}_i^{(0)} = \mathrm{Embed}(s_i) + \boldsymbol{p}_i \in \mathbb{R}^d$, where $\mathrm{Embed}(\cdot)$ represents the token embedding function, and $\boldsymbol{p}_i$ denotes learnable positional embeddings. The model then consists of L Transformer blocks, each following the form:

$$\boldsymbol{h}_i^{(l)} = \boldsymbol{x}_i^{(l-1)} + \mathrm{Attn}^{(l)}\left(\boldsymbol{x}_i^{(l-1)}; \{\boldsymbol{x}_j^{(l-1)} : j \le i\}\right),$$

$$\boldsymbol{x}_i^{(l)} = \boldsymbol{h}_i^{(l)} + \mathrm{FFN}^{(l)}(\boldsymbol{h}_i^{(l)}),$$

where $l \in [L]$. Here, $\mathrm{Attn}^{(l)}$ and $\mathrm{FFN}^{(l)}$ denote the multi-head self-attention layer and the feed-forward network of the l-th Transformer block:

$$\mathrm{Attn}^{(l)}(\boldsymbol{x}, \mathcal{S}) = \sum_{h=1}^{H} \left(\boldsymbol{W}_{\mathrm{O}}^{(l,h)}\right)^{\top} \cdot \mathrm{H}^{(l,h)}(\boldsymbol{x}, \mathcal{S}),$$

$$\mathrm{H}^{(l,h)}(\boldsymbol{x}, \mathcal{S}) =$$

$$\mathrm{softmax}_{\boldsymbol{z} \in \mathcal{S}}\left((\boldsymbol{W}_{\mathrm{K}}^{(l,h)}\boldsymbol{z})^{\top}(\boldsymbol{W}_{\mathrm{Q}}^{(l,h)}\boldsymbol{x})\right)\boldsymbol{W}_{\mathrm{V}}^{(l,h)}\boldsymbol{z},$$

$$\mathrm{FFN}^{(l)}(\boldsymbol{x}) = \boldsymbol{W}_2^{(l)}\sigma(\boldsymbol{W}_1^{(l)}\boldsymbol{x}),$$

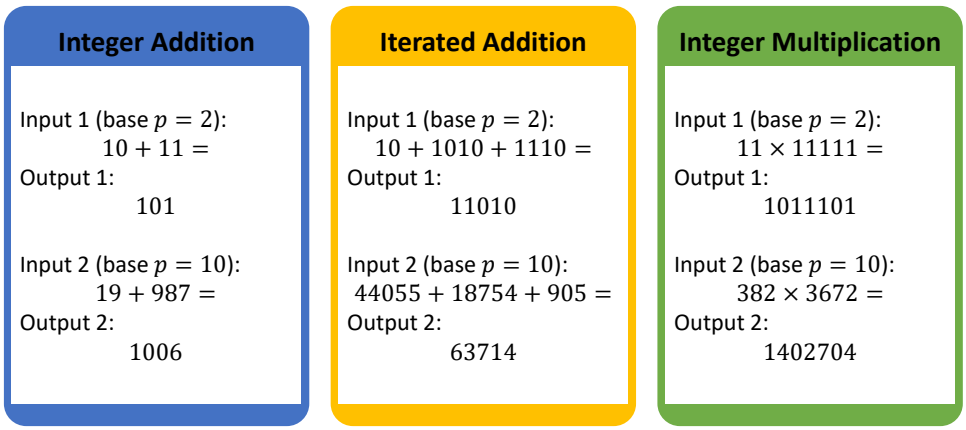

Figure 1: Examples for three elementary arithmetic tasks we consider in this paper: integer addition, iterated addition, and integer multiplication.

where $\boldsymbol{W}_{\mathrm{Q}}^{(l,h)}, \boldsymbol{W}_{\mathrm{K}}^{(l,h)}, \boldsymbol{W}_{\mathrm{V}}^{(l,h)}, \boldsymbol{W}_{\mathrm{O}}^{(l,h)} \in \mathbb{R}^{\lceil \frac{d}{H} \rceil \times d}$ are the query, key, value, and output matrices of the h-th head in the l-th layer. The weight matrices in the feed-forward network are denoted as $\boldsymbol{W}_1^{(l)}, \boldsymbol{W}_2^{(l)} \in \mathbb{R}^{d \times d}$. The activation function σ is chosen to be GeLU (Hendrycks and Gimpel, 2016), following the work of (Radford et al., 2019; Devlin et al., 2019).

The computed embedding $\boldsymbol{x}_n^{(M)}$ is then used to predict the next token s_{n+1}, which is concatenated to the input to continue the sequence generation process. This process terminates when an <EOS> token is generated. Further discussions on related work are listed in Appendix A.

3 Problem Setup

This paper explores the arithmetic reasoning capabilities of LLMs by focusing on three elementary arithmetic tasks: integer addition, iterated addition, and integer multiplication under the autoregressive paradigm. Below, we define the integer representations used throughout the study and provide formal descriptions for each task.

Integer Representations. We assume all integers are non-negative and represented in the base-p form, where $p \geq 2$ is a fixed base. Specifically, an integer with n digits is expressed as $(x_{n-1} \cdots x_0)_p$, where the sequence of digits $\mathbf{x} = [x_{n-1}, \cdots, x_0]$ is the input to the Transformer model. Each digit x_i is treated as an individual token according to the model's tokenization scheme. During generation, the Transformer also produces the outcome digit by digit, respecting the established tokenization structure.

Unlike prior works (Feng et al., 2023; Yang et al., 2024), which represent entire integers as single tokens, we treat each digit of an integer as an individual token, allowing Transformers to process and generate numbers digit by digit. Given that most modern LLMs tokenize large numbers in this manner, our approach not only aligns with practical tokenization strategies but also mitigates the issue of inflated vocabulary size.

Integer Addition. Let $\boldsymbol{a} = (a_{n_1-1} \cdots a_0)_p$ and $\boldsymbol{b} = (b_{n_2-1} \cdots b_0)_p$ denote two integers encoded in base-p. The input sequence is constructed by concatenating the tokenized representations of $\boldsymbol{a}$ and $\boldsymbol{b}$, separated by the addition operator '+'. The task is to generate the sum $\boldsymbol{s} = (s_n \cdots s_0)_p$, which contains $n+1$ digits where $n = \max(n_1, n_2)$ and represents the base-p sum of $\boldsymbol{a}$ and $\boldsymbol{b}$, output digit-by-digit.

Iterated Addition. Consider k integers $\boldsymbol{a}_1 = (a_{1,n_1-1} \cdots a_{1,0})_p, \cdots, \boldsymbol{a}_k = (a_{k,n_k-1} \cdots a_{k,0})_p$ where $n = \max\{n_1, \cdots, n_k\}$. The input sequence consists of the tokenized representations of these integers, concatenated and separated by the addition operator '+'. The Transformer model must output the sum $\boldsymbol{s} = (s_{n+\lceil \log_p k \rceil - 1} \cdots s_0)_p$, which contains $n + \lceil \log_p k \rceil$ digits and represents the base-p sum of the k integers.

Integer Multiplication. In the integer multiplication task, the goal is to compute the product of two integers, truncated to a pre-specified length l. Let $\boldsymbol{a} = (a_{n_1-1} \cdots a_0)_p$ and $\boldsymbol{b} = (b_{n_2-1} \cdots b_0)_p$ be two integers, each no longer than n digits in base-p. The input sequence is constructed by concatenating the tokenized representations of $\boldsymbol{a}$ and $\boldsymbol{b}$, separated by the multiplication operator '$\times$'. The objective is to generate the lowest l digits ($l \leq 2n$) of the product $\boldsymbol{s} = (s_{l-1} \cdots s_0)_p$ in base-p arithmetic.

Remark 3.1. We consider a generalized case of integer multiplication where overflow may occur if the result exceeds the given digit length. Standard integer multiplication is a special case of this framework when $l = 2n$.

Figure 1 illustrates examples of the three tasks. Integer addition is the simplest of these operations and can be viewed as a specific instance of iterated addition. Furthermore, integer multiplication inherently involves the summation of several intermediate products. Consequently, we present these tasks in increasing order of complexity. In the subsequent sections, we use the notations $\mathrm{ADD}(n, p)$ to denote n-digit addition in base-p arithmetic, $\mathrm{IterADD}(n, k, p)$ for the iterated addition of k integers with n digits each in base-p, and $\mathrm{MUL}(n, l, p)$ for the multiplication of two n-digit integers in base-p, truncated to l digits.

4 Low-Precision Transformers Struggle with Basic Arithmetic Tasks

Recent studies (Marchisio et al., 2024; Jin et al., 2024) have shown that LLMs operating under low-precision constraints encounter significant challenges in performing basic mathematical tasks. In this section, we examine the expressive limitations of Transformers under such constraints and seek to explain the sharp decline in their arithmetical capabilities. Specifically, we demonstrate that Transformers restricted to low-precision arithmetic exhibit substantial difficulty in solving even elementary arithmetic problems.

To formalize these limitations, we build on the framework introduced by Li et al. (2024) and utilize the setting of a **constant-precision Transformer** (See formal definition in Appendix B). In this setting, the internal states of the model's neurons are constrained to represent real numbers using only c bits, where c is a small constant independent of the input sequence length. These numbers may be represented by floating point in IEEE 754 formats (Kahan, 1996) or fixed point formats. This configuration mirrors many practical deployment scenarios, in which LLMs often employ reduced-precision formats such as `float8`, `int8`, or even `int4`, particularly during inference (Han et al., 2015). Given that these models typically process input sequences comprising thousands of tokens, it is reasonable and realistic to assume that the numerical precision remains fixed at a small constant, independent of sequence length. Under the constant-precision setting, we examine the expressiveness of the Transformer model in elementary arithmetic problems.

Theorem 4.1. *For any fixed integers p, there exist constant-precision Transformers with constant depth L and hidden dimension $d = O(n^2)$ that can solve the $\mathrm{ADD}(n, p)$ task.*

Theorem 4.1 suggests that the bounded-depth Transformers with reasonable hidden dimensions are capable of solving the integer addition task. However, as we will show in subsequent theorems, constant-precision Transformers exhibit pronounced limitations when considering more complex arithmetic problems.

Theorem 4.2. *For any integers p and L, and for any polynomial f, there exist problem scales n and k such that no constant-precision autoregressive Transformer with L layers and hidden dimension $d < f(n, k)$ can correctly solve the $\mathrm{IterADD}(n, p)$ task.*

Theorem 4.3. *For any integers p and L, and for any polynomial f, there exist problem scales n and l such that no constant-precision autoregressive Transformer with L layers and hidden dimension $d < f(n, l)$ can correctly solve the $\mathrm{MUL}(n, l, p)$ task.*

What accounts for this limitation? As presented in Appendix D, our proof is grounded in circuit complexity theory. By modeling the constant-precision Transformer as a computational circuit, we rigorously analyze its expressive limitations through the lens of circuit complexity (Feng et al., 2023; Merrill et al., 2022; Merrill and Sabharwal, 2023; Li et al., 2024). Specifically, Li et al. (2024) proves that the expressiveness of constant-precision Transformers with polynomial size and bounded depth is upper-bounded by the computation complexity class AC^0. In contrast, we demonstrate that the complexity of tasks such as $\mathrm{IterADD}$ and MUL exceeds that of AC^0, using reductions from `Majority`, a well-established problem that has been provably unsolvable by the circuits in AC^0 (Razborov, 1987; Smolensky, 1987). Consequently, these tasks are inherently hard for low-precision Transformers.

Practical Implications. Intuitively, when performing iterated addition, a Transformer must compute the sum of multiple digits at one time. However, under low-precision constraints, maintaining accurate intermediate results becomes significantly challenging. For example, for the model under

`int4` precision, adding 8 and 9 in base-10 arithmetic will require several neurons to store the results to avoid overflow. In such cases, a Transformer operating in a low-precision regime would either require additional steps to compute the correct sum or need much more neurons to distribute the computation to avoid overflow. This will finally result in the Transformers failing to complete the task with a fixed number of layers and a reasonable model size.

While low-precision Transformers can effectively handle some simplest arithmetic tasks, such as basic integer addition, their capacity is severely limited when addressing more complex tasks. As demonstrated, low numerical precision, such as `int8` and `float8`, imposes fundamental constraints, preventing these models from solving problems that would require Transformers with super-polynomial size.

5 Standard-Precision Transformers Are Sufficient for Arithmetic Tasks

In Section 4, we demonstrated that low-precision Transformers struggle with arithmetic tasks due to their expressive limitations. In this section, we will show that increasing numerical precision is essential to overcoming this limitation. In particular, we focus on **standard-precision** Transformers and show that such models can overcome these limitations and solve arithmetic problems efficiently.

To formalize the notion of standard precision (e.g., `float32`), we follow Feng et al. (2023) and adopt the setting of a **logarithmic-precision Transformer** (See formal definition in Appendix B). In this setting, the Transformer's internal neurons can represent real numbers with up to $O(\log n)$ bits, where n denotes the maximum input sequence length. Given that modern LLMs often limit their context length to hundreds of thousands of tokens (OpenAI, 2023; Touvron et al., 2023; Anthropic, 2024), it is natural to treat 32 as the logarithmic scale corresponding to $100,000$. Hence, the logarithmic-precision setting reflects practical deployment scenarios.

We first establish that, under logarithmic precision, a Transformer with constant depth and dimension can solve both the integer addition and iterated addition tasks for arbitrarily large input lengths, as shown in Theorems 5.1 and 5.2.

Theorem 5.1. *For any integers n and p, there exists a logarithmic-precision Transformer with constant depth L and constant hidden dimension d (independent of n) that can generate the correct output for any input on the $\mathrm{ADD}(n, p)$ task.*

Theorem 5.2. *For any integers n, k, and p, there exists a logarithmic-precision Transformer with constant depth L and constant hidden dimension d (independent of n and k) that can generate the correct output for any input on the $\mathrm{IterADD}(n, k, p)$ task.*

We now turn to integer multiplication. As established in Theorem 5.3, a logarithmic-precision Transformer with constant depth and polynomial hidden dimensions is capable of solving the integer multiplication task.

Theorem 5.3. *For any integers n, l, and p, there exists a logarithmic-precision Transformer with constant depth and hidden dimensions $O(n^2)$ that can generate the correct output for any input on the $\mathrm{MUL}(n, l, p)$ task.*

Theorems 5.1 to 5.3 demonstrate that, under standard precision, a bounded-depth Transformer with reasonable size can solve all elementary arithmetic tasks. Compared to the theoretical results for low-precision Transformers (Theorems 4.1 to 4.3), even a modest increase in numerical precision leads to a substantial improvement in expressiveness for arithmetic tasks.

The Reason for Increased Expressiveness. In Section 4, we highlighted low-precision Transformers' difficulties in performing elementary arithmetic operations. A critical insight is that low-precision Transformers struggle to store intermediate results, even for simple operations like adding two digits. This issue is mitigated by using standard precision. For instance, `float32` can accurately represent integers within a sufficiently large range to store intermediate results during computation. Thus, standard precision allows Transformers to perform arithmetic tasks more effectively.

Practical Implications. Our theoretical results underscore the critical importance of numerical precision when deploying Transformers for arithmetic tasks. Under low-precision settings, a Transformer requires super-polynomial model size to solve even elementary arithmetic problems, which is impractical for real-world applications. While low-precision models may offer computational efficiency, they are likely to fail in scenarios that demand accurate numerical reasoning, such as mathematical problem-solving or scientific computing. However, a slight increase in precision—such as

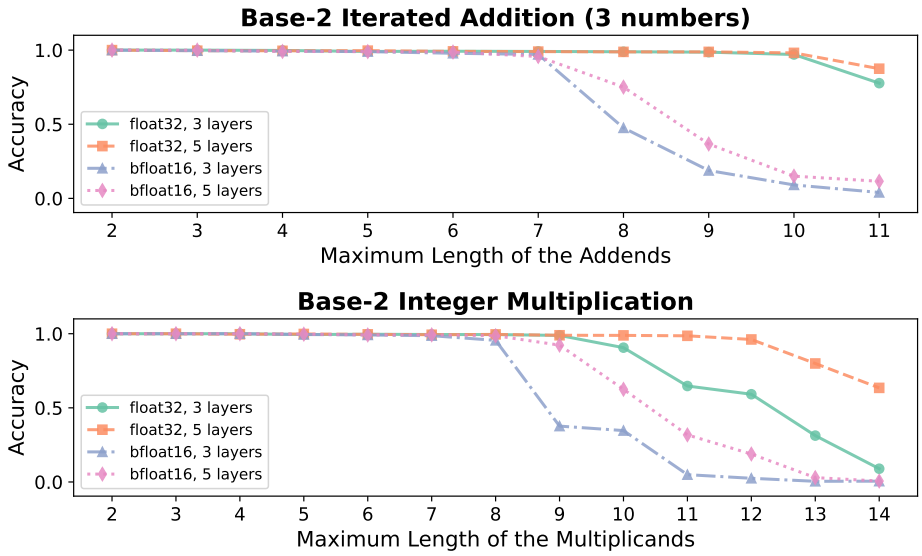

Figure 2: Model performance on different tasks in base-2. Within each sub-figure, the x-axis represents the maximum digits length and the y-axis represents the accuracy gained by each model. The figure indicates that, for all tasks, Transformers utilizing `float32` (32-bit precision) with 3 layers and 5 layers outperform their `bfloat16` (16-bit precision) counterparts.

using `float32`—enables Transformers to handle more complex arithmetic operations while maintaining a reasonable hidden dimension. Thus, employing sufficient numerical precision is crucial for ensuring both accuracy and robustness in arithmetic tasks, and should be a key consideration when designing or deploying LLMs for applications involving complex arithmetic reasoning.

6 Experiments

In the preceding sections, we employed complexity theory to demonstrate that low-precision Transformers face significant challenges in performing elementary arithmetic tasks. To validate these theoretical insights, we conduct a series of experiments to compare the performance of Transformer models under different precisions. The results provide empirical evidence that the model's ability to execute arithmetic operations drops as precision decreases, reinforcing our theoretical results.

6.1 Experimental Setup

Tasks and datasets. We evaluate three elementary arithmetic tasks: integer addition, iterated addition, and integer multiplication, as presented in Figure 1. Each task involves a series of experiments with base $p = 2, 10$ and varying choices of digit length n. For integer addition, we examine the addition of integers in both base-2 and base-10, with digit lengths $n \in \{4, 8, 16, 32, 64\}$. For iterated addition, we examine the addition of three numbers in base-2, with digit lengths $n \in [2, 11]$, as well as in base-10, with digit lengths $n \in [1, 4]$. Similarly, for integer multiplication, we run experiments in base-2 with digit lengths $n \in [2, 14]$, and in base-10 with digit length $n \in [2, 5]$. We dynamically generated datasets for both training and testing, with further details on dataset construction available in Appendix F.

Model configurations. For all experiments, we use Transformer models with hidden dimension $d = 256$, heads $H = 4$, and model depth $L \in \{3, 5\}$. The causal self-attention layer employs rotary positional embeddings (Su et al., 2024), which replace the traditional sinusoidal embeddings. For activation, we choose NewGeLU, the variant of GeLU (Hendrycks and Gimpel, 2016), and apply Xiaver initialization across the parameters (Glorot and Bengio, 2010).

Model training and inference. We adopt the AdamW optimizer (Loshchilov and Hutter, 2019) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, lr $= 10^{-3}$ and weight decay $= 0.01$ in all experiments. We utilize a scheduler with linear warmup and cosine decay over 100k steps, with a warm-up period of 10k steps. We optimize the cross-entropy loss on answer tokens. Each model is trained for 100k steps with a batch size of 512. During inference, models are required to output the exact answers to the

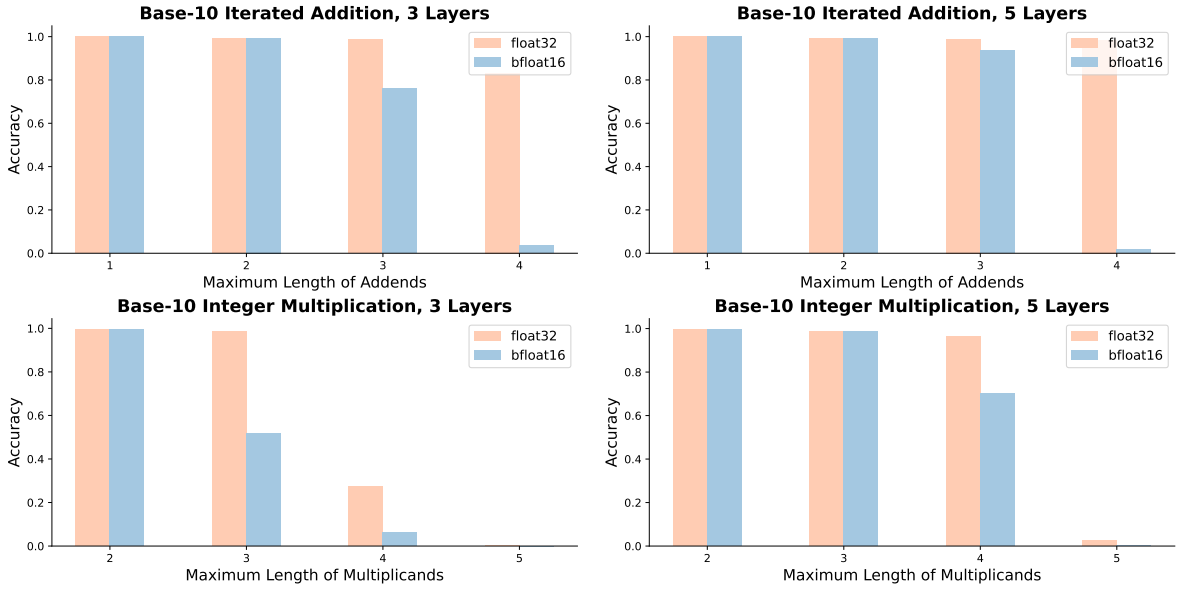

Figure 3: We evaluate model performance on iterated addition tasks involving three numbers and integer multiplication tasks. Each sub-figure presents a comparison of the performance between `float32` (32-bit precision) and `bfloat16` (16-bit precision).

tasks. We report the accuracy as the evaluation metric, and for each task, we test the accuracy on about 50k samples. To assess the impact of model precision, we conduct experiments with both `float32` and `bfloat16` numerical precisions.

6.2 Experimental Results

Integer addition proves to be too easy, maintaining more than 94% accuracy even as digit length increases to 32 in both base-2 and base-10 with both `float32` and `bfloat16` (detailed results in Appendix F). The results of iterated addition and multiplication in base-2 are shown in Figure 2, while the corresponding results in base-10 are shown in Figure 3. Each sub-figure corresponds to a task with the x-axis representing the maximum length of the addends and the multiplicands respectively, and the y-axis representing the test accuracy. In the context of the iterated addition task, the accuracy shows a significantly more pronounced decline in 16-bit precision compared to 32-bit precision as the maximum digits length increases, across all model depths. Specifically, the 16-bit precision exhibits a marked decline in performance for lengths ranging from 7 to 10 in base-2, whereas the 32-bit precision maintains near-complete accuracy across these ranges. In base-10, 32-bit can achieve 90% correctness in contrast to the 16-bit representation, which struggles to yield accurate results. In the context of multiplication tasks, as the maximum digit length increases, 32-bit precision achieves significantly higher accuracy compared to 16-bit precision. Notably, when the length reaches 13 in base-2, the accuracy of 16-bit drops to a very low level, indicating their inability to generate correct results. Additionally, in base-10, we observe a noticeable reduction in accuracy when transitioning from 32-bit to 16-bit precision, particularly for inputs with a maximum length of 3 in 3-layer models and a maximum length of 4 in 5-layer models. This suggests that the differences of precision for performing elementary arithmetic tasks are consistent with our theoretical results.

6.3 Further Experiments on LLMs

We further conducted an extensive set of experiments on LLMs to empirically support our theoretical findings. Specifically, we evaluated the models' performance across three elementary arithmetic tasks, as illustrated in Figure 1. In the integer addition task, we tested the addition of two base-10 integers, varying their digit lengths from 1 to 13. For the iterated addition task, we expanded the setup to include the addition of three and five base-10 numbers, also with digit lengths ranging from 1 to 9 and 1 to 5. Similarly, in the integer multiplication task, we examined the multiplication of two base-10 numbers, with digit lengths spanning from 1 to 5. The process of generating data is the same as in previous experiments.

All experiments utilized the LLaMA 3.1 8B Instruct model (Dubey et al., 2024) as the base

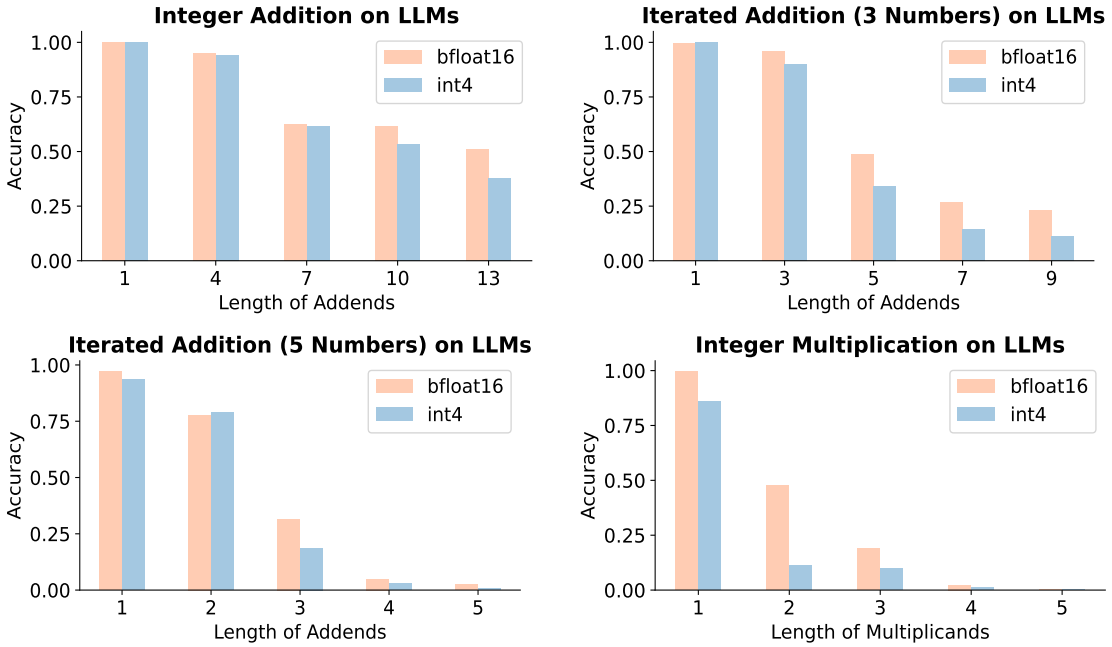

Figure 4: The performance of LLAMA 3.1 8B Instruct on various tasks in base-10. Within each sub-figure, we compare the performance of `bfloat16` (16-bit precision) and `int4` (4-bit precision).

model, employing a few-shot learning approach for inference. Detailed specifications of the prompt construction and generation parameters can be found in the Appendix F. During inference, the LLMs were tasked with producing exact solutions to the given arithmetic problems. For each task, we evaluate the model on 1k samples to compute the accuracy serving as the evaluation metric. Since the LLaMA 3.1 model is trained with `bfloat16` precision, we adopted its performance under `bfloat16` as our baseline. To further investigate the impact of reduced numerical precision, we additionally evaluated the model's performance when quantized to 4-bit precision via the AWQ algorithm (Lin et al., 2024).

The results of the experiments are shown in Figure 4. Each sub-figure presents the results of a task, where the x-axis denotes the maximum length of the addends or multiplicands, and the y-axis represents the test accuracy. In the integer addition task, reducing the numerical precision from 16-bit to 4-bit has no observable effect on accuracy when the number length is less than or equal to 7, and only a slight reduction in accuracy is observed when the number length exceeds 7. This comparable performance between 16-bit and 4-bit precision supports our theorem, which asserts that both low-precision and standard-precision arithmetic can successfully handle integer addition tasks. In contrast, for iterated addition and integer multiplication tasks, the reduction in numerical precision results in a significant decrease in accuracy. Specifically, in the iterated addition task, accuracy drops by nearly 50% as the length of the addends increases. The impact is even more pronounced in the integer multiplication task, where performance deteriorates further. These experimental findings support our theoretical results that numerical precision is a critical factor in the success of iterated addition and integer multiplication tasks. Overall, the results underscore the consistency between the precision requirements for these elementary arithmetic tasks and our theoretical predictions.

7 Conclusion

In this work, we have theoretically analyzed the impact of numerical precision on LLMs for mathematical reasoning. By focusing on three elementary arithmetic tasks, integer addition, iterated addition, and integer multiplication, we constructively demonstrate that the Transformers operating under standard precision can handle these tasks effectively. In contrast, Transformers with low precision struggle with complex arithmetic tasks, excelling only at integer addition. Extensive experimental results corroborate our theoretical findings, showing that standard precision models outperform low precision ones. We believe this study offers valuable insights for the development of more powerful LLMs in mathematical reasoning.

8 Limitations

One limitation of this work is that we have not fully explored all key components of mathematical reasoning. While the arithmetic tasks considered are foundational, there remain other essential elements of mathematical reasoning whose dependence on numerical precision is still unclear. Additionally, our focus was exclusively on numerical precision, but we acknowledge that other factors are likely to play a significant role in applying LLMs to mathematical reasoning. We leave these explorations for future work.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: Student Research Workshop*, pages 225–237, St. Julian's, Malta. Association for Computational Linguistics.

Ekin Akyürek, Dale Schuurmans, Jacob Andreas, Tengyu Ma, and Denny Zhou. 2022. What learning algorithm is in-context learning? investigations with linear models. *arXiv preprint arXiv:2211.15661*.

Silas Alberti, Niclas Dern, Laura Thesing, and Gitta Kutyniok. 2023. Sumformer: Universal approximation for efficient transformers. In *Topological, Algebraic and Geometric Learning Workshops 2023*, pages 72–86. PMLR.

Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2024. Learning from mistakes makes llm better reasoner. *Preprint*, arXiv:2310.20689.

Anthropic. 2024. The claude 3 model family: Opus, sonnet, haiku.

Sanjeev Arora and Boaz Barak. 2009. *Computational complexity: a modern approach*. Cambridge University Press.

Lochan Basyal and Mihir Sanghvi. 2023. Text summarization using large language models: A comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models. *Preprint*, arXiv:2310.10449.

Satwik Bhattamishra, Kabir Ahuja, and Navin Goyal. 2020. On the ability and limitations of transformers to recognize formal languages. *arXiv preprint arXiv:2009.11264*.

David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Advances in neural information processing systems*, volume 33, pages 1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. 2023. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*.

Vincent Cheng and Zhang Yu. 2023. Analyzing ChatGPT's mathematical deficiencies: Insights and contributions. In *Proceedings of the 35th Conference on Computational Linguistics and Speech Processing (ROCLING 2023)*, pages 188–193, Taipei City, Taiwan. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP).

David Chiang, Peter Cholak, and Anand Pillay. 2023. Tighter bounds on the expressivity of transformer encoders. In *Proceedings of the 40th International Conference on Machine Learning*, pages 5544–5562.

Damai Dai, Yutao Sun, Li Dong, Yaru Hao, Shuming Ma, Zhifang Sui, and Furu Wei. 2023. Why can gpt learn in-context? language models implicitly perform gradient descent as meta-optimizers. In *ICLR 2023 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. 2019. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Nouha Dziri, Ximing Lu, Melanie Sclar, Xiang (Lorraine) Li, Liwei Jiang, Bill Yuchen Lin, Sean Welleck, Peter West, Chandra Bhagavatula, Ronan Le Bras, Jena Hwang, Soumya Sanyal, Xiang Ren, Allyson Ettinger, Zaid Harchaoui, and Yejin Choi. 2023. Faith and fate: Limits of transformers on compositionality. In *Advances in Neural Information Processing Systems*, volume 36, pages 70293–70332. Curran Associates, Inc.

Nelson Elhage, Neel Nanda, Catherine Olsson, Tom Henighan, Nicholas Joseph, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, et al. 2021. A mathematical framework for transformer circuits. *Transformer Circuits Thread*, 1.

Guhao Feng, Bohang Zhang, Yuntian Gu, Haotian Ye, Di He, and Liwei Wang. 2023. Towards revealing the mystery behind chain of thought: A theoretical perspective. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. 2024. Mathematical capabilities of chatgpt. *Advances in neural information processing systems*, 36.

Shivam Garg, Dimitris Tsipras, Percy Liang, and Gregory Valiant. 2022. What can transformers learn in-context? a case study of simple function classes. In *Advances in Neural Information Processing Systems*.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Siavash Golkar, Mariel Pettee, Alberto Bietti, Michael Eickenberg, Miles Cranmer, Geraud Krawezik, Francois Lanusse, Michael McCabe, Ruben Ohana, Liam Holden Parker, Bruno Régaldo-Saint Blancard, Tiberiu Tesileanu, Kyunghyun Cho, and Shirley Ho. 2024. xval: A continuous number encoding for large language models.

Sophia Gu. 2023. Llms as potential brainstorming partners for math and science problems. *Preprint*, arXiv:2310.10677.

Michael Hahn. 2020. Theoretical limitations of self-attention in neural sequence models. *Transactions of the Association for Computational Linguistics*, 8:156–171.

Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*.

Yiding Hao, Dana Angluin, and Robert Frank. 2022. Formal language recognition by hard attention transformers: Perspectives from circuit complexity. *Transactions of the Association for Computational Linguistics*, 10:800–810.

Joy He-Yueya, Gabriel Poesia, Rose E. Wang, and Noah D. Goodman. 2023. Solving math word problems by combining language models with symbolic solvers. *Preprint*, arXiv:2304.09102.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

IEEE. 2019. Ieee standard for floating-point arithmetic. *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pages 1–84.

Shima Imani, Liang Du, and Harsh Shrivastava. 2023. MathPrompter: Mathematical reasoning using large language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*, pages 37–42, Toronto, Canada. Association for Computational Linguistics.

Renren Jin, Jiangcun Du, Wuwei Huang, Wei Liu, Jian Luan, Bin Wang, and Deyi Xiong. 2024. A comprehensive evaluation of quantization strategies for large language models. *arXiv preprint arXiv:2402.16775*.

William Kahan. 1996. Ieee standard 754 for binary floating-point arithmetic. *Lecture Notes on the Status of IEEE*, 754(94720-1776):11.

Jikun Kang, Xin Zhe Li, Xi Chen, Amirreza Kazemi, Qianyi Sun, Boxing Chen, Dong Li, Xu He, Quan He, Feng Wen, Jianye Hao, and Jun Yao. 2024. Mindstar: Enhancing math reasoning in pre-trained llms at inference time. *Preprint*, arXiv:2405.16265.

Nayoung Lee, Kartik Sreenivasan, Jason D. Lee, Kangwook Lee, and Dimitris Papailiopoulos. 2024. Teaching arithmetic to small transformers. In *The Twelfth International Conference on Learning Representations*.

Zhiyuan Li, Hong Liu, Denny Zhou, and Tengyu Ma. 2024. Chain of thought empowers transformers to solve inherently serial problems. In *The Twelfth International Conference on Learning Representations*.

Zhenwen Liang, Dian Yu, Xiaoman Pan, Wenlin Yao, Qingkai Zeng, Xiangliang Zhang, and Dong Yu. 2024. MinT: Boosting generalization in mathematical reasoning via multi-view fine-tuning. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 11307–11318, Torino, Italia. ELRA and ICCL.

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. 2024. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6:87–100.

Bingbin Liu, Jordan T. Ash, Surbhi Goel, Akshay Krishnamurthy, and Cyril Zhang. 2023. Transformers learn shortcuts to automata. In *The Eleventh International Conference on Learning Representations*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. 2024. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *The Twelfth International Conference on Learning Representations*.

Shengjie Luo, Shanda Li, Shuxin Zheng, Tie-Yan Liu, Liwei Wang, and Di He. 2022. Your transformer may not be as powerful as you expect. In *Advances in Neural Information Processing Systems*.

Yujun Mao, Yoon Kim, and Yilun Zhou. 2024. Champ: A competition-level dataset for fine-grained analyses of llms' mathematical reasoning capabilities. *Preprint*, arXiv:2401.06961.

Kelly Marchisio, Saurabh Dash, Hongyu Chen, Dennis Aumiller, Ahmet Üstün, Sara Hooker, and Sebastian Ruder. 2024. How does quantization affect multilingual llms? *arXiv preprint arXiv:2407.03211*.

Sean McLeish, Arpit Bansal, Alex Stein, Neel Jain, John Kirchenbauer, Brian R. Bartoldson, Bhavya Kailkhura, Abhinav Bhatele, Jonas Geiping, Avi Schwarzschild, and Tom Goldstein. 2024. Transformers can do arithmetic with the right embeddings. *Preprint*, arXiv:2405.17399.

William Merrill and Ashish Sabharwal. 2023. The parallelism tradeoff: Limitations of log-precision transformers. *Transactions of the Association for Computational Linguistics*.

William Merrill, Ashish Sabharwal, and Noah A Smith. 2022. Saturated transformers are constant-depth threshold circuits. *Transactions of the Association for Computational Linguistics*, 10:843–856.

Swaroop Mishra, Matthew Finlayson, Pan Lu, Leonard Tang, Sean Welleck, Chitta Baral, Tanmay Rajpurohit, Oyvind Tafjord, Ashish Sabharwal, Peter Clark, and Ashwin Kalyan. 2022. LILA: A unified benchmark for mathematical reasoning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5807–5832, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. 2021. Investigating the limitations of transformers with simple arithmetic tasks. *Preprint*, arXiv:2102.13019.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. 2022. In-context learning and induction heads. *Transformer Circuits Thread*. Https://transformer-circuits.pub/2022/in-context-learning-and-induction-heads/index.html.

OpenAI. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Jorge Pérez, Pablo Barceló, and Javier Marinkovic. 2021. Attention is turing complete. *The Journal of Machine Learning Research*, 22(1):3463–3497.

Jorge Pérez, Javier Marinković, and Pablo Barceló. 2019. On the turing completeness of modern neural network architectures. *arXiv preprint arXiv:1901.03429*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Syed Rifat Raiyan, Md Nafis Faiyaz, Shah Md. Jawad Kabir, Mohsinul Kabir, Hasan Mahmud, and Md Kamrul Hasan. 2023. Math word problem solving by generating linguistic variants of problem statements. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pages 362–378, Toronto, Canada. Association for Computational Linguistics.

Alexander A Razborov. 1987. Lower bounds for the size of circuits of bounded depth with basis f^; g. *Math. notes of the Academy of Sciences of the USSR*, 41(4):333–338.

Ankit Satpute, Noah Gießing, André Greiner-Petter, Moritz Schubotz, Olaf Teschke, Akiko Aizawa, and Bela Gipp. 2024. Can llms master math? investigating large language models on math stack exchange. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 2316–2320, New York, NY, USA. Association for Computing Machinery.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019. Analysing mathematical reasoning abilities of neural models. In *International Conference on Learning Representations*.

Paulo Shakarian, Abhinav Koyyalamudi, Noel Ngu, and Lakshmivihari Mareedu. 2023. An independent evaluation of chatgpt on mathematical word problems (mwp). *Preprint*, arXiv:2302.13814.

Yunfan Shao, Linyang Li, Junqi Dai, and Xipeng Qiu. 2023. Character-LLM: A trainable agent for role-playing. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13153–13187, Singapore. Association for Computational Linguistics.

Ruoqi Shen, Sebastien Bubeck, Ronen Eldan, Yin Tat Lee, Yuanzhi Li, and Yi Zhang. 2024. Positional description matters for transformers arithmetic.

Roman Smolensky. 1987. Algebraic methods in the theory of lower bounds for boolean circuit complexity. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 77–82.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Pragya Srivastava, Manuj Malik, Vivek Gupta, Tanuja Ganu, and Dan Roth. 2024. Evaluating LLMs' mathematical reasoning in financial document question answering. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 3853–3878, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. 2024. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

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*.

Johannes Von Oswald, Eyvind Niklasson, Ettore Randazzo, João Sacramento, Alexander Mordvintsev, Andrey Zhmoginov, and Max Vladymyrov. 2023. Transformers learn in-context by gradient descent. In *International Conference on Machine Learning*, pages 35151–35174. PMLR.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations*.

Colin Wei, Yining Chen, and Tengyu Ma. 2022a. Statistically meaningful approximation: a case study on approximating turing machines with transformers. *Advances in Neural Information Processing Systems*, 35:12071–12083.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. 2022b. Chain of thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2021. Thinking like transformers. In *International Conference on Machine Learning*, pages 11080–11090. PMLR.

Kaiyue Wen, Xingyu Dang, and Kaifeng Lyu. 2024. Rnns are not transformers (yet): The key bottleneck on in-context retrieval. *Preprint*, arXiv:2402.18510.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024a. An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Yiran Wu, Feiran Jia, Shaokun Zhang, Hangyu Li, Erkang Zhu, Yue Wang, Yin Tat Lee, Richard Peng, Qingyun Wu, and Chi Wang. 2024b. Mathchat: Converse to tackle challenging math problems with llm agents. *Preprint*, arXiv:2306.01337.

Ryutaro Yamauchi, Sho Sonoda, Akiyoshi Sannai, and Wataru Kumagai. 2023. Lpml: Llm-prompting markup language for mathematical reasoning. *Preprint*, arXiv:2309.13078.

Kai Yang, Jan Ackermann, Zhenyu He, Guhao Feng, Bohang Zhang, Yunzhen Feng, Qiwei Ye, Di He, and Liwei Wang. 2024. Do efficient transformers really save computation? In *Forty-first International Conference on Machine Learning*.

Shunyu Yao, Binghui Peng, Christos Papadimitriou, and Karthik Narasimhan. 2021. Self-attention networks can process bounded hierarchical languages. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3770–3785.

Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhu Chen. 2024. MAmmoTH: Building math generalist models through hybrid instruction tuning. In *The Twelfth International Conference on Learning Representations*.

Chulhee Yun, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank J Reddi, and Sanjiv Kumar. 2019. Are transformers universal approximators of sequence-to-sequence functions? *arXiv preprint arXiv:1912.10077*.

Chulhee Yun, Yin-Wen Chang, Srinadh Bhojanapalli, Ankit Singh Rawat, Sashank Reddi, and Sanjiv Kumar. 2020. O (n) connections are expressive enough: Universal approximability of sparse transformers. In *Advances in Neural Information Processing Systems*, volume 33, pages 13783–13794.

Aojun Zhou, Ke Wang, Zimu Lu, Weikang Shi, Sichun Luo, Zipeng Qin, Shaoqing Lu, Anya Jia, Linqi Song, Mingjie Zhan, and Hongsheng Li. 2024a. Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. In *The Twelfth International Conference on Learning Representations*.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Joshua M. Susskind, Samy Bengio, and Preetum Nakkiran. 2024b. What algorithms can transformers learn? a study in length generalization. In *The Twelfth International Conference on Learning Representations*.

Yongchao Zhou, Uri Alon, Xinyun Chen, Xuezhi Wang, Rishabh Agarwal, and Denny Zhou. 2024c. Transformers can achieve length generalization but not robustly. In *ICLR 2024 Workshop on Mathematical and Empirical Understanding of Foundation Models*.

Wenhao Zhu, Hongyi Liu, Qingxiu Dong, Jingjing Xu, Shujian Huang, Lingpeng Kong, Jiajun Chen, and Lei Li. 2024. Multilingual machine translation with large language models: Empirical results and analysis. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 2765–2781, Mexico City, Mexico. Association for Computational Linguistics.

A Related Work

A.1 LLMs for Mathematical Reasoning

Mathmetical Reasoning. Recent studies highlight the limitations of current LLMs in mathematical reasoning (Ahn et al., 2024; Srivastava et al., 2024). Satpute et al. (2024) demonstrated that advanced models like GPT-4 can generate relevant answers, but these answers are not always accurate. Additionally, Mao et al. (2024) found that current LLMs struggle even with verifying the solutions to mathematical problems. To enhance the mathematical capabilities of LLMs, several studies have carefully designed prompting strategies (Shakarian et al., 2023; Cheng and Yu, 2023; Gu, 2023; Lu et al., 2024) or finetuned LLMs on mathematics-related datasets (An et al., 2024; Liang et al., 2024; Raiyan et al., 2023; Mishra et al., 2022; Yue et al., 2024). Other approaches include inference-based searching methods (Kang et al., 2024), the application of external tools (Yamauchi et al., 2023; He-Yueya et al., 2023; Chen et al., 2023), and the introduction of simulated interaction processes (Wu et al., 2024b) or self-verification mechanisms (Wang et al., 2023; Zhou et al., 2024a).

Arithmetical Reasoning. Bubeck et al. (2023) highlighted arithmetical reasoning as a key component of true mathematical ability. However, Saxton et al. (2019); Dziri et al. (2023) identified significant challenges that LLMs encounter when solving elementary arithmetic tasks, such as multi-digit addition and multiplication. A common approach to mitigate these difficulties is to reverse the output digit order (Shen et al., 2024), or both the input and output digit order simultaneously (Lee et al., 2024). Other studies have focused on developing improved positional encodings (Golkar et al., 2024; McLeish et al., 2024) or positional tokens (Nogueira et al., 2021) that are more suitable for arithmetic tasks. Zhou et al. (2024b,c) further examined the length extrapolation capabilities of LLMs in solving basic arithmetic problems, emphasizing the importance of data formats and positional embeddings for better generalization.

A.2 Computational Powers of Transformers

Another more relevant line of work investigates the theoretical expressive power of Transformers from a computational perspective.

Universal Approximation. Early theoretical work on Transformers primarily focused on their function approximation capabilities. Yun et al. (2019) demonstrated that Transformers can universally approximate any continuous sequence-to-sequence functions, given sufficient size. This universality result has since been extended to various Transformer variants, such as Sparse Transformers (Yun et al., 2020), Linear Transformers (Alberti et al., 2023), and Transformers with relative positional encodings (RPE) (Luo et al., 2022). Additionally, previous studies established that infinite-precision Transformers are Turing-complete (Pérez et al., 2019, 2021), while Wei et al. (2022a) showed that finite-precision Transformers are approximately Turing-complete. Although these results highlight Transformers' computational capacity, our work develops expressiveness results under more practical settings, exploring the differences in expressiveness across varying levels of numerical precision.

Formal Language Learning. Another line of research focuses on the ability of Transformers to learn formal languages. Liu et al. (2023) explored how Transformers simulate finite state automata, while Bhattamishra et al. (2020); Yao et al. (2021) studied their ability to recognize counter languages and Dyck languages, respectively. On the negative side, Hahn (2020) showed that Transformers are not capable of learning distributions over languages. In addition to affirmative results, several works have characterized the limitations of Transformers from the perspective of formal language modeling (Hahn, 2020; Bhattamishra et al., 2020; Weiss et al., 2021; Yao et al., 2021; Chiang et al., 2023) or circuit simulation (Hao et al., 2022; Merrill et al., 2022; Merrill and Sabharwal, 2023). However, few of these studies focus on the autoregressive Transformers commonly used in LLMs, which we investigate in this paper.

Chain-of-Thought and In-Context Learning. Chain-of-Thought (CoT) prompting (Wei et al., 2022b) plays a crucial role in tasks requiring complex reasoning structures, and several studies aim to understand its underlying mechanisms. For instance, Feng et al. (2023); Li et al. (2024) analyzed CoT from an

expressiveness perspective, and Yang et al. (2024); Wen et al. (2024) examined CoT across more different model variants. In-context learning (Brown et al., 2020; Garg et al., 2022) is another powerful aspect of LLMs. Some theoretical work has shown that in-context learning can be explained through gradient descent (Akyürek et al., 2022; Dai et al., 2023; Von Oswald et al., 2023), while others attribute it to the induction heads mechanism (Elhage et al., 2021; Olsson et al., 2022).

B Additional Background and Preliminary

B.1 Circuit Complexity

Circuit complexity classes capture various aspects of computational complexity, typically bounding circuit width and depth. For a more detailed introduction, we refer to Arora and Barak (2009).

We begin by defining Boolean circuits. A Boolean circuit over a basis of gates is represented as a finite-size directed acyclic graph (DAG), where each vertex corresponds to either a basis function (or gate) or an input bit. Some internal nodes are designated as outputs, and the *fan-in* of a vertex is defined as its in-degree. Building on this definition, we can define the complexity classes NC^i, AC^i, and TC^i:

- NC^i: This class consists of constant fan-in, polynomial-sized circuits made up of AND, OR, and NOT gates, with a depth of $O(\log^i n)$.
- AC^i: This class includes unbounded fan-in, polynomial-sized circuits composed of AND, OR, and NOT gates (with NOT gates allowed only on inputs), also having a depth of $O(\log^i n)$.
- TC^i: This class extends AC^i by allowing majority gates.

The relationships among the NC, AC, and TC hierarchies are as follows:

$$\mathsf{NC}^i \subset \mathsf{AC}^i \subset \mathsf{TC}^i \subset \mathsf{NC}^{i+1},\ \mathsf{NC}^0 \subsetneq \mathsf{AC}^0 \subsetneq \mathsf{TC}^0.$$

B.2 Constant-precision Transformer

Previous work has investigated the expressiveness of constant-precision Transformers (Li et al., 2024), utilizing a simplified version of the IEEE 754 standards (IEEE, 2019). Our constant-precision setting is analogous, and we will introduce the floating-point representations we consider here.

A $(e + 2s + 1)$-floating point representation includes e exponent bits, $2s$ precision bits, and one sign bit. The numbers representable under this representation are defined as follows:

$$\mathbb{F}_{e,s} := \{S \cdot 2^{-s+E} \mid -2^{-2s} + 1 \leq S \leq 2^{2s} - 1, -2^{e-1} \leq E \leq \max(2^{e-1} - 1, 0), S, E \in \mathbb{Z}\}.$$

For any $x \in \mathbb{R}$, its representation under this floating-point format is determined by rounding to the nearest value in $\mathbb{F}$. In the event of a tie, we select the number with the smaller absolute value. In this paper, we focus on the case where $e = 0$, which means all representable numbers take the form $S \cdot 2^{-s}$, with $S \in \mathbb{Z}$ such that $-2^{-2s} + 1 \leq S \leq 2^{2s} - 1$. However, this is necessary only for Theorem 4.1, while Theorems 4.2 and 4.3 do not depend on specific numerical representations.

Li et al. (2024) demonstrated that constant-depth Transformers with constant precision belong to the complexity class AC^0.

B.3 Logarithmic-precision Transformer

A key limitation of constant-precision representation is that it fails to capture the input size n within a single neuron. To address this, we consider logarithmic precision, allowing for $O(\log n)$ bits for numerical representations. Logarithmic-precision Transformers possess several advantageous properties (Feng et al., 2023):

- For floating-point representations with $O(\log n)$ bits, any real number $x \in O(\mathrm{poly}(n))$ can be represented with $O(\mathrm{poly}(1/n))$ error.

- Each neuron in the Transformer can only store $O(\log n)$ bits of information, which means it cannot retain all input data. Consequently, computation must be distributed across the network, aligning with the operational principles of Transformers.

Previous work (Merrill et al., 2022; Merrill and Sabharwal, 2023) has shown that logarithmic-precision Transformers fall within the complexity class TC^0.

C Technical Lemmas

C.1 Technical Lemmas for Logarithmic Precision MLP

We will first provide some basic results for logarithmic precision MLP, which comes from (Feng et al., 2023). We omit the proofs here. The complete proof can be found in the appendix of (Feng et al., 2023).

Lemma C.1 (Feng et al., 2023, Lemma C.1)**.** *For any $\epsilon > 0$, there exists a two-layer MLP $f : \mathbb{R}^2 \to \mathbb{R}$ with 4 hidden dimension and* GeLU *activation, such that for any $a, b \in [-M, M]$, $|f(a, b) - ab| \leq \epsilon$. Moreover, the ℓ_∞ norm of f is bounded by $O(\mathrm{poly}(M, 1/\epsilon))$.*

Lemma C.2 (Feng et al., 2023, Lemma C.2)**.** *For any two-layer MLP $\boldsymbol{g} : \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}$ with* ReLU *activation, and the l_∞ norm bounded by M. Then, for any $\epsilon > 0$, there exists a two-layer MLP $\boldsymbol{f}$ with the same size and* GeLU *activation, such that for all $\boldsymbol{x} \in \mathbb{R}^{d_1}$, we have $\|\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{g}(\boldsymbol{x})\|_\infty \leq \epsilon$. Moreover, the ℓ_∞ norm of $\boldsymbol{f}$ is bounded by $O(\mathrm{poly}(M, 1/\epsilon))$.*

Lemma C.3 (Feng et al., 2023, Lemma C.4)**.** *Consider selection function $\boldsymbol{g} : \mathbb{R}^d \times \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^d$: $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t) = \boldsymbol{x}$ if $t > 0$, and $\boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t) = \boldsymbol{y}$ otherwise. Then, for any $\epsilon > 0$, $\alpha > 0$, and $M > 0$, there exist a two-layer MLP with $2d + 2$ hidden dimension and* GeLU *activation, such that for all $\boldsymbol{x} \in [-M, M]^d$, $\boldsymbol{y} \in [-M, M]^d$, and $t \in (-\infty, -\alpha] \cup [\alpha, +\infty)$, $\|\boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}, t) - \boldsymbol{g}(\boldsymbol{x}, \boldsymbol{y}, t)\|_\infty \leq \epsilon$. Moreover, the ℓ_∞ norm is bounded by $O(\mathrm{poly}(M, 1/\alpha, 1/\epsilon))$.*

C.2 Technical Lemmas for Logarithmic Precision Attention Layer

Feng et al. (2023) also studied the expressive power of the standard attention layer and introduced two basic operations: COPY and MEAN, showing that the standard attention layer with log-precision is capable of these operations under some regularity assumptions. Here, we will provide the results and further consider a special operation called SINGLE COPY.

Consider a sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$ where $\boldsymbol{x}_i = (\tilde{\boldsymbol{x}}_i, r_i, 1) \in [-M, M]^{d+2}$ and M is a constant. Denote attention matrices $\boldsymbol{K}, \boldsymbol{Q}, \boldsymbol{V} \in \mathbb{R}^{d' \times (d+2)}$ and $\boldsymbol{q}_i = \boldsymbol{Q}\boldsymbol{x}_i$, $\boldsymbol{k}_j = \boldsymbol{K}\boldsymbol{x}_j$, $\boldsymbol{v}_j = \boldsymbol{V}\boldsymbol{x}_j$. For any scalars $0 < \rho, \delta < M$, define the matching set $\mathcal{S}_i = \{j \leq i : |\boldsymbol{q}_i \cdot \boldsymbol{k}_j| \leq \rho\}$ and consider the following operations:

- COPY: The output is $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n$ with $\boldsymbol{u}_i = \boldsymbol{v}_{\mathrm{pos}(i)}$, where $\mathrm{pos}(i) = \mathrm{argmax}_{j \in \mathcal{S}_i} r_j$. The output $\boldsymbol{u}_i$ is undefined when $|\mathcal{S}_i| = 0$.
- MEAN: The output is $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n$ with $\boldsymbol{u}_i = \mathrm{mean}_{j \in \mathcal{S}_i} \boldsymbol{v}_j$. The output $\boldsymbol{u}_i$ is undefined when $|\mathcal{S}_i| = 0$.
- SINGLE COPY: The output is $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n$ with $\boldsymbol{u}_i = \boldsymbol{v}_{\mathrm{pos}(i)}$, where $\mathrm{pos}(i)$ is the unique element in $\mathcal{S}_i$. The output is undefined if $|\mathcal{S}_i| \neq 1$.

We next make the following regularity assumption:

Assumption C.4. For any input sequences $\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_n$, the matrices $\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}$ and scalars ρ, δ satisfy:

- For any $i, j \in [n]$, $|\boldsymbol{q}_i \cdot \boldsymbol{k}_j| \leq \rho$ or $\boldsymbol{q}_i \cdot \boldsymbol{k}_j \leq -\delta$ hold.
- For any $i, j \in [n]$, $i = j$ or $|r_i - r_j| \geq \delta$ hold.
- $\|\boldsymbol{V}\|_\infty \leq 1$.

Now, we will show a logarithmic precision attention layer with $O(d)$ embedding dimension and one attention head can perform the operations defined above.

Lemma C.5 (Feng et al., 2023, Lemma C.7)**.** *Suppose Assumption C.4 holds and $\rho \leq \frac{\delta^2}{8M}$. For any $\epsilon > 0$, there exists an attention layer with a single attention head and $O(d)$ embedding dimension that can approximate* COPY *operation, and the ℓ_∞ norm of the parameters is bounded by $O(\text{poly}(M, 1/\delta, \log(n), \log(1/\epsilon)))$. Formally, for any input sequences $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, denote the attention output as $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_n$. Then for any $i \in [n]$ with $\mathcal{S}_i \neq \varnothing$, we have $\|\boldsymbol{o}_i - \boldsymbol{u}_i\|_\infty \leq \epsilon$.*

Lemma C.6 (Feng et al., 2023, Lemma C.8)**.** *Suppose Assumption C.4 holds and $\rho \leq \frac{\delta\epsilon}{16M\ln(\frac{4Mn}{\epsilon})}$. For any $0 < \epsilon \leq M$, there exists an attention layer with a single attention head and $O(d)$ embedding dimension that can approximate* MEAN *operation, and the ℓ_∞ norm of the parameters is bounded by $O(\text{poly}(M, 1/\delta, \log(n), \log(1/\epsilon)))$. Formally, for any input sequences $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, denote the attention output as $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_n$. Then for any $i \in [n]$ with $\mathcal{S}_i \neq \varnothing$, we have $\|\boldsymbol{o}_i - \boldsymbol{u}_i\|_\infty \leq \epsilon$.*

We omit the proofs of Lemmas C.5 and C.6 here. The complete proof can be found in the appendix of (Feng et al., 2023).

Lemma C.7. *Assume Assumption C.4 holds and $\delta - \rho \geq c\rho$ where $c > 0$. For any $\epsilon > 0$, there exists an attention layer with a single attention head and $O(d)$ embedding dimension that can approximate* SINGLE COPY *operation, and the ℓ_∞ norm of the parameters is bounded by $O(\text{poly}(M, 1/\delta, 1/c, \log(n), \log(1/\epsilon)))$. Formally, for any input sequences $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$, denote the attention output as $\boldsymbol{o}_1, \boldsymbol{o}_2, \ldots, \boldsymbol{o}_n$. Then, for any $i \in [n]$ with $|\mathcal{S}_i| = 1$, we have $\|\boldsymbol{o}_i - \boldsymbol{u}_i\|_\infty \leq \epsilon$.*

Proof. We construct the query, key and value as follows:

- Query: $\lambda \boldsymbol{q}_i \in \mathbb{R}^d$
- Key: $\boldsymbol{k}_i \in \mathbb{R}^d$
- Value: $\boldsymbol{v}_i \in \mathbb{R}^d$

where $\lambda > 0$ is a constant. Denote a_{ij} as the attention score, we have

$$a_{i,j} = \frac{\exp(\lambda(\boldsymbol{q}_i \cdot \boldsymbol{k}_j))}{\sum_{j'} \exp(\lambda(\boldsymbol{q}_i \cdot \boldsymbol{k}_{j'}))} = \frac{\exp(\lambda(\boldsymbol{q}_i \cdot \boldsymbol{k}_j))}{\sum_{j'} \exp(\lambda(\boldsymbol{q}_i \cdot \boldsymbol{k}_{j'}))}.$$

Since $\delta - \rho \geq c\rho$, we have $\delta - \rho \geq \frac{c}{c+1}\delta$. By setting $\lambda = \frac{(c+1)\ln(\frac{2nM}{\epsilon})}{c\delta}$ (which is bounded by $O(\text{poly}(M, 1/\delta, 1/c, \log(n), \log(1/\epsilon)))$), we have

$$\begin{aligned}
a_{i,\text{pos}(i)} &\geq \frac{\exp(-\lambda\rho)}{\exp(-\lambda\rho) + (n-1)\exp(-\lambda\delta)} && (1)\\
&= \frac{1}{1 + (n-1)\exp(-\lambda(\delta - \rho))}\\
&\geq 1 - (n-1)\exp(-\lambda(\delta - \rho)) && (2)\\
&\geq 1 - n\exp\left(-\ln\left(\frac{2nM}{\epsilon}\right)\right)\\
&= 1 - \frac{\epsilon}{2M}.
\end{aligned}$$

Here, we use Assumption C.4 and $|\mathcal{S}_i| = 1$ in (1), which implies that for $j' \neq \text{pos}(i)$, we have $\boldsymbol{q}_i \cdot \boldsymbol{k}_{j'} \leq -\delta$; and in (2) we apply the inequality $\frac{1}{1+x} \geq 1 - x$ for all $x \geq 0$. Thus we have

$$\begin{aligned}
\|\boldsymbol{o}_i - \boldsymbol{u}_i\|_\infty = \left\|\sum_j a_{ij}\boldsymbol{v}_j - \boldsymbol{v}_{\text{pos}(i)}\right\|_\infty &\leq M\|\boldsymbol{V}\|_\infty \cdot \left(1 - a_{i,\text{pos}(i)} + \sum_{j \neq \text{pos}(i)} a_{i,j}\right)\\
&= M\|\boldsymbol{V}\|_\infty(2 - 2a_{i,\text{pos}(i)}) \leq \epsilon,
\end{aligned}$$

which concludes the proof. □

C.3 Technical Lemmas for Constant Precision Calculation

In this section, we provide some technical lemmas for constant precision calculation and constant precision MLPs. Suppose we have $2s$-bit precision bits and no exponent bits, and denote $B_s = 2^s - 2^{-s}$. The maximum number we could represent is B_s, while the minimum is $-B_s$.

Lemma C.8 (Li et al., 2024, Lemma E.1 and E.2)**.** *For any* $s \in \mathbb{N}_+$, $\exp(-B_s) = 0$ *and* $\exp(B_s) = B_s$.

Proof. Notice that $\exp(B_s) \geq \mathrm{e}B_s > 2^{s+1}$, thus $\exp(-B_s) \leq 2^{-s-1}$. This implies that $\exp(-B_s) = 0$. On the other hand, $\exp(B_s) \geq B_s + 1 > B_s$, which implies that $\exp(B_s) = B_s$. □

Lemma C.9. *For any* $s \in \mathbb{N}_+$, $\mathrm{GeLU}(-B_s) = 0$.

Proof. It suffices to prove $B_s\Phi(-B_s) \leq 2^{-s-1}$ where Φ is the CDF of Gaussian distribution.

For $s = 1$, we have $B_s = \frac{3}{2}$, thus

$$B_s\Phi(-B_s) \leq \frac{3}{2}\Phi(-1) \leq \frac{3}{2} \cdot \frac{1 - 0.68}{2} < \frac{1}{4}.$$

For $s \geq 2$, we have

$$\begin{aligned} B_s\Phi(-B_s) &= \frac{B_s}{\sqrt{2\pi}} \int_{B_s}^{+\infty} \mathrm{e}^{-\frac{x^2}{2}} \mathrm{d}x \leq \frac{B_s}{\sqrt{2\pi}} \int_{B_s}^{+\infty} \mathrm{e}^{-\frac{B_s x}{2}} \mathrm{d}x \\ &\leq \sqrt{\frac{2}{\pi}} \mathrm{e}^{-\frac{B_s^2}{2}} \leq \frac{2\sqrt{2}}{\sqrt{\pi}(B_s^2 + 2)} \leq \frac{2\sqrt{2}}{\sqrt{\pi}2^{2s}} \leq \frac{1}{2^{s+1}}, \end{aligned}$$

completing our proof. □

D Proofs for Section 4

Theorem 4.1. *For any fixed integers* p*, there exist constant-precision Transformers with constant depth* L *and hidden dimension* $d = O(n^2)$ *that can solve the* $\mathrm{ADD}(n, p)$ *task.*

For ease of reading, we first describe an algorithm to perform $\mathrm{ADD}(n, p)$ (Algorithm 1) and prove the correctness, then construct a Transformer with the given configurations in Theorem 4.1 to simulate Algorithm 1.

Algorithm 1: $\mathrm{ADD}(n, p)$ Algorithm

Input : Two p-adic numbers $\boldsymbol{a}, \boldsymbol{b}$ with length n_1, n_2
Output : The sum of the inputs $\boldsymbol{o}$, containing $(n + 1)$ bits where $n = \max(n_1, n_2)$

```
a_n = 0, b_n = 0;
foreach i ∈ {0, ··· , n − 1} do
    Compute the carry-on bits c:
    i_∧ = max{j ≤ i | a_i + b_i ≥ p};
    i_∨ = max{j ≤ i | a_i + b_i ≤ p − 2};
    c_i = 1_{i_∧ > i_∨};
end
Compute the outcome o: o_i = (a_i + b_i + c_{i−1}) mod p;
```

Lemma D.1 (An algorithm to perform $\mathrm{ADD}(n, p)$)**.** *Algorithm 1 outputs* $\boldsymbol{o} = \boldsymbol{a} + \boldsymbol{b}$ *for all inputs* $\boldsymbol{a}, \boldsymbol{b}$.

Proof. Given two n-bit p-adic numbers $\boldsymbol{a}$ and $\boldsymbol{b}$, we can compute the carry-over bits $\boldsymbol{c} = (c_n, \cdots, c_1)$ as follows:

$$\begin{aligned} &c_{-1} = 0, \\ &c_0 = \mathbf{1}_{a_0 + b_0 \geq p}, \\ &c_1 = (c_0 \cdot \mathbf{1}_{a_1 + b_1 \geq p - 1}) \vee \mathbf{1}_{a_1 + b_1 \geq p}, \\ &\cdots, \\ &c_i = (c_{i-1} \cdot \mathbf{1}_{a_i + b_i \geq p - 1}) \vee \mathbf{1}_{a_i + b_i \geq p}. \end{aligned} \tag{3}$$

By removing the recursive computation, we can compute the carry-over by the following formulas:

$$\begin{aligned} i_\wedge &= \max\{j \le i \mid a_i + b_i \ge p\}, \\ i_\vee &= \max\{j \le i \mid a_i + b_i \le p-2\}, \\ c_i &= \mathbf{1}_{i_\wedge > i_\vee}. \end{aligned} \tag{4}$$

Or equivalently,

$$c_i = \bigvee_{0 \le j \le i} \left[\mathbf{1}_{a_j + b_j \ge p} \wedge \bigwedge_{j \le k \le i} \mathbf{1}_{a_k + b_k \ge p-2} \right]. \tag{5}$$

In Equation (4), $i_\wedge$ indicates the closest bit that can contribute an additional carry to the higher bits up to the i th bit, while $i_\vee$ identifies the highest bit lower than i, such that the carry generated by the bits below $i_\vee$ does not affect the bits higher than $i_\vee$. Therefore, the carry-over $c_i = 1$ if and only if $i_\wedge < i_\vee$.

Then after computing the carryover bits, we can compute the sum of the input integers as follows:

$$\begin{aligned} o_0 &= (a_0 + b_0) \bmod p, \\ o_1 &= (a_1 + b_1 + c_0) \bmod p, \\ &\cdots \\ o_i &= (a_i + b_i + c_{i-1}) \bmod p, \\ o_n &= c_{n-1}. \end{aligned} \tag{6}$$

Therefore, the output $\boldsymbol{o}$ is exactly the sum of the integers and the Algorithm 1 outputs $\mathrm{ADD}(\boldsymbol{a}, \boldsymbol{b})$ for all $\boldsymbol{a}, \boldsymbol{b} \in \{0,1\}^n$. □

Next, we provide the proof for Theorem 4.1.

Proof for Theorem 4.1. Now, we demonstrate that a constant-precision transformer, with a constant depth L, a fixed number of attention heads, and $O(n^2)$ hidden dimensions, is capable of simulating Algorithm 1. Consequently, this model can accurately generate correct output for any input integers $\boldsymbol{a}, \boldsymbol{b}$.

Initial Embeddings: The total length of the input sequence is no longer than $2(n+1)$. We categorize the tokens into two classes: number tokens $(0, 1, \cdots, p-1)$ and auxiliary tokens ($+$, $=$, <SOS> and <EOS>).

- **Embedding of input token a_i:** $\boldsymbol{u}^0_{a,i} = (a_i \boldsymbol{e}_{i+1}, \mathbf{0}, -1, \mathbf{0}, 0, 1, 1)$.
- **Embedding of input token b_i:** $\boldsymbol{u}^0_{b,i} = (\mathbf{0}, b_i \boldsymbol{e}_{i+1}, -1, \mathbf{0}, 0, 2, 1)$.
- **Embedding of output token o_i:** $\boldsymbol{u}^0_{o,i} = (\mathbf{0}, \mathbf{0}, o_i, \boldsymbol{e}_{i+1}, 0, 3, -1)$.
- **Embedding of "+" token:** $\boldsymbol{u}^0_{+} = (\mathbf{0}, \mathbf{0}, -1, \mathbf{0}, 0, 4, -1)$.
- **Embedding of "=" token:** $\boldsymbol{u}^0_{=} = (\mathbf{0}, \mathbf{0}, -1, \mathbf{0}, 1, 5, -1)$.
- **Embedding of <SOS> token:** $\boldsymbol{u}_{\text{<SOS>}} = (\mathbf{0}, \mathbf{0}, -1, \mathbf{0}, 0, 6, -1)$.
- **Embedding of <EOS> token:** $\boldsymbol{u}_{\text{<EOS>}} = (\mathbf{0}, \mathbf{0}, 0, \mathbf{0}, 0, 3, -1)$.

where $\boldsymbol{e}_i \in \mathbb{R}^{n+1}$ is one-hot vector. Suppose we have $2s$-bit precision and define $B_s = 2^s - 2^{-s}$. Recall that, the maximum number we can represent is B_s, while the minimum is $-B_s$.

Block 1. The first block of the Transformer will COPY the value of a_i, b_i to position of outputs. We use the attention head to finish the COPY operation. We set the query, key and value as follows:

- Query: $\boldsymbol{q} = B_s$.
- Key: $\boldsymbol{k} = \boldsymbol{u}^0[3n+6]$, i.e., $\boldsymbol{k} = 1$ for input number tokens, and $\boldsymbol{k} = -1$ otherwise.

- Value: $\boldsymbol{v} = \boldsymbol{u}^0[1, \cdots, 2n+2]$, i.e., $\boldsymbol{v} = (a_i \boldsymbol{e}_{i+1}, \boldsymbol{0})$ for input $\boldsymbol{a}$, $\boldsymbol{v} = (\boldsymbol{0}, b_i \boldsymbol{e}_{i+1})$ for input $\boldsymbol{b}$, and $\boldsymbol{v} = \boldsymbol{0}$ otherwise.

Since we have only constant precision here, we need to check the attention values carefully. The attention value (without normalization) will be B_s for tokens a_i, b_i, and $-B_s$ otherwise. By Lemma C.8, we can get $\exp(B_s) = B_s, \exp(-B_s) = 0$, and the normalization term is $2nB_s = B_s$, indicating the attention weights will be 1 for token x_i, y_i, and 0 otherwise. Thus the attention output at the position of output tokens is always $(a_0, \cdots, a_n, b_0, \cdots, b_n)$.

Block 2. The second block of the Transformer uses MLPs to calculate the output $\boldsymbol{o}$ using Algorithm 1. We will finish these calculations following the steps below:

- Calculate $r_i = a_i + b_i$ for $i = 0, \cdots, n$. This can be implemented by a MLP with constant hidden dimension. Here, we require $B_s \geq 2p$ to avoid overflow of r_i.
- Calculate $f_i = \mathbf{1}_{r_i \geq p}$ and $g_i = \mathbf{1}_{r_i \geq p-2}$. By Lemma C.9, we have

$$f_i = \frac{\text{GeLU}[B_s \cdot (2r_i - 2p + 1)]}{\text{GeLU}(B_s)}, \; g_i = \frac{\text{GeLU}[B_s \cdot (2r_i - 2p + 5)]}{\text{GeLU}(B_s)}.$$

  Here, we require $B_s \geq 4p$ to avoid overflow of $2r_i - 2p + 1$.
- Calculate c_i by the following formula:

$$c_i = \bigvee_{0 \leq j \leq i} \left[\mathbf{1}_{a_j + b_j \geq p} \wedge \bigwedge_{j \leq k \leq i} \mathbf{1}_{a_k + b_k \geq p-2} \right] = \bigvee_{0 \leq j \leq i} \left[f_j \wedge \bigwedge_{j \leq k \leq i} g_k \right].$$

  Notice that

$$\bigvee_{1 \leq i \leq \gamma} \alpha_i = \frac{\text{GeLU}\left[B_s \left(\sum_{i=1}^{\gamma} \alpha_i\right)\right]}{\text{GeLU}(B_s)}, \quad \bigwedge_{1 \leq i \leq \gamma} \alpha_i = 1 - \bigvee_{1 \leq i \leq \gamma} (1 - \alpha_i),$$

  which implies that $\vee, \wedge$ can be implemented by constant-depth, constant-precision MLP with constant hidden dimension. Therefore, we can calculate the value of c_i using $O(n)$ hidden dimension.
- Calculate $o_i = a_i + b_i + c_{i-1}$ for $i = 0, \cdots, n$. This can be implemented by a MLP with constant hidden dimension. Again, we require $B_s \geq 2p$ to avoid overflow of o_i.

Since we need to calculate r_i, f_i, g_i, c_i for any i, this requires the hidden dimension of $O(n^2)$.

Block 3. Finally, this block filters out the token o_i from $\boldsymbol{o}$. Let's consider the token o_{i+1}, where $i \in \{0, \cdots, n-1\}$, we want to predict the next token o_i. First, we calculate $\boldsymbol{e}_{i+1}$ by using $\boldsymbol{e}_{i+2}$ from the positional embedding of $\boldsymbol{m}^0_{o,i+1}$. Then, we will calculate o_i by

$$o_i = \langle \boldsymbol{e}_{i+1}, \boldsymbol{o} \rangle.$$

Since $x = \text{GeLU}(x) - \text{GeLU}(-x)$, we have

$$o_i = \sum_{j=1}^{n+1} \boldsymbol{e}_{i+1}[j] \boldsymbol{o}[j] = \sum_{j=1}^{n+1} [\text{GeLU}(\boldsymbol{e}_{i+1}[j] - B_s(2 - 2\boldsymbol{o}[j])) - \text{GeLU}(-\boldsymbol{e}_{i+1}[j] - B_s(2 - 2\boldsymbol{o}[j]))],$$

which implies we can calculate the value of o_i using $O(n)$ hidden dimension. The final output from this layer is characterized by the equation:

$$\boldsymbol{e}^3_{o,i+1} = \begin{cases} (o_i, \boldsymbol{e}_{i+1}) & \text{if } i > 0, \\ (0, \boldsymbol{0}) & \text{if } i = 0. \end{cases}$$

Predict Next Token. Given the output embeddings of the last Transformer layer $\boldsymbol{e}^3_{o,i}$, and the word embeddings, the Transformer can simply predict the next token by finding the nearest word embeddings.

In this construction, we only require $B_s \geq 4p$, which implies constant precision is sufficient. □

Theorem 4.2. *For any integers p and L, and for any polynomial f, there exist problem scales n and k such that no constant-precision autoregressive Transformer with L layers and hidden dimension $d < f(n,k)$ can correctly solve the* $\mathrm{IterADD}(n,p)$ *task.*

Proof. Suppose there exists integers $p \geq 2, L$ and polynomial f, such that for any problem scales n, k, there exists a constant-precision autoregressive Transformer with L layers and hidden dimension $d \leq f(n,k)$ that can correctly solve the $\mathrm{IterADD}(k,n,p)$ task.

Consider $\mathrm{Maj}(b_1, \cdots, b_k)$, where $b_i \in \{0,1\}$. Here we provide a reduction from $\mathrm{Maj}(b_1, \cdots, b_k)$ to $\mathrm{IterADD}(2, k', p)$ where $k' = p^{\lceil \log_p k \rceil} \leq pk$. Let $a_i = b_i(p^2-1)$ for $i = 1, \cdots, k$, and

$$a_{k+1} + \cdots + a_{k'} = p^{\lceil \log_p k \rceil + 1} - (p^2-1)\left\lceil \frac{k}{2} \right\rceil.$$

This is feasible since

$$p^{\lceil \log_p k \rceil + 1} - (p^2-1)\left\lceil \frac{k}{2} \right\rceil \leq (p^{\lceil \log_p k \rceil} - k)(p^2-1)$$

for $p \geq 2$. Thus we can get

$$\mathrm{Maj}(b_1, \cdots, b_k) = 1 \iff \sum_{i=1}^{k} b_i \geq \left\lceil \frac{k}{2} \right\rceil \iff \sum_{i=1}^{k'} a_i \geq p^{\lceil \log_p k \rceil + 1} \iff o_{\lceil \log_p k \rceil + 1} > 0$$

Since a bounded-depth, fixed-precision Transformer with polynomial-size generating a single token falls within the complexity class AC^0, by the reduction above we can get $\mathrm{Maj} \in \mathsf{AC}^0$, which is a contradiction. □

Theorem 4.3. *For any integers p and L, and for any polynomial f, there exist problem scales n and l such that no constant-precision autoregressive Transformer with L layers and hidden dimension $d < f(n,l)$ can correctly solve the* $\mathrm{MUL}(n,l,p)$ *task.*

Proof. Suppose there exists integers $p \geq 2, L$ and polynomial f, such that for any problem scales n, l, there exists a constant-precision autoregressive Transformer with L layers and hidden dimension $d \leq f(n,l)$ that can correctly solve the $\mathrm{MUL}(n,l,p)$ task.

Consider $\mathrm{Maj}(c_1, \cdots, c_k)$, where $c_i \in \{0,1\}$. Here we provide a reduction from $\mathrm{Maj}(c_1, \cdots, c_k)$ to $\mathrm{MUL}(n,l,p)$ where

$$n = \left(\lceil \log_p k \rceil + 1\right)\left(p^{\lceil \log_p k \rceil} + \left\lfloor \frac{k}{2} \right\rfloor\right) = O(k \log k),\ l = n + \lceil \log_p k \rceil = O(k \log k).$$

Let

$$k' = p^{\lceil \log_p k \rceil} + \left\lfloor \frac{k}{2} \right\rfloor,\ c_{k+1} = \cdots = c_{k'} = 1,$$

and

$$\boldsymbol{a} = c_1 \underbrace{0 \cdots 0}_{\lceil \log_p n \rceil} c_2 \underbrace{0 \cdots 0}_{\lceil \log_p n \rceil} \cdots c_{k'} \underbrace{0 \cdots 0}_{\lceil \log_p n \rceil},\ \boldsymbol{b} = 1 \underbrace{0 \cdots 0}_{\lceil \log_p n \rceil} 1 \underbrace{0 \cdots 0}_{\lceil \log_p n \rceil} \cdots 1 \underbrace{0 \cdots 0}_{\lceil \log_p n \rceil}.$$

Under this construction, we can get

$$\mathrm{Maj}(c_1, \cdots, c_k) = 1 \iff c_1 + \cdots + c_k \geq \left\lceil \frac{k}{2} \right\rceil \iff c_1 + \cdots + c_{k'} \geq p^{\lceil \log_p k \rceil} \iff o_{l-1} > 0$$

Since a bounded-depth, fixed-precision Transformer with polynomial-size generating a single token falls within the complexity class AC^0, by the reduction above we can get $\mathrm{Maj} \in \mathsf{AC}^0$, which is a contradiction. □

E Proofs for Section 5

Theorem 5.1. *For any integers n and p, there exists a logarithmic-precision Transformer with constant depth L and constant hidden dimension d (independent of n) that can generate the correct output for any input on the* $\mathrm{ADD}(n, p)$ *task.*

Proof. Theorem 5.1 is a special case of Theorem 5.2. Taking $k = 2$ in Theorem 5.2 completes our proof. Notice that $m = \lceil \log_p k \rceil = 1$, which implies that we don't need to combine neighboring bits actually. □

Theorem 5.2. *For any integers n, k, and p, there exists a logarithmic-precision Transformer with constant depth L and constant hidden dimension d (independent of n and k) that can generate the correct output for any input on the* $\mathrm{IterADD}(n, k, p)$ *task.*

For ease of reading, we also describe an algorithm to perform $\mathrm{IterADD}(n, k, p)$ (Algorithm 2) and prove the correctness first. Then, we will construct a constant-size Transformer with logarithmic precision to simulate Algorithm 2.

Algorithm 2: $\mathrm{IterADD}(n, k, p)$ Algorithm

Input : k p-adic numbers $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k$, with maximum length n
Output : The sum of the inputs $\boldsymbol{o}$

$m = \lceil \log_p k \rceil$;
Compute the sum of each bit as $r_j = \sum_{i \in [k]} a_{ij}$ for $j = 0, \cdots, n-1$;
Combine neighboring m bits:

$$s_i = \sum_{j=0}^{m-1} r_{ik+j} p^j$$

for $i = 0, \cdots, \lfloor n/m \rfloor$;
Decompose s_i by $s_i = b_i p^m + q_i$, where $q_i \in [0, p^m - 1]$ and $b_i, q_i \in \mathbb{N}$;
$c_0 = 0$;
foreach $i = 0, \cdots, \lfloor n/m \rfloor$ **do**
  Compute the carry-on bits $\boldsymbol{c}$:
  $i_\wedge = \max\{j \le i \mid q_j + b_{j-1} \ge p^m\}$;
  $i_\vee = \max\{j \le i \mid q_j + b_{j-1} \le p^m - 2\}$;
  $c_i = \mathbf{1}_{i_\wedge > i_\vee}$;
end
Compute the p^m-adic outcome $\tilde{\boldsymbol{o}}$: $\tilde{o}_i = (q_i + b_{i-1} + c_{i-1}) \bmod p^m$ for $i = 0, \cdots, \lfloor n/m \rfloor + 1$;
Covert p^m-adic $\tilde{\boldsymbol{o}}$ to p-adic $\boldsymbol{o}$:

$$o_i = \left\lfloor \frac{\tilde{o}_j \bmod p^{(l+1)}}{p^l} \right\rfloor$$

for $i = jk + l$ where $l \in \{0, \cdots, k-1\}$, $j \in \mathbb{Z}$;

Lemma E.1 (An algorithm to perform $\mathrm{IterADD}(n, k, p)$)**.** *Algorithm 2 outputs $\boldsymbol{o} = \boldsymbol{a}_1 + \cdots + \boldsymbol{a}_k$ for all inputs $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_k$.*

Proof. In the initial four steps, we convert p-adic addition into p^m-adic addition. This transformation enables the expression $\sum_i s_i p^{im}$ to accurately represent the sum of k numbers.

At this stage, $s_i \in [0, kp^m)$. To compute the final results accurately, it is crucial to account for carry-over effects to ensure the final outputs $\tilde{o}_i$ range from 0 to $p^m - 1$. We can decompose each s_i using the formula $s_i = b_i p^m + q_i$, where $q_i \in [0, p^m - 1]$ and $b_i < k \le p^m$. Consequently, the overflow b_i will

only *directly* overflow to the immediately subsequent bit q_{i+1}. And we have $q_{i+1} + b_i \leq 2(p^m - 1)$. Therefore, let $\boldsymbol{c}$ denote the vector to record the carry-over effects on the position i, and we have

$$\begin{aligned} &c_{-1} = 0, \\ &c_0 = \mathbf{1}_{q_0+b_{-1}\geq p^m} \ (b_{-1} := 0), \\ &c_1 = (c_0 \cdot \mathbf{1}_{q_1+b_0\geq p^m-1}) \vee \mathbf{1}_{q_1+b_0\geq p^m}, \\ &\cdots \\ &c_i = (c_{i-1} \cdot \mathbf{1}_{q_i+b_{i-1}\geq p^m-1}) \vee \mathbf{1}_{q_i+b_{i-1}\geq p^m}. \end{aligned} \tag{7}$$

By removing the recursive computation, we can compute the carry-over by the following formulas:

$$\begin{aligned} &i_\wedge = \max\{j \leq i \mid q_j + b_{j-1} \geq p^m\}, \\ &i_\vee = \max\{j \leq i \mid q_j + b_{j-1} \leq p^m - 2\}, \\ &c_i = \mathbf{1}_{i_\wedge > i_\vee}. \end{aligned} \tag{8}$$

Or equivalently,

$$c_i = \bigvee_{0\leq j\leq i} \left[\mathbf{1}_{q_j+b_{j-1}\geq p^m} \wedge \bigwedge_{j\leq k\leq i} \mathbf{1}_{q_k+b_{k-1}\geq p^m-2} \right]. \tag{9}$$

In Equation (8), $i_\wedge$ indicates the nearest bit that can contribute an additional carry to the higher bits up to the i-th bit, while $i_\vee$ identifies the highest bit lower than i, such that the carry generated by bits below $i_\vee$ does not affect bits higher than $i_\vee$. Therefore, the carry-over $c_i = 1$ if and only if $i_\wedge > i_\vee$.

Then after computing the carry-over vectors, we can compute the p^m-adic sum of the input integers as follows:

$$\begin{aligned} &\tilde{o}_0 = q_0, \\ &\tilde{o}_1 = (q_1 + b_0 + c_0) \bmod p^m, \\ &\cdots \\ &\tilde{o}_i = (q_i + b_{i-1} + c_{i-1}) \bmod p^m. \end{aligned} \tag{10}$$

Finally, to convert the p^m-adic number to the final p-adic number $\tilde{\boldsymbol{o}}$, we only need to perform the modulus operation:

$$o_i = \left\lfloor \frac{\tilde{o}_j \bmod p^{(l+1)}}{p^l} \right\rfloor$$

for $i = jk + l$ where $l \in \{0, \cdots, k-1\}$, $j \in \mathbb{Z}$. Therefore, the output $\boldsymbol{o}$ is exactly the sum of the m n-bits integers and the Algorithm 2 outputs $\mathrm{IterADD}(\boldsymbol{a}_1, \cdots, \boldsymbol{a}_m)$ for all $\boldsymbol{a}_1, \cdots, \boldsymbol{a}_m$. □

Next, we provide the proof for Theorem 5.2.

Proof for Theorem 5.2. Now, we demonstrate that a log-precision transformer, with a constant depth, a fixed number of attention heads, and constant embedding dimensions, is capable of simulating Algorithm 2. Consequently, this model can accurately generate correct output for any input integers $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_k$.

Initial Embeddings: The total length of the input sequence is no longer than $k(n+1)$. We categorize the tokens into two classes: number tokens $(0, 1, \cdots, p-1)$ and auxiliary tokens ($+$, $=$, <SOS> and <EOS>). Given the parameters k, n, we determine the parameter $m = \lceil \log_p k \rceil$, as specified in Algorithm 2. The embeddings for these classes are defined as follows:

- **Embedding of input token $a_{i,j}$:** $\boldsymbol{e}^0_{i,j} = \left(a_{i,j}, 0, 0, i, j, j \bmod m, \lfloor \frac{j}{m} \rfloor, p^{j \bmod m}, p^{-(j \bmod m)}, \mathrm{ape}_{i,j}\right)$.
- **Embedding of the i-th "+" token:** $\boldsymbol{e}^0_{i,+} = (0, 1, 0, i, -1, -1, -1, 0, 0, \mathrm{ape}_{i,+})$.
- **Embedding of the "=" token:** $\boldsymbol{e}^0_{=} = (0, 1, 0, k+1, -1, -1, -1, 0, 0, \mathrm{ape}_{=})$.
- **Embedding of the <SOS> token:** $\boldsymbol{e}^0_{\texttt{<SOS>}} = (0, 1, 0, 0, -1, -1, -1, 0, 0, \mathrm{ape}_{\texttt{<SOS>}})$.

- **Embedding of the <EOS> token:** $\boldsymbol{e}^0_{\texttt{<EOS>}} = (0, 0, 1, 0, -1, -1, -1, 0, 0, \text{ape}_{\texttt{<EOS>}})$.
- **Embedding of output token o_i:** $\boldsymbol{e}^0_{o,i} = (o_i, 0, 0, 0, i, i \bmod m, \lfloor \frac{i}{m} \rfloor, p^{i \bmod m}, p^{-(i \bmod m)}, \text{ape}_{o,i})$.

where $\text{ape}_{\dots}$ is the absolute positional encoding. In this construction, the first three dimensions of each initial embedding represent the word embedding, while the last six dimensions account for the position embedding.

Block 1. The first block of the Transformer calculates the following values:

1. $l_{i,j}$: the number of previous tokens (inclusive) $a_{i',j'}$ where $i' = i$ and $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$. $l_{i,j}$ is only defined on input number tokens $a_{i,j}$. If the value is undefined, we will set $l = -1$.
2. $f_{i,j}$: $f_{i,j} = 1$ if no previous tokens (exclusive) $a_{i',j'}$ with $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$. $f_{i,j}$ is only defined on input number tokens $a_{i,j}$. If the value is undefined, we will set $f = -1$.

To calculate the first value, we can set the query, key, value and r in Appendix C.2 as follows:

- Query: $\boldsymbol{q}_{i,j} = \left(-1, 2i, -i^2, -1, 2\lfloor \frac{j}{m} \rfloor, -\lfloor \frac{j}{m} \rfloor^2\right)$.
- Key: $\boldsymbol{k}_{i',j'} = \left(i'^2, i', 1, \lfloor \frac{j'}{m} \rfloor^2, \lfloor \frac{j'}{m} \rfloor, 1\right)$.
- Value: $\boldsymbol{v}_{i',j'} = (\text{ape}_{i',j'})$.
- r: $r_{i',j'} = -\text{ape}_{i',j'}$.

We can calculate the values required in query or key by previous MLPs using Lemma C.1. Thus $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\left(\lfloor \frac{j'}{m} \rfloor - \lfloor \frac{j}{m} \rfloor\right)^2 - (i' - i)^2$, which implies that $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = 0$ if $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$ and $i = i'$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -1$ otherwise. By Lemma C.5, we can use one attention head to copy the absolute position j'' of the first token satisfying the conditions. Finally, the number of the tokens is $j - j'' + 1$.

To calculate the second value, we can set the query, key, and r in Appendix C.2 as follows:

- Query: $\boldsymbol{q}_{i,j} = \left(-1, 2\lfloor \frac{j}{m} \rfloor, -\lfloor \frac{j}{m} \rfloor^2\right)$.
- Key: $\boldsymbol{k}_{i',j'} = \left(\lfloor \frac{j'}{m} \rfloor^2, \lfloor \frac{j'}{m} \rfloor, 1\right)$.
- Value: $\boldsymbol{v}_{i',j'} = (\text{ape}_{i',j'})$.
- r: $r_{i',j'} = -\text{ape}_{i',j'}$.

Thus $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\left(\lfloor \frac{j'}{m} \rfloor - \lfloor \frac{j}{m} \rfloor\right)^2$, which implies that $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = 0$ if $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -1$ otherwise. By Lemma C.5, we can use one attention head to copy the absolute position j'' of the first token satisfying the conditions. Finally, we have $f_{i,j} = 1$ if $j'' = j$. It suffices to determine whether $j'' = j$. Notice that $\mathbf{1}_{j''=j} = \text{ReLU}[1 - (j - j'')^2]$, thus we can calculate the value of $f_{i,j}$ by a constant-width MLP using Lemma C.1.

Finally, if the values are undefined, those values will be set as -1 in the MLP stage by using conditional selection (Lemma C.3) using information in positional embeddings. To sum up, the new embeddings generated in this block can be summarized as: $\boldsymbol{e}^1 = (l, f)$. All these embeddings will be concatenated with the original embeddings.

Block 2. The second block of the Transformer is tailored to execute the first three lines of Algorithm 2. In this block, the model aggregates the adjacent m bits to derive s_i via the attention mechanism. For each token $a_{i,j}$, denote $t_{i,j}$ as the number of previous tokens (inclusive) $a_{i',j'}$ where $\lfloor \frac{j}{m} \rfloor = \lfloor \frac{j'}{m} \rfloor$. The second block of the Transformer calculates the following values:

1. $\frac{1}{t_{i,j}}$ where $t_{i,j}$ is defined above. If the value is undefined, we will set the value as -1.

2. $c_{i,j}$: the mean value applied across $a_{i',j'}p^{j' \bmod m}$ for previous tokens (inclusive) $a_{i',j'}$ where $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$. If the value is undefined, we will set the value as -1.

Then we can get $s_w = c_{i,mw}t_{i,mw}$ where i is the maximize value such that the length of $\boldsymbol{a}_i$ is greater than mk.

To calculate the first value, we first set the query, key and value as follows:

- Query: $\boldsymbol{q}_{i,j} = \left(-1, 2\lfloor \frac{j}{m} \rfloor, -\lfloor \frac{j}{m} \rfloor^2\right)$.
- Key: $\boldsymbol{k}_{i',j'} = \left(\lfloor \frac{j'}{m} \rfloor^2, \lfloor \frac{j'}{m} \rfloor, 1\right)$.
- Value: $\boldsymbol{v}_{i',j'} = (f_{i',j'})$.

These values can be calculated using Lemma C.1. Thus $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\left(\lfloor \frac{j'}{m} \rfloor - \lfloor \frac{j}{m} \rfloor\right)^2$, which implies that $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = 0$ if $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -1$ otherwise. By the definition of $f_{i,j}$ and Lemma C.6, we can get the attention output is $\frac{1}{t_{i,j}}$, as desired.

To calculate the second value, we can set the query, key and value as follows:

- Query: $\boldsymbol{q}_{i,j} = \left(-1, 2\lfloor \frac{j}{m} \rfloor, -\lfloor \frac{j}{m} \rfloor^2\right)$.
- Key: $\boldsymbol{k}_{i',j'} = \left(\lfloor \frac{j'}{m} \rfloor^2, \lfloor \frac{j'}{m} \rfloor, 1\right)$.
- Value: $\boldsymbol{v}_{i',j'} = (a_{i',j'}p^{j' \bmod m})$.

Similar to the calculation of the first value, these values can be calculated using Lemma C.1. Thus $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\left(\lfloor \frac{j'}{m} \rfloor - \lfloor \frac{j}{m} \rfloor\right)^2$, which implies that $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = 0$ if $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -1$ otherwise. By Lemma C.6, the attention output is $c_{i,j}$, as desired.

Finally, if the values are undefined, those values will be set as -1 in the MLP stage by using conditional selection (Lemma C.3) using information in positional embeddings. To sum up, the new embeddings generated in this block can be summarized as: $\boldsymbol{e}^2 = (\frac{1}{t}, c)$. All these embeddings will be concatenated with the original embeddings.

Block 3. The third block of the Transformer calculates the value of $c_{i,j}t_{i,j}$. We will first calculate $t_{i,j}$ by attention layer and $\frac{1}{t_{i,j}}$ from the last block, then calculate $c_{i,j}t_{i,j}$ using Lemma C.1.

Notice that $t_{i,j}$ will not exceed the absolute positional of the current token. We set the query, key and value as follows:

- Query: $\boldsymbol{e}_{i,j} = \left(\frac{1}{t_{i,j}^2}, -\frac{2}{t_{i,j}}, 1\right)$.
- Key: $\boldsymbol{k}_{i',j'} = \left(\text{ape}_{i',j'}^2, \text{ape}_{i',j'}, 1\right)$.
- Value: $\boldsymbol{v}_{i',j'} = (\text{ape}_{i',j'})$.

These values can be calculated using Lemma C.1. Thus $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\left(\frac{\text{ape}_{i',j'}}{t_{i,j}} - 1\right)^2$, which implies that $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = 0$ if $\text{ape}_{i',j'} = t_{i,j}$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -\frac{1}{n^2k^2}$ otherwise since $t_{i,j} \leq nk$. By Lemma C.6, the attention output is $t_{i,j}$, as desired.

Finally, we can calculate the value of $c_{i,j}t_{i,j}$ using the subsequent MLP. To sum up, the new embeddings generated in this block can be summarized as: $\boldsymbol{e}^3 = (ct)$. All these embeddings will be concatenated with the original embeddings.

Block 4. This block of the Transformer executes the fourth line of Algorithm 2, i.e., decompose $c_{i,j}t_{i,j}$ as $b_{i,j}p^m + q_{i,j}$. Notice that $b_{i,j} \leq i$, thus $b_{i,j}$ will not exceed the absolute positional of the current token. We can set the query, key and value as follows:

- Query: $\boldsymbol{q}_{i,j} = \big(-(c_{i,j}t_{i,j} + \frac{1}{2})^2, 2p^m(c_{i,j}t_{i,j} + \frac{1}{2}), -p^{2m}\big)$.
- Key: $\boldsymbol{k}_{i',j'} = \big(1, \text{ape}_{i',j'} - \frac{1}{2}, (\text{ape}_{i',j'} - \frac{1}{2})^2\big)$.
- Value: $\boldsymbol{v}_{i',j'} = \text{ape}_{i',j'}$.

These values can be calculated using Lemma C.1. Thus

$$\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\left[c_{i,j}t_{i,j} - \left(\text{ape}_{i',j'} - \frac{1}{2}\right)p^m + \frac{1}{2}\right]^2,$$

which implies that $|\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle| \leq \left(\frac{p^m-1}{2}\right)^2$ if $\text{ape}_{i',j'} = \lfloor \frac{c_{i,j}t_{i,j}}{p^m} \rfloor$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -\left(\frac{p^m+1}{2}\right)^2$ otherwise. Using Lemma C.7,

$$c = \frac{(p^m+1)^2 - (p^m-1)^2}{(p^m-1)^2} \geq \frac{4}{p^m},$$

thus $1/c = O(p^m) = O(k)$. This implies we can set the query, key and value, such that the attention output is $\lfloor \frac{c_{i,j}t_{i,j}}{p^m} \rfloor = b_{i,j}$. Finally, we calculate $q_{i,j}$ by $q_{i,j} = c_{i,j}t_{i,j} - p^m b_{i,j}$ using the subsequent MLP. The new embeddings generated in this block is $\boldsymbol{e}^4 = (b, q)$.

Block 5. This block of the Transformer computes $q_{w+1} + b_w$ for s_w. Recall that, $s_w = c_{i,mw}t_{i,mw}$ where i is the maximize value such that the length of $\boldsymbol{a}_i$ is greater than mw. We want to calculate those values at the corresponding positions.

First, we use attention layer to COPY q_{w+1} for token $\boldsymbol{a}_i$ defined above. Notice that we can always get the correct value since the position with correct value of s_{w+1} is always in front of that with correct value of s_k. To finish this, we use the attention mechanism to COPY from the position of the value s_{w+1}. This can be implemented by setting query, key, value and r in Appendix C.2:

- Query: $\boldsymbol{q}_{i,j} = \Big(-1, 2\lfloor \frac{j}{m} \rfloor, -\lfloor \frac{j}{m} \rfloor^2, -1\Big)$.
- Key: $\boldsymbol{k}_{i',j'} = \Big((\lfloor \frac{j'}{m} \rfloor - 1)^2, \lfloor \frac{j'}{m} \rfloor - 1, 1, (j' \bmod m)^2\Big)$.
- Value: $\boldsymbol{v}_{i',j'} = (q_{i',j'}, \lfloor \frac{j'}{m} \rfloor)$.
- r: $r_{i',j'} = \text{ape}_{i',j'}$.

We can calculate the values required in query or key by previous MLPs using Lemma C.1. Thus $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = -\Big(\lfloor \frac{j'}{m} \rfloor - \lfloor \frac{j}{m} \rfloor - 1\Big)^2 - (j' \bmod m)^2$, which implies that $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle = 0$ if $\lfloor \frac{j'}{m} \rfloor = \lfloor \frac{j}{m} \rfloor + 1$ and $j' \bmod m = 0$, and $\langle \boldsymbol{q}_{i,j}, \boldsymbol{k}_{i',j'} \rangle \leq -1$ otherwise.

By Lemma C.5, we can use one attention head to copy the value of $q_{i',j'}$ and $\lfloor \frac{j'}{m} \rfloor$ from the last token satisfying the conditions. This indicates the first dimension of attention output is q_{w+1} if there exists an input number with length greater than $m(w+1)$, as desired. Otherwise, q_{w+1} should be zero, while the attention output is undefined above. We can distinguish between these two cases by checking the second dimension of attention output. This is because if no input numbers has the length greater than $m(k+1)$, then the second dimension of the attention output will be at most $\lfloor \frac{j}{m} \rfloor$. We can use Lemma C.3 to distinguish two cases and set $q_{w+1} = 0$ if necessary. Finally, we can use the subsequent MLP to calculate the correct value of $q_{w+1} + b_w$ for s_w. Furthermore, we also use the MLP to calculate $\mathbf{1}_{q_{w+1}+b_w \geq p^m}$, $\mathbf{1}_{q_{w+1}+b_w \leq p^m-2}$, $\mathbf{1}_{b_w \geq p^m}$, $\mathbf{1}_{b_w \leq p^m-2}$ by

$$\mathbf{1}_{q_{w+1}+b_w \geq p^m} = \text{ReLU}[q_{w+1} + b_w - (p^m - 1)] - \text{ReLU}[q_{w+1} + b_w - p^m]$$

and Lemma C.2.

To sum up, the embeddings generated in this block are:

- Positions with the correct value of s_w:

$$\boldsymbol{e}^5 = (q_{w+1} + b_w, b_w, \mathbf{1}_{q_{w+1}+b_w \geq p^m},\ \mathbf{1}_{q_{w+1}+b_w \leq p^m-2},\ \mathbf{1}_{b_w \geq p^m},\ \mathbf{1}_{b_w \leq p^m-2}, w).$$

- Other positions: $\boldsymbol{e}^6 = (-1, -1, -1, -1, -1, -1, -1)$. (This can be implemented by filtering out the unfeasible values using Lemma C.3).

Block 6. This block of the Transformer first calculates the following values for the positions with the correct value of s_w:

- The minimum $w_1 \geq w$ such that $\mathbf{1}_{q_{w+1}+b_w \geq p^m} = 1$.
- The minimum $w_2 \geq w$ such that $\mathbf{1}_{q_{w+1}+b_w \leq p^m-2} = 1$.

Both are standard COPY operation, thus can be implemented by Lemma C.5. To make sure the value of w_1, w_2 is valid (i.e., the existence of w_1, w_2), we can COPY $\mathbf{1}_{q_{w_1+1}+b_{w_1} \geq p^m}, \mathbf{1}_{q_{w_2+1}+b_{w_2} \leq p^m-2}$ and check whether they are 1, and filter out the invalid values by MLP using Lemma C.3. The embeddings generated in this block are:

- Positions with the correct value of s_w: $\boldsymbol{e}^6 = (w_1, w_2)$.
- Other positions: $\boldsymbol{e}^6 = (-1, -1)$. (This can be implemented by filtering out the unfeasible values using Lemma C.3).

Block 7. The last block of the Transformer executes the final four steps of Algorithm 2. This layer calculates the carry-over bits $\boldsymbol{c}$ and p^m-adic representation of the final output $\boldsymbol{o}$ via the attention mechanism and the MLP, subsequently converting the p^m-adic number into a p-adic number.

The computation of carry-on bits, as described in Equation (8) within Algorithm 2, adheres to the following equations:

$$\begin{aligned} i_\wedge &= \max\{w \leq i \mid q_w + b_{w-1} \geq p^m\}, \\ i_\vee &= \max\{w \leq i \mid q_w + b_{w-1} \leq p^m - 2\}, \\ c_i &= \mathbf{1}_{i_\wedge > i_\vee}. \end{aligned} \tag{11}$$

In the attention layer, operations are restricted to output tokens and other tokens will maintain the embeddings via the residual connection and the filter operation by MLP. Let's consider the token $o_{(i+1)m+j+1}$, where $j \in \{0, \cdots, m-1\}$, we want to predict the next token $o_{(i+1)k+j}$. The model executes the COPY operation, duplicating the previous embeddings to extract $q_{i+1} + b_i$, $i_\wedge$, and $i_\vee$. The extraction is similar to previous blocks, but here we only need to focus on positions with correct value of s_w. To find out the value of $i_\wedge, i_\vee$, we first COPY the embedding of the position with the correct value of s_i, and find the minimum w' which shares the same value of w_1, w_2 with s_i. Again, this can be implemented by several COPY operation with Lemma C.5.

The carry-over bit c_i and the p^m-adic results $\tilde{o}_{i+1}$ are then computed as follows:

$$c_i = \mathbf{1}_{i_\wedge > i_\vee},\ \tilde{o}_{i+1} = b_i + c_i + q_{i+1}.$$

This computation is facilitated by a constant-size MLP. Subsequently, for the output token $\tilde{o}_{(i+1)k+j}$, the result $o_{(i+1)k+j} = \tilde{o}_{i+1} \bmod p^{j+1}$ is required. We first calculate $\tilde{o}_{i+1}/p^{j+1}$ using the positional embedding and Lemma C.1, then calculate $\lfloor \tilde{o}_{i+1}/p^{j+1} \rfloor$ using the similar fashion to what we did in Block 4, and then calculate $\tilde{o}_{i+1} \bmod p^{j+1}$ using MLP. Finally, we can get the value of $\lfloor \frac{\tilde{o}_{i+1} \bmod p^{j+1}}{p^j} \rfloor$ using the similar fashion to what we did in Block 4.

Upon outputting the token o_0, the model anticipates the <EOS> token, employing an MLP to filter the hidden embeddings and output the word embedding for <EOS>. Thus, the final output from this layer is characterized by the equation:

$$\boldsymbol{e}^7_{o,i} = \begin{cases} (o_{i-1}, i, 0) & \text{if } i > 0, \\ (-1, -1, 1) & \text{if } i = 0. \end{cases}$$

Predict Next Token. Given the output embeddings of the last transformer layer $\boldsymbol{e}^7_{o,i}$, and the word embeddings, the transformer can simply predict the next token by finding the nearest word embeddings.

In this construction, the norm of the parameters is bounded by $poly(n, k)$, therefore, this construction can be implemented by a log-precision transformer with arbitrarily small error. □

Theorem 5.3. *For any integers n, l, and k, there exists a polynomial $f(n)$ such that a logarithmic-precision Transformer with constant depth and hidden dimensions $f(n)$ can generate the correct output for any input on the* $\mathrm{MUL}(n, l, p)$ *task.*

Here, we first describe an algorithm to perform $\mathrm{Mul}(n, l, p)$ (Algorithm 3) and prove the correctness of Algorithm 3. Then, we construct a Transformer with the configurations in Theorem 5.3 capable for simulating Algorithm 3.

Algorithm 3: $\mathrm{Mul}(n, l, p)$ Algorithm

Input : Two p-adic numbers $\boldsymbol{a}, \boldsymbol{b}$ no longer than n bits, truncating length l
Output : $\boldsymbol{o} := \boldsymbol{ab} \bmod p^l$

$m = \lceil \log_p n \rceil + 1$;
Compute the product of each pair of bits: $d_{i,j} = a_i b_j$;
Compute each bit as

$$r_j = \sum_{k=\max(0,j-(n-1))}^{\min(n-1,j)} d_{k,j-k}$$

for $j = 0, \cdots, 2n-1$;
Combine neighboring m bits:

$$s_i = \sum_{j=0}^{m-1} r_{ik+j} p^j$$

for $i = 0, \cdots, \lfloor (2n-1)/m \rfloor$;
Decompose s_i by $s_i = b_i p^m + q_i$, where $q_i \in [0, p^m - 1]$ and $b_i, q_i \in \mathbb{N}$;
$b_{-1} = 0$;
foreach $i = 0, \cdots, \lfloor (2n-1)/m \rfloor$ **do**
  $f_i = \mathbf{1}_{q_i + b_{i-1} \geq p^m}$;
  $g_i = \mathbf{1}_{q_i + b_{i-1} \geq p^m - 2}$;
end
Compute the carry-on bits $\boldsymbol{c}$:

$$c_i = \bigvee_{0 \leq j \leq i} \left(f_j \wedge \bigwedge_{j \leq k \leq i} g_k \right)$$

for $i = 0, \cdots, \lfloor (2n-1)/m \rfloor$;
Compute the p^m-adic outcome $\tilde{\boldsymbol{o}}$: $\tilde{o}_i = (q_i + b_{i-1} + c_{i-1}) \bmod p^m$ for $i = 0, \cdots, \lfloor (2n-1)/m \rfloor$;
Covert p^m-adic $\tilde{\boldsymbol{o}}$ to p-adic $\boldsymbol{o}$:

$$o_i = \left\lfloor \frac{\tilde{o}_j \bmod p^{(l+1)}}{p^l} \right\rfloor$$

for $i = jk + l$ where $l \in \{0, \cdots, k-1\}$, $j \in \mathbb{Z}$;

Lemma E.2 (An algorithm to perform $\mathrm{Mul}(n, l, p)$)**.** *Algorithm 3 outputs $\boldsymbol{o} = \boldsymbol{ab} \bmod p^l$ for all inputs $\boldsymbol{a}, \boldsymbol{b}$.*

Proof. It's easy to verify $\sum_i s_i p^{im}$ accurately represents the product of $\boldsymbol{a}, \boldsymbol{b}$. For the subsequent steps, the proof is the same as that of Lemma E.1 since they share the same procedures. □

Next, we provide the proof for Theorem 5.3.

Proof for Theorem 5.3. Now, we demonstrate that a log-precision transformer, with a constant depth, a fixed number of attention heads, and $O(n^2)$ embedding dimensions, is capable of simulating Algorithm 3. Consequently, this model can accurately generate correct output for any input integers $\boldsymbol{a}, \boldsymbol{b}$.

Initial Embeddings: The total length of the input sequence is no longer than $2(n+1)$. We categorize the tokens into two classes: number tokens $(0, 1, \cdots, p-1)$ and auxiliary tokens ($+$, $=$, <SOS> and <EOS>). Given the parameters k, n, we determine the parameter $m = \lceil \log_p k \rceil + 1 \geq 2$, as specified in Algorithm 3. The embeddings for these classes are defined as follows:

- **Embedding of input token a_i:** $\boldsymbol{u}^0_{a,i} = \big(a_i \boldsymbol{e}_{i+1}, 0, -1, -1, 0, 1, i, 0, \text{ape}_{a,i}\big)$.
- **Embedding of input token b_i:** $\boldsymbol{u}^0_{b,i} = \big(0, b_i \boldsymbol{e}_{i+1}, -1, -1, 0, 2, i, 0, \text{ape}_{b,i}\big)$.
- **Embedding of the "$\times$" token:** $\boldsymbol{u}^0_{\times} = (-1, -1, -1, -1, -1, 4, -1, 0, \text{ape}_{\times})$.
- **Embedding of the "$=$" token:** $\boldsymbol{u}^0_{=} = (-1, -1, -1, -1, -1, 5, -1, 0, \text{ape}_{=})$.
- **Embedding of the <SOS> token:** $\boldsymbol{u}^0_{\text{<SOS>}} = (-1, -1, -1, -1, -1, 6, -1, 0, \text{ape}_{\text{<SOS>}})$.
- **Embedding of the <EOS> token:** $\boldsymbol{u}^0_{\text{<EOS>}} = (-1, -1, -1, -1, -1, 7, -1, 0, \text{ape}_{\text{<EOS>}})$.
- **Embedding of output token o_i:** $\boldsymbol{u}^0_{o,i} = (-1, -1, o_i, \boldsymbol{e}_{\lfloor i/m \rfloor}, -1, 3, i, p^{-(i \bmod m)}, \text{ape}_{o,i})$.

where $\boldsymbol{e}_i \in \mathbb{R}^n$ is one-hot vector, and $\text{ape}_{\dots}$ is absolute positional embedding. In this construction, the first $3n+3$ dimensions of each initial embedding represent the word embedding, while the last three dimensions accounts for the position embedding.

Block 1. The first block of the Transformer executes the first three lines of Algorithm 3. To be specific, we first aggregate the input number $\boldsymbol{a}, \boldsymbol{b}$ to the positions of b_0, and then calculate the values of r_j.

To aggregate the input number $\boldsymbol{a}, \boldsymbol{b}$ to the positions of b_0, we set the query, key and value as follows:

- Query: $\boldsymbol{q} = (\boldsymbol{e}^0[2n+2])$, i.e., $\boldsymbol{q} = (0)$ for input number $\boldsymbol{a}, \boldsymbol{b}$, and $\boldsymbol{q} = (-1)$ otherwise.
- Key: $\boldsymbol{k} = (1)$.
- Value: $\boldsymbol{v} = \boldsymbol{e}^0[1, \cdots, 2n]$.

Thus $\langle \boldsymbol{q}, \boldsymbol{k} \rangle = 0$ for key of input number tokens, and $\langle \boldsymbol{q}, \boldsymbol{k} \rangle \leq -1$ otherwise. By Lemma C.6, the attention output is

$$\frac{1}{\text{ape}_{b,0} - 2}(a_0, \cdots, a_{n-1}, b_0, \cdots, b_{n-1}).$$

By Lemma C.1, we can use the subsequent MLP to get $(a_0, \cdots, a_{n-1}, b_0, \cdots, b_{n-1})$ given the value of $\text{ape}_{b,0}$. Then we can calculate all $d_{i,j}$ using the MLP, which requires $O(n^2)$ hidden dimension by Lemma C.1.

Finally, we calculate $(r_{2n-1}, \cdots, r_0)$ by

$$r_j = \sum_{k=\max(0, j-(n-1))}^{\min(n-1, j)} d_{k, j-k}.$$

Block 2. This block of the Transformer uses several MLPs to executes line 4-12 of Algorithm 3. All the calculations below are also calculated at the position of b_0, subsequent to what we did in Block 1.

- For the calculation of s_i, it's easy to get the values via $(r_{2n-1}, \cdots, r_0)$.

- For the calculation of b_i, q_i, notice that $b_i \le p^m \le np^2$, thus we can use

$$b_i = \sum_{j=0}^{np^2} \mathrm{ReLU}(s_i - p^m)$$

for each b_i, which requires $O(n^2)$ hidden dimension in total by Lemma C.2. Then $q_i = s_i - b_i p^m$, which can be easily implemented by MLP as well.

- For the calculation of f_i, g_i, we can get those values by

$$\begin{aligned} f_i &= \mathrm{ReLU}[q_i + b_{i-1} - (p^m - 1)] - \mathrm{ReLU}[q_i + b_{i-1} - p^m], \\ g_i &= \mathrm{ReLU}[q_i + b_{i-1} - (p^m - 2)] - \mathrm{ReLU}[q_i + b_{i-1} - (p^m - 1)] \end{aligned}$$

and Lemma C.2, which requires $O(n)$ hidden dimension in total.

- For the calculation of c_i, notice that

$$\bigwedge_{1 \le i \le \gamma} \alpha_i = \mathrm{ReLU}\left(\sum_{i=1}^{\gamma} \alpha_i - \gamma + 1\right), \quad \bigvee_{1 \le i \le \gamma} \alpha_i = 1 - \mathrm{ReLU}\left(1 - \sum_{i=1}^{\gamma} \alpha_i\right).$$

Combining with Lemma C.2, we can calculate the value of each c_i with $O(n)$ hidden dimension.

- Finally, for the calculation of $\tilde{o}_i$, we can use the similar fashion of the calculation of q_i. Since $q_i + b_{i-1} + c_{i-1} < 2p^m$, we can calculate each $\tilde{o}_i$ using constant hidden dimension, which implies we can calculate $\tilde{\boldsymbol{o}}$ using $O(n)$ hidden dimension in total.

Block 3. The last block of the Transformer executes the last step of Algorithm 3. Let's consider the token $o_{(i+1)m+j+1}$, where $j \in \{0, \cdots, m-1\}$, we want to predict the next token $o_{(i+1)k+j}$. We first COPY the value of $\tilde{\boldsymbol{o}}$ from the position of b_0, then extracts $\tilde{o}_{i+1}$ by $\tilde{o}_{i+1} = \langle \tilde{\boldsymbol{o}}, \boldsymbol{e}_{i+1} \rangle$ using the positional embedding of $\boldsymbol{u}^0_{o,i}$.

Subsequently, for the output token $o_{(i+1)k+j}$, the result $o_{(i+1)k+j} = \tilde{o}_{i+1} \bmod p^{j+1}$ is required. We first calculate o_{i+1}/p^{j+1} using the positional embedding and Lemma C.1, then calculate $\lfloor \tilde{o}_{i+1}/p^{j+1} \rfloor$ using the similar fashion to what we did when calculating s_i, b_i in Block 2. Since $\tilde{o}_{i+1} < 2p^m \le np^2$, this can be implemented by a MLP with $O(n)$ hidden dimension. Then we can calculate $\tilde{o}_{i+1} \bmod p^{j+1}$ using MLP. Similarly, we can finally get the value of $\lfloor \frac{\tilde{o}_{i+1} \bmod p^{j+1}}{p^j} \rfloor$ using a MLP with $O(n)$ hidden dimension.

Upon outputting the token o_0, the model anticipates the <EOS> token, employing an MLP to filter the hidden embeddings and output the word embedding for <EOS>. Thus, the final output from this layer is characterized by the equation:

$$\boldsymbol{e}^3_{o,i} = \begin{cases} (o_{i-1}, i, 3) & \text{if } i > 0, \\ (-1, -1, 7) & \text{if } i = 0. \end{cases}$$

Predict Next Token. Given the output embeddings of the last transformer layer $\boldsymbol{e}^3_{o,i}$, and the word embeddings, the transformer can simply predict the next token by softmax.

In this construction, the norm of the parameters is bounded by $O(n^2)$, therefore, this construction can be implemented by a log-precision transformer with arbitrarily small error. □

F Experimental Details

In this section, we present the experimental details.

F.1 Datasets

The iterated addition and integer addition data are generated according to Algorithm 4. The multiplication data are generated according to Algorithm 5. Both datasets are used online for training and testing.

Algorithm 4: Iterated Addition Data Generation

```
Function large_number_add(a, b, base):
    Input: a: List of digits of the first number
           b: List of digits of the second number
           base: The numerical base
    Output: result: List of digits of the sum of a and b
    carry ← 0, result ← []
    max_length ← max(length(a), length(b))
    for i ← 0 to max_length - 1 do
        sum ← carry
        if i < length(a) then
            sum ← sum + a[i]
        end
        if i < length(b) then
            sum ← sum + b[i]
        end
        carry ← floor(sum / base)
        result.append(sum mod base)
    end
    if carry ≠ 0 then
        result.append(carry)
    end
    return result
Function get_data(batch, length, num_count, base):
    Input:
    batch: Number of samples
    length: Maximum length of addends
    num_count: Number of addends
    base: The numerical base
    Output: tokenized_data: Tensor of generated sequences
    data ← random integers in range [0, base) with shape (batch, length, num_count)
    tokenized_data ← []
    for i ← 0 to batch − 1 do
        numbers ← data[i, :, :]
        strip leading zeros of numbers and get stripped_numbers
        for num in numbers do
            sum_digits ← large_number_add(sum_digits, num, base)
        end
        reverse stripped_numbers and sum_digits
        add token of '+' and '=' and '<EOS>' to form sequence pad the sequence into the same
          length
        tokenized_data.append(sequence)
    end
    convert tokenized_data to tensor
    return tokenized_data
```

Algorithm 5: Integer Multiplication Data Generation

```
Function large_number_mult(a, b, base):
    Input: a: List of digits of the first number
           b: List of digits of the second number
           base: The numerical base
    Output: result: List of digits of the product of a and b
    result ← [0] * (length(a) + length(b))
    for i ← 0 to length(a) − 1 do
        carry ← 0
        for j ← 0 to length(b) − 1 do
            product ← a[i] ∗ b[j] + result[i + j] + carry
            carry ← floor(product / base)
            result[i + j] ← product mod base
        end
        if carry > 0 then
            result[i + length(b)] ← result[i + length(b)] + carry
        end
    end
    strip leading zeros from result
    return result
Function get_mult_data(batch, length, base):
    Input:
    batch: Number of samples
    length: Maximum length of multiplicands
    base: The numerical base
    Output: tokenized_data: Tensor of generated sequences
    data ← random integers in range [0, base) with shape (batch, length, 2)
    tokenized_data ← []
    for i ← 0 to batch − 1 do
        num_1 ← data[i, :, 0]
        num_2 ← data[i, :, 1]
        strip leading zeros of numbers and get stripped_numbers
        product_digits ← large_number_mult(num_1, num_2, base)
        reverse stripped_numbers and product_digits
        add token of '×' and '=' and '<EOS>' to form sequence pad the sequence into the same
          length
        tokenized_data.append(sequence)
    end
    convert tokenized_data to tensor
    return tokenized_data
```

	Base 2		Base 10	
Length	`float32` **Accuracy**	`bfloat16` **Accuracy**	`float32` **Accuracy**	`bfloat16` **Accuracy**
8	99.8%	99.6%	99.4%	99.0%
16	99.3%	98.4%	99.2%	98.1%
24	98.9%	96.3%	99.2%	97.4%
32	99.3%	95.9%	99.2%	94.1%

Table 2: Evaluation of integer addition accuracy across various length with both 32-bit and 16-bit precision.

F.2 Model Training

We tried 3 different seeds and select the maximum accuracy for each task. All the experiments were conducted on a single NVIDIA GeForce RTX 4090 GPU over a duration of two weeks.

F.3 Integer Addition Results

The results of the experiments are presented in Table 2.

Generation Configuration	
TopK	50
TopP	0.95
Temperature	0.1

Table 3: Generation Configuration for LLAMA 3.1 8B Instruct in arithmetic tasks.

F.4 Prompt For LLM

Detailed prompts for the three elementary arithmetic tasks are listed in the Tables 4 and 5 and generation configuration can be found in the Table 3.

Prompt for LLAMA 3.1 8B Instruct in Integer Addition and Iterated Addition tasks.
Please directly calculate the following arithmetic expression in base base with the following format: <Expression> = <Result> It is important that you should not show any intermediate steps in your calculation process. The final answer should be computed in one step and provided the final result immediately without any explanation. Here are some examples 32 + 78= 110 1234 + 4567 + 2134 + 4567 = 12502 2135 + 523 + 2135 + 523 = 5316 2314 + 4567 + 2314 + 4567 = 13762 Arithmetic Expression: <Expression>

Table 4: Prompt for LLAMA 3.1 8B Instruct in Integer Addition and Iterated Addition tasks.

Prompt for LLAMA 3.1 8B Instruct in Integer Multiplication task.
Please directly calculate the following arithmetic expression in base base. It is important that you should not show any intermediate steps in your calculation process. The final answer should be computed in one step and provided the final result immediately without any explanation. Here are some examples Examples: 32 * 56 = 1792 867 * 467 = 404889 123 * 456 = 56088 Arithmetic Expression: <Expression>

Table 5: Prompt for LLAMA 3.1 8B Instruct in Integer Multiplication task.