Scaling Behavior for Numeral System: Tokenize Your Numbers into 1-digit

Anonymous ACL submission

Abstract

Though Large Language Models (LLMs) have shown remarkable abilities in mathematics reasoning, they are still struggling with performing numeric operations accurately, such as addition and multiplication. Numbers can be tokenized into tokens in various ways by different LLMs and affect the numeric operations performance. Currently, there are two representatives: 1) Tokenize into 1-digit, and 2) Tokenize into $1 \sim 3$ digit. The difference is roughly equivalent to using different numeral systems (namely base 10 or base 10^3). In light of this, we study the scaling behavior of different numeral systems in the context of transformer-based large language models. We empirically show that a base 10 system is consistently more data-efficient than a base 10^2 or 10^3 system across training data scale, model sizes under from-scratch training settings, while different number systems have very similar performances when fine-tuned. Through thorough analysis and experiments, we conclude that tokenizing numbers into 1-digit is more favorable for LLMs in numerical operations. Additionally, we reveal *extrapolation* behavior patterns on addition and multiplication that sheds light on the mechanism learnt by the models.

1 Introduction

Large Language Models (LLMs) have stormed the world with their amazing reasoning abilities (OpenAI, 2023; Google, 2023; Touvron et al., 2023b). However, numeric operations remain challenging for LLMs to comprehend under the architecture of Transformer (Vaswani et al., 2017; Lee et al., 2023; Yuan et al., 2023; Zhou et al., 2024; McLeish et al., 2024). Several techniques have been proposed to improve the performance of numeric operations including improving positional embeddings (Kazemnejad et al., 2024; McLeish et al., 2024) and using scratchpad (Nye et al., 2021; Liu and Low, 2023). These works mostly focus on a random initialized Transformer with 1-digit tokenization. However, pre-trained LLMs have various tokenizers that can affect the numeric operations performances. Currently, there are two main tokenization schemes: 1) Tokenize into 1-digit (Touvron et al., 2023a,b; Jiang et al., 2023; Bai et al., 2023; Team et al., 2024; Shao et al., 2024), and 2) Tokenize into $1 \sim 3$ digit (Biderman et al., 2023; OpenAI, 2023; Cai et al., 2024). An example of different tokenization is shown in Table 1. Abstracting away practical details of tokenizers, these two schemes can be viewed as using a base 10 numeral system versus a base 10^3 system. The former aligns better with human intuition and the prevalent base 10 system in daily usage. Yet, the latter encodes numbers into fewer tokens. Our question follows intuitively: What is the difference between these schemes in numeric operations?

We resort to data-scaling efficiency to answer this question. That is, there would be substantial differences in the scaling behavior of these numeral systems. More essentially, we are inspecting the scaling proclivity towards learning *token classification* or *length information* for numeric operations. Intuitively, a base 10 system has a smaller set of tokens that could appear in the input but would be more lengthy in decoding the same number than a base 10^3 system. Out of practical considerations, we choose to restrict our study to the base 10, base 10^2, and base 10^3 systems which adhere to tokenizers of existing large language models.

To design experiments for scaling behavior, we identify the following critical dimensions: 1) numeral system 2) data scale, and 3) model size. To further corroborate the generalizability of our claim, we also test if our conclusion holds for different numeric operations. On the other hand, it is possible that pre-trained models have a bias towards $2 \sim 3$ digit tokens. To strengthen our claim, we test if our observed trend holds irrespective of whether our models are trained from-scratch or

Type	Models	Tokenize 31415926535
1-digit	Llama1-2/Mistral/QWen/Gemma	['3', '1', '4', '1', '5', '9', '2', '6', '5', '3', '5']
Multiple	Pythia/GPT-4o/Llama3/InternLM	['314', '159', '265', '35']

Table 1: Tokenizing 31415926535 via different large language models.

fine-tuned. Our contributions can be summarized as follows:

- A base 10 system is consistently more data-efficient than a base 10^2 or base 10^3 system under different data scales, model sizes, and different operators, especially training from-scratch.
- We study the effects of data scales and model sizes on the metrics of interest, and showcase that performance generally improves with these two factors.
- We identify several calculation patterns in the extrapolation setting including truncated addition and base-10 carry extrapolates at longer input lengths.

2 Related Work

Numeral System A numeral system represents a number by a sequence of tokens within pre-defined sets. In order to perform numeric operations, the model would have learned to discern between the tokens precisely. Numeral systems are closely related to tokenizers. We first review prevalent tokenization conventions. Llama1/2 (Touvron et al., 2023a,b) tokenize numbers into 1-digit, enforcing a base-10 system. This design is also adopted by other general-domain LLMs (Jiang et al., 2023; Bai et al., 2023; Team et al., 2024) and math-specialized model Deepseek-Math (Shao et al., 2024). On the other hand, the most capable model of date, GPT-4o, tokenizes numbers into $1 \sim 3$ digit, which is roughly equivalent to using a base 10^3 system. To the best of our knowledge, no one has systematically studied how the numeral system affects the transformers' arithmetic ability.

Arithmetic Operations in Transformers To improve the arithmetic abilities of the transformer (Wang et al., 2021; Nogueira et al., 2021), people have designed positional embeddings (Kazemnejad et al., 2024; McLeish et al., 2024), scratchpad (Nye et al., 2021), and special training procedures (Liu and Low, 2023; Deng et al., 2023). In this paper, we do not improve the performance of arithmetic operations, but to understand the scaling impact of choices of numeral systems. We focus on using decoder-only transformers to decode the results of arithmetic operations directly (i.e. without a scratchpad).

Scaling Laws in Large Language Models Scaling laws have been widely studied in the context of LLMs (Kaplan et al., 2020; Hernandez et al., 2021; Gao et al., 2023; Bi et al., 2024) which aims for predicting model losses based on different data scale and model parameters. Different from this line of research, we do not aim to accurately predict performances when we scale up computing. We leverage scaling behavior as a proxy to study the impact of numeral systems selection.

3 Scaling Behavior Experiment Designs

To understand how the numeral systems affect numeric operation in LLMs, we identify the following dimensions of interest for our experiments when training an LLM with numerical operation: 1) *numeral system* 2) *training data scale* 3) *model size* 4) *from-scratch* or *fine-tuning* 5) *different operations*.

For 1) and 2), we generate synthetic inputs according to the process explained in section 3.1. For 3), we make use of the Pythia scaling suite (Biderman et al., 2023) for ranging over different model sizes. For 4), we replicate experiments for both settings to the best of our effort. For 5), we choose to include results of addition and multiplication.

We list the complete configurations for our experiments. 1) numeral system: base 10, base 10^2, base 10^3 2) training data scale: $2^{13\sim19}$ training samples 3) model size: 70M, 410M, 1.4B, 6.9B, 12B from Pythia 4) random-initialized or fine-tune from Pythia (i.e. from-scratch or fine-tune) 5) operations: addition, multiplication. After choosing a configuration, we train our model based on our generated samples and evaluate the model on a non-overlapped evaluation set. The training procedure is the same as training an instruction language model which masks the input (for example $12 + 23 =$) and only calculates the losses on the outputs (35).

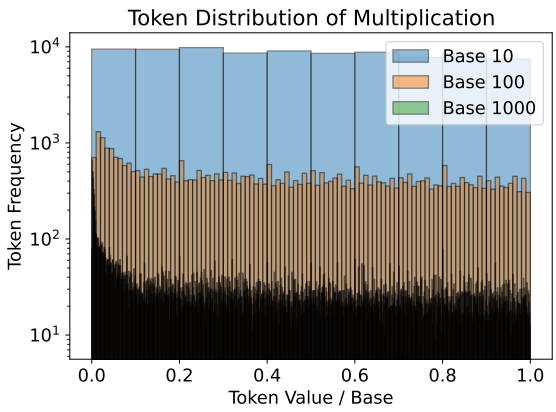

Figure 1: Answer Token Distribution for Multiplication. We sample 2^{13} addition samples to illustrate the distribution. Token values are normalized to $[0, 1]$.

3.1 Synthetic Data Generation

We generate synthetic data of scales $2^{13\sim19}$ for numeral systems base 10, base 10^2, and base 10^3. We abstract away the nitty-gritty details involved in practical tokenization schemes and generate synthetic input ids and labels directly.

We first illustrate our training distribution generation process using *addition* as an example. Let a and b be two operands, each row would be in the form of $a+b=c$. Let la and lb be the digit lengths of a and b in base 10. We fix $la, lb \in [1, 10]$, and we attempt to evenly distribute generated data over $la \times lb$. If the total number of pairs for $la \times lb$ is smaller than we request, we take all possible pairs. No pairs are repeated during our generation.

Based on our generated training distribution, we convert each number into the corresponding base 10, base 10^2, and base 10^3 representations. Note that this could be easily done by grouping digits in the original base 10 representation. We then map the digit numbers onto their corresponding token ids. Intuitively, base 10 would have 10 ids (corresponding to $0 \sim 9$), base 10^2 would have 100 ids, and base 10^3 would have 1000 ids. In Figure 1, we demonstrate the answer token distribution for each numeral system. We obtain the distribution by converting all answers into ids. We normalize the token values from each numeral system by dividing against the base. As the base gets larger, the probability density gets more imbalanced.

3.2 Evaluation Setup

We sample non-overlapping operand pairs for evaluation. We attempt to evenly sample 1000 pairs for each $la \times lb$. If half of the total number of pairs is smaller than 1000, then we reserve half for evaluation. Overall, we strive to make sure that the training and evaluation sets are from the same distribution and have no overlap.

To observe a clear trend, we record the following metrics 1) relative error 2) exact match accuracy 3) normalized edit distance. Of these three metrics, exact match accuracy is the most intuitive, which is a hard match between model-generated tokens and ground truth tokens. Based on our initial experiments, this metric is not informative enough for a range of settings. We thus design two more metrics to reveal the underlying dynamics of our models.

Relative Error To generate a metric that is meaningful to practical settings, we calculate *relative error* as $\left|\log \frac{o}{gt}\right|$, where gt is the ground truth answer and o is the model output. We then compute the mean of the magnitude difference over all evaluation pairs. This metric is more informative than exact match accuracy since it captures the relative error made by the model.

Note that this metric has two inductive biases. First, this metric gives more weight to the length difference between model outputs and ground truth answers. Even if the output has a long common sub-sequence with the ground truth, it will still be penalized for not getting the output length right.

Second, this metric biases towards the accuracy of leading digits. If we make a connection between the numeral system and signal processing, this is equivalent to putting more weight on *low frequency* component of the number (trailing digits change rapidly while leading digits change slowly).

Generalized Normalized Edit Distance Since numbers in a numeral system are sequences of tokens, we introduce a generalized normalized edit distance metric, which would give credit to partially correct answers based on string similarity. *Edit Distance* is a powerful metric that can capture substring similarities. We extend this metric to our scenario using the following setup:

Each number could be represented as a sequence of chars (instead of tokens) with each char taking the range of $0 \sim 9$. We define the *generalized edit distance* as the minimum number of insertions, deletions, and substitutions needed to transform one sequence into another. Suppose that the two sequences are $A = a_1a_2...a_n$ and $B = b_1b_2...b_m$. Let ed be the edit distance be-

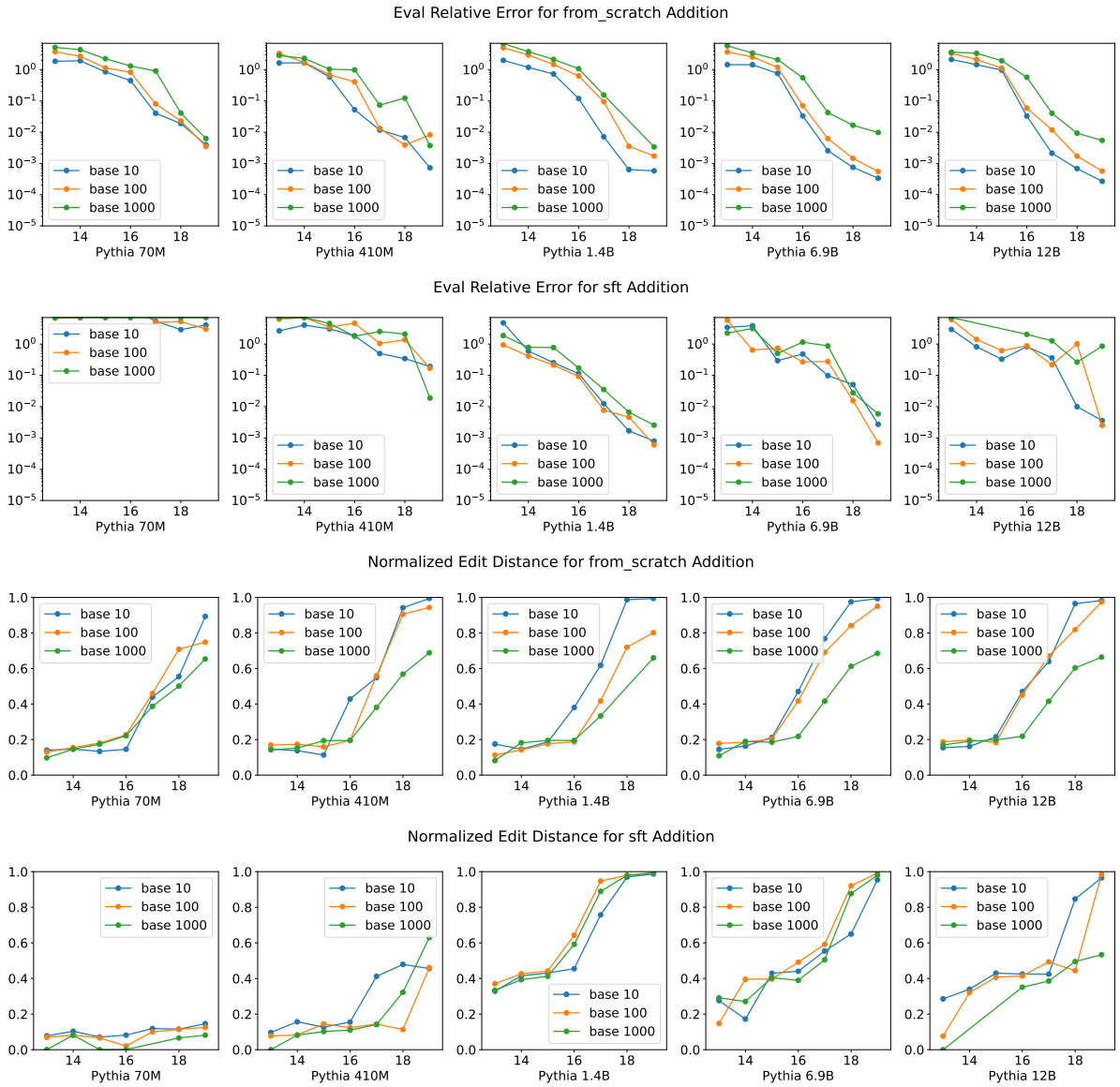

Figure 2: Relative error and normalized edit distance for addition operation with different data scales, model parameter sizes, from-scratch or fine-tune, and numeral systems.

tween A and B. We define the *normalized edit distance* as $ned = \frac{\max(m,n) - ed}{\max(m,n)}$. This metric is normalized into $[0, 1]$.

Compared with the *Relative Error* metric, this metric connects more closely to human perception. It prioritizes answers that would have the longest sub-sequences with the ground truth. Since human perception is largely visual for numbers, this metric aligns more with the *visual similarity* between the answer and the ground truth.

Note that *Relative Error* can be somewhat viewed as a revised version of the *Normalized Edit Distance* we used, where *insert* and *delete* operations are penalized harder, and *replace* operation is reweighted by the magnitude of the difference.

4 Experiments and Results

In this section, we present the main results of our experiments that demonstrate the scaling efficiency of different numeral systems. There are some differences between the results of *addition* and *multiplication*, which could have arisen because of the distinct difficulties of the tasks, but the overall trend is consistent.

4.1 Overall Trends

For each scenario, our main metrics of interest are *Relative Error* and *Normalized Edit Distance*. For the addition operation, we also report *Exact*

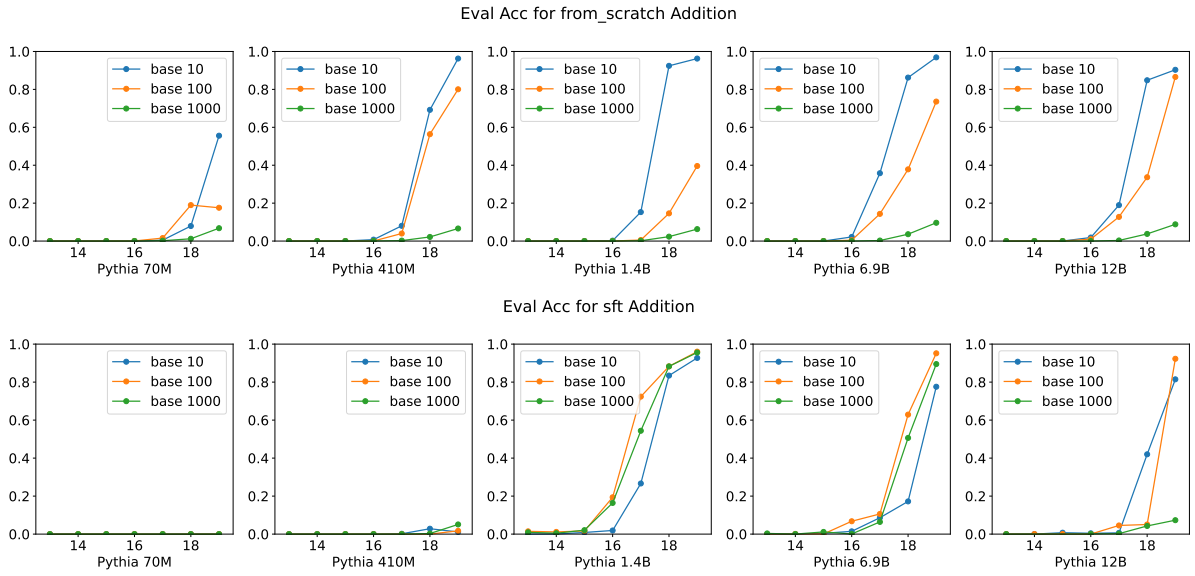

Figure 3: Exact match accuracy for addition operation with different data scales, model parameter sizes, from-scratch or fine-tune, and numeral systems.

Match Accuracy. However, for multiplication, exact match accuracy is too low such that no information could be gained.

Overall, a base 10 system is **consistently more data-efficient** than a base 10^2 or a base 10^3 system when trained from scratch, as shown in Figure 2 and Figure 4. That is, to obtain a certain performance, a base 10 system would need data of a smaller scale to achieve it. We highlight the fact that do not display trends that favor base 100 and base 1000. During pre-training, most tokenizers lean towards combining consecutive digits, which would have favored base 10^2 and base 10^3 over base 10. Considering this, the decent performances of base 10 fine-tuned models further corroborate the superiority of the base 10 system.

When we plot *Relative Error* against data scale, we are essentially using a log scale for both the x and y axes. Along the x-axis, the training data sample number grows exponentially. Along the y-axis, the metric of interest grows exponentially. Under such a scale, there is a near-linear relationship for most graphs. This connects with scaling laws described in Kaplan et al. (2020). On the other hand, *Normalized Edit Distance* is already near-linear against the log of data samples. This could be explained by the fact that *Edits* are made to positions of the number, which affects the magnitudes of numbers exponentially.

We find larger models usually obtain better performances under different scenarios. Models larger than 1.4B show a different trend compared to models less than 1.4B.

4.2 Interpolation evaluation

4.2.1 Addition

In Figure 2, we showcase the scaling behavior for addition. We first focus on *from-scratch* scenario. We can observe a clear trend that base 10 is consistently better than base 10^2 and base 10^3 for both metrics, which is a strong affirmation of our claim. Such a trend indicates that a base 10 system is at least of a constant magnitude more data efficient than base 10^2 and base 10^3 systems, and this trend does not diminish as the model size gets larger.

For fine-tuning experiments, the difference between numeral systems is less profound. Using *NED* as a metric, all numeral systems showcase similar performances. On the whole, a base 10 system is at least on par with base 10^2 or base 10^3, as we do not observe an exaggerated performance difference as we scale up data. Pythia is pre-trained on tokenization with base 10^3, which weakens the advantages of base 10.

4.2.2 Multiplication

We plot the *Normalized Edit Distance* for multiplication in Figure 4. We can also conclude that the base 10 number system is consistently more data-efficient than base 10^2 and base 10^3. The trend is consistent for both *from-scratch* and *supervised fine-tuning* settings.

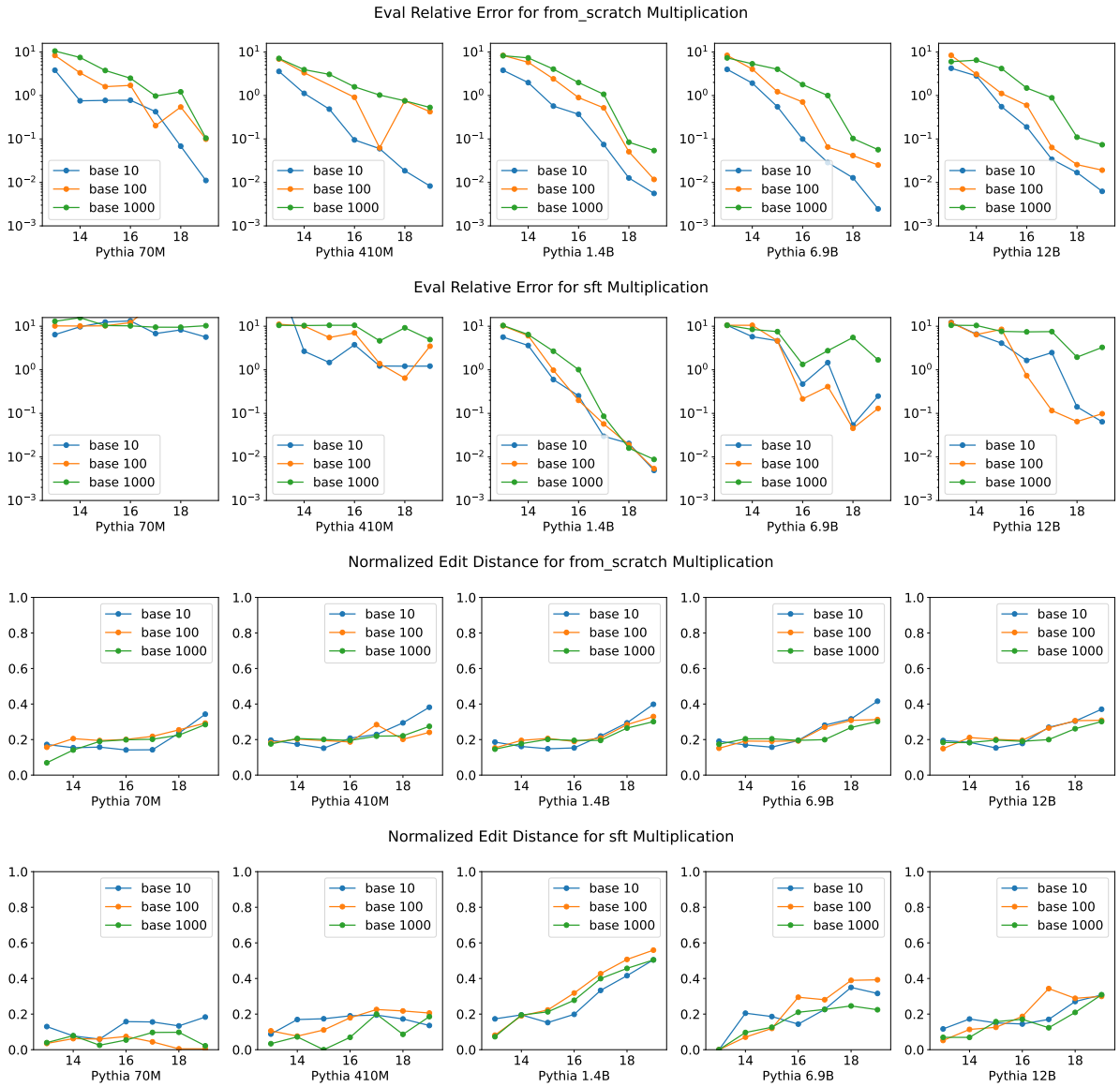

Figure 4: Relative error and normalized edit distance for multiplication operation with different data scales, model parameter sizes, from-scratch or fine-tune, and numeral systems.

The superiority of a base 10 system is more pronounced and more consistent under the multiplication setting. First, for Relative Error of models trained from scratch, the advantage of a base 10 system is more perceivable than the addition setting. For the Normalized Edit Distance metric, we observe a trend where the data efficiency of a base 10 system gains more advantage at large data scales. We relate this phenomenon to the differences between Figure 7 and Figure 6 in the Appendix. As shown, addition is a much simpler task as compared with multiplication. For a large range of operand length pairs, the exact match accuracy remains zero. The hypothesis that we have proposed is that the sample efficiency of a base 10 system against base 100 and base 1000 systems is magnified by the difficulty of the task.

4.3 Extrapolation evaluation

In our previous evaluations, we have tested whether our trained models could interpolate between the points identically sampled from the same training distribution (i.e. whether our models could generalize in-domain). An equally important question is whether our models could extrapolate to unseen data points, especially in terms of length.

During training distribution generation, we only consider numbers that are less than 10^{11}. Therefore, we generate cases where one operand lies in the range of $10^{11} \sim 10^{16} - 1$, and the other operand

ranges from $1 \sim 10^{16} - 1$. To this end, we perform

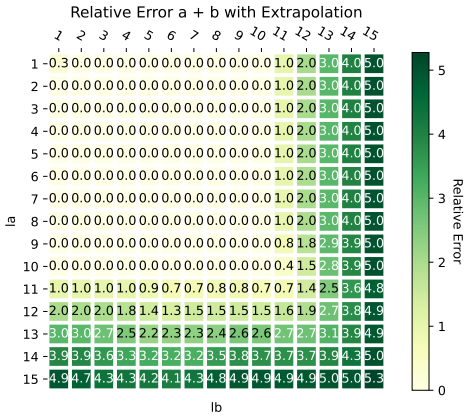

Figure 5: Relative Error Matrix for Extrapolation Behavior Analysis. The results are obtained using a 1.4B model fine-tuned on 2^{19} training samples.

12 sets of case study experiments. Here we list the complete configurations: 1) numeral system: base 10, 10^2, 10^3 2) data scale: 2^{19} 3) model size: 6.9B 4) if from-scratch: True, False 5) operations: addition, multiplication. We discover intriguing extrapolation behavior that could shed light on the mechanisms that the models have learned.

4.3.1 Addition

Note that although addition is an easy task to train, the models have only seen numbers less than 10^{11}. We leverage addition pairs of length la and lb, where at least one of la or lb is greater than 10. To illustrate how the performance of our model decays, we plot the *Relative Error* matrices where one operand length is in $[1, 5]$ and the other in $[11, 15]$ in Figure 5. The results are obtained using a 1.4B fine-tuned model trained on 2^{19} training samples. For each pair of $la \times lb$, we randomly generate 100 samples, which results in a total of 5000 samples. Of all such samples, the extrapolation exact match accuracy is 0.0.

Yet, the models do not collapse completely on out-of-domain length distribution. We conduct case studies in Table 2. Our first discovery is that there is a consistent behavior of *Truncated Addition* across all numeral systems of fine-tuned models. Our second observation is that fine-tuned models are much better at aligning the tokens involved in extrapolated addition, as compared with models trained from-scratch.

Truncated Addition Extrapolates While we are manually inspecting the extrapolation behavior of fine-tuned models, we discover consistently that models would try to perform the addition **truncating** the tokens that exceed training length it has seen. We elaborate on this behavior under two configurations. For illustrative purposes, we add a comma to denote the max training length position the models have seen. First, take for example a model trained on a base 10 system, a fine-tuned model is given input $a = 8318686348, 0$ and $b = 3$, where the token representation of a is 11. Only the first 10 digits of a would participate in addition, yielding a result of $8318686348 + 3 = 8318686351$. A fine-tuned model trained on the base 10^2 system displays very similar results. Given $a = 734766443, 03$ and $b = 3$, the model performs $734766643 + 3 = 734766446$, ignoring two trailing digits, which is equivalent to ignoring the last token under base 10^2. The phenomenon of truncated addition is hardly observed on models trained from-scratch. The main obstacle could arise from the inability to align corresponding tokens with unseen token lengths. For example, a base 10 model trained from-scratch would calculate $2635078980, 7 + 1 = 2635079091$, where the 1 seems to have been added to multiple positions. This could also indicate that fine-tuned models have learned to utilize positional information better.

Base 10 Carry Extrapolates While we attempted to explain extrapolation behavior using *truncated addition*, we noticed some outliers where the answer is only 1 absolute value larger than the truncated addition result. Manual inspection quickly reveals that the models generate carry for out-of-distribution positions. For a base 10 fine-tuned model, $3968299531, 8 + 2 = 3968299534$ $(= 3968299531 + 2 + 1)$, where a carry has been generated because $8 + 2 = 10$. Note that the carry is not generated by aligning the ones digit since $1 + 2 = 3 < 10$, which is an ablation showcasing that calculating carry exhibits extrapolation behavior.

Tokens Generalize, Length Does Not For a base 10^3 system, two kinds of behavior have been observed. Before we describe the behaviors, we restate our experiment settings. Our training distribution only contains numbers that are less than 10^{11} under the base 10 system. This creates two scenarios for extrapolation experiments of a model trained with base 10^3. 1) both operands are less than 10^{13} 2) at least one of the operands is no smaller than 10^{13}. For 1), although the model has not seen any data points within the range of $10^{11} \sim 10^{13} - 1$, the length of both operands does not exceed 4, which

Model	Pattern	a	b	$a + b$	Model Output
SFT-Base 10	w/o carry	8318686348,0	3	8318686348,3	8318686351
	carry success	3968299531,8	2	3968299532,0	3968299534
SFT-Base 10^2	w/o carry	7 34 76 64 43, 03	3	7 34 76 64 43, 06	7 34 76 64 46
	misaligned	1 63 47 53 10, 81	2	1 63 47 53 10, 83	1 63 47 53 12, 83
	carry failure	7 28 37 46 59, 47	94	7 28 37 46 60, 41	7 28 37 47 53
SFT-Base 10^3	w/o carry	2 929 747 175, 022	9	2 929 747 175, 031	2 929 747 184
	misaligned	8 748 392 297, 087	2	8 748 392 297, 089	8 748 392 299, 089
	carry failure	8 172 938 472, 837	494	8 172 938 473, 331	8 172 938 966

Table 2: Representative Cases for Addition Extrapolation. We add a comma to denote the maximum token length of a single number that the model has seen during training.

Model	a	b	$a \times b$	Model Output
SFT-Base 10	9298574444, 7	6	5579144666,82	5579144666,82
SFT-Base 10^2	3 44 97 17 48, 09	8	27 59 77 39 84, 72	27 59 77 39 84, 72
SFT-Base 10^3	18 419 335, 384	4	73 677 341, 536	73 677 341, 536
Scratch-Base 10	44527557923	8	356220463384	358888899984
Scratch-Base 10^2	2 45 57 14 10, 66	8	19 64 57 12 85, 28	20 10 88 12 12, 48
Scratch-Base 10^3	17 709 751, 495	5	88 548 757, 475	87 229 700, 075

Table 3: Representative Cases for Multiplication Extrapolation. We list successful cases for fine-tuned models (i.e. SFT) and showcase the failure of from-scratch models.

has been trained. For 2), the length of at least one operand has not been seen during training at all.

Out of a sample size of 100 for each $la \times lb$ pair, a base 10^3 fine-tuned model could achieve 90% exact match accuracy with $la = 11, lb \in [1, 8]$. However, accuracy quickly drops to 0 if one of the operands has a token length greater than 4 under the base 10^3 system.

4.3.2 Multiplication

Different from addition, there is at least one successful example of extrapolation of operand length for fine-tuned models of all number systems shown in Table 3. Yet, the exact match accuracy on the extrapolation set of models tuned from-scratch is consistently zero. Moreover, a closer look at the generated results showcases that the model is only able to get correct the starting tokens and ending tokens of the answer, with gibberish and repetitive tokens in the middle.

5 Conclusion

In this paper, we study the question of selecting a numeral system for large language models. Specifically, we compare the data-scaling efficiencies of base 10, 100, and 1000 systems. We carefully design scaling experiments with respect to 5 factors: 1) numeral system 2) training data scale 3) model size 4) from-scratch vs fine-tuning 5) addition and multiplication. Through the lens of our selected metrics, we showcase the superiority of the base 10 system. We find the specialized tokenization design of tokenizing number sequences into 1-digit, especially training from scratch. Pythia is pretrained with a base of 1000 numeral system, which weakens the advantage of a base 10 numeral system under the fine-tuning setting. We offer an analysis of the extrapolation behavior of trained models on addition and multiplication. We reveal calculation patterns that successfully extrapolate, such as carry generation in addition and magnitude estimation in multiplication. Our work sheds light on the inner workings that models have learned and hopefully would promote future research on this topic.

Limitations

Scaling behavior analysis requires a huge amount of computational resources. Limited by this factor, we have not performed a thorough grid search for hyperparameters of every setting. It is possible that for every configuration that is of interest, we should use a unique set of hyperparameters to achieve optimal performance. In our experiments, we have witnessed instability issues regarding some data points where the training loss seemingly collapses. It is possible that such issues arose because of suboptimal hyperparameter choices. Research on numerical operation has few potential risks.

References

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint arXiv:2309.16609*.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, et al. 2024. Internlm2 technical report. *arXiv preprint arXiv:2403.17297*.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. 2023. Implicit chain of thought reasoning via knowledge distillation. *ArXiv*, abs/2311.01460.

Leo Gao, John Schulman, and Jacob Hilton. 2023. Scaling laws for reward model overoptimization. In *International Conference on Machine Learning*, pages 10835–10866. PMLR.

Google. 2023. Gemini: A family of highly capable multimodal models. *Preprint*, arXiv:2312.11805.

Danny Hernandez, Jared Kaplan, Tom Henighan, and Sam McCandlish. 2021. Scaling laws for transfer. *arXiv preprint arXiv:2102.01293*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

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

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. 2024. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36.

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

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

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

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

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

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Gemma Team, Thomas Mesnard, Cassidy Hardin, Robert Dadashi, Surya Bhupatiraju, Shreya Pathak, Laurent Sifre, Morgane Rivière, Mihir Sanjay Kale, Juliette Love, et al. 2024. Gemma: Open models based on gemini research and technology. *arXiv preprint arXiv:2403.08295*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023a. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten,

Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models. *Preprint*, arXiv:2307.09288.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Cunxiang Wang, Boyuan Zheng, Yuchen Niu, and Yue Zhang. 2021. Exploring generalization ability of pretrained language models on arithmetic and logical reasoning. In *Natural Language Processing and Chinese Computing: 10th CCF International Conference, NLPCC 2021, Qingdao, China, October 13–17, 2021, Proceedings, Part I 10*, pages 758–769. Springer.

Zheng Yuan, Hongyi Yuan, Chuanqi Tan, Wei Wang, and Songfang Huang. 2023. How well do large language models perform in arithmetic tasks? *Preprint*, arXiv:2304.02015.

Hattie Zhou, Arwen Bradley, Etai Littwin, Noam Razin, Omid Saremi, Josh Susskind, Samy Bengio, and Preetum Nakkiran. 2024. What algorithms can transformers learn? a study in length generalization.

A Metric Matrix for Different Length Pairs

We take a 1.4B model trained from-scratch on addition and multiplication as an exemplar and plot matrices for both Exact Match Accuracy and Normalized Edit Distance with respect to each pair of input lengths in Figure 6 and Figure 7.

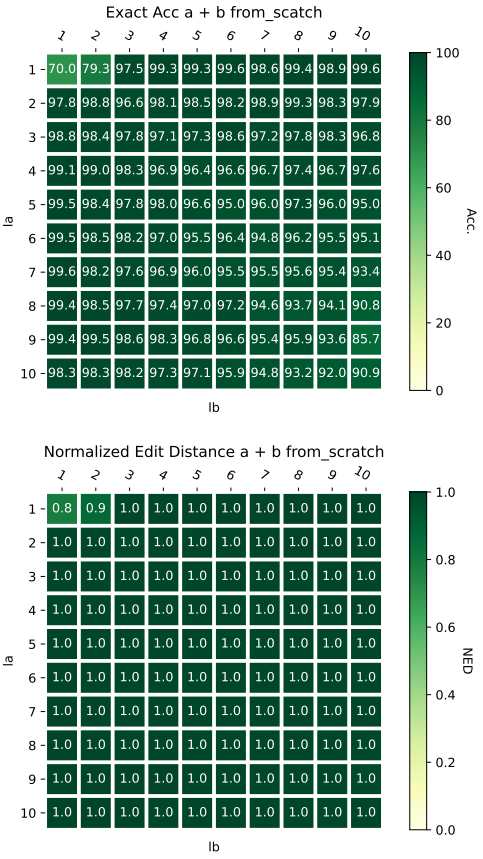

Figure 6: Exact Match Accuracy and Normalized Edit Distance Matrices for Addition Eval Set. The results are obtained using a 1.4B model trained from scratch on 2^{19} samples.

B Overfitting Analysis

Alongside our main results, we also perform ablation studies on overfitting under addition settings, since the accuracy quickly saturates to 100.0%. First of all, we subsample a portion of our training set to forward through the model. We attempt to sample 1000 examples for each $la \times lb$ pair in 10×10. If the total number of training pairs is smaller, we take all training pairs for $la \times lb$.

Generally, for all the metrics of interest, we observe nearly identical performance on our training and evaluation set. Furthermore, since our evaluation set is non-overlapping with the training set,

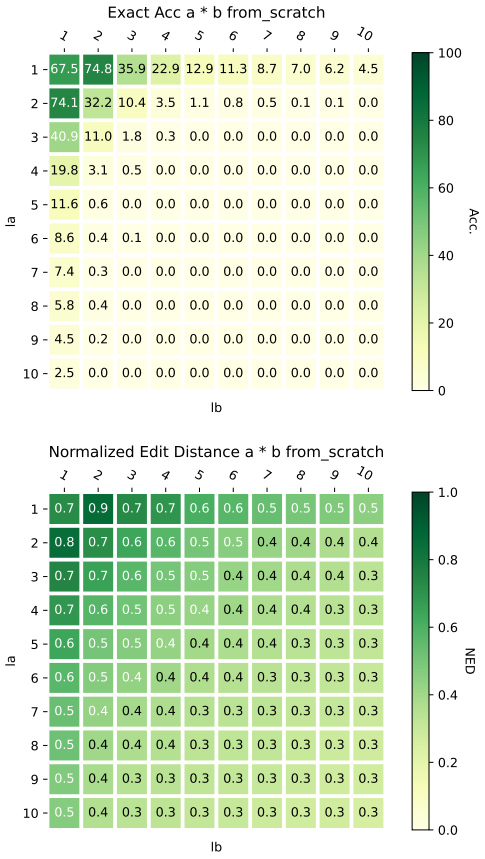

Figure 7: Exact Match Accuracy and Normalized Edit Distance Matrices for Multiplication Eval Set. The results are obtained using a 1.4B model trained from scratch on 2^{19} samples.

it would be safe to conclude that no overfitting phenomenon has been observed.

C Hyper-parameters

We briefly discuss the hyperparameter search process that we have gone through for each configuration. Based on our initial experiments, models exhibit nearly identical behavior in both the training set and the evaluation set. We therefore use training set metrics for hyperparameter selection.

Then, we first fix the learning rate magnitude and sweep for training epochs. We observed that fine-tuned models are insensitive for training epochs, while the model training from-scratch consistently improves with more epochs. We choose epochs where the performances of models begin to plateau. Generally, epoch performance trends only depend on the training setting (i.e. fine-tuning or from-scratch). Fixing the training epoch, we perform a grid search over learning rate magnitudes $\{2e-3, 2e-4, 2e-5, 2e-6, 2e-7\}$ for each configuration. Generally, we found that 70M and 410M models favor a larger learning rate of $2e-4$ while models larger than $1.4B$ use $2e-5$. There is no significant difference between fine-tuning learning rates and from-scratch learning rates. To speed up training, we pack sequences to a maximum length of 2048, therefore fixing the batch size. All experiments are trained using 8xA100 Nvidia GPUs.

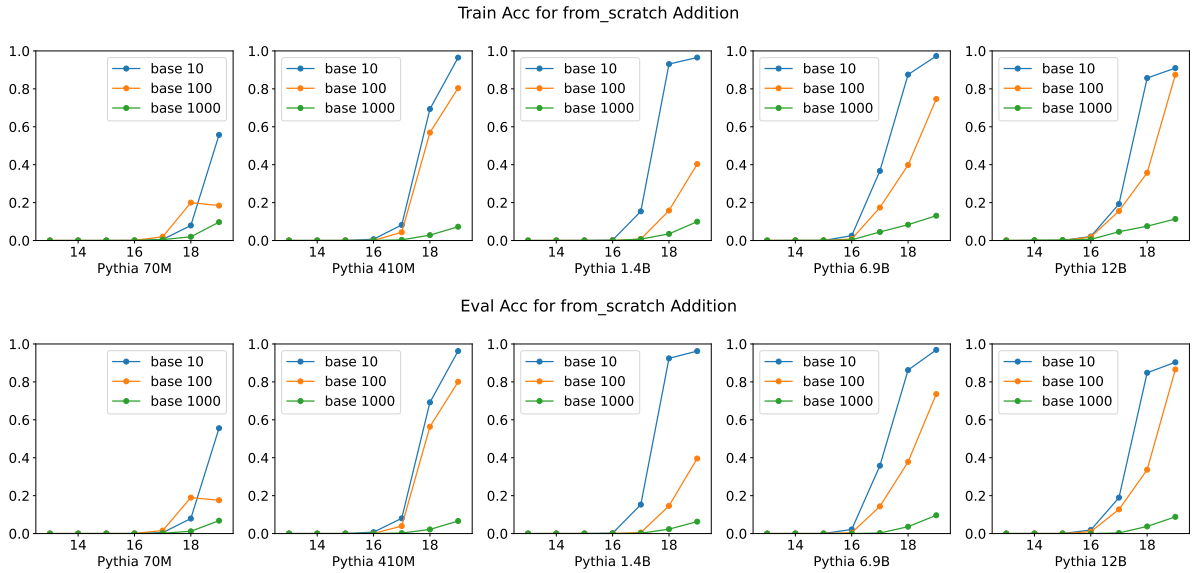

Figure 8: Exact Match Accuracy on the training set versus the eval set for addition operation with different models trained from scratch, on different data scale and numeral systems.