Defining and Testing Essential Number Properties for LMs

Karan Raj Bagri & Dhamale Vivek Shrikrishna & Pawar Rushikesh Gajanansa
Indian Institute of Science
Bengaluru, KA, India
{karanraj,viveksd,rushikeshp}@iisc.ac.in

Abstract

Language Models have achieved impressive feats in Natural Language Processing, excelling at tasks like Text Generation, Machine Translation, Question Answering and Text Summarization. However, NLP systems rarely give special considerations to numbers. They are treated just like any other text tokens, but there is a fundamental difference between words/letters and numbers. This results in a relatively poor numerical ability compared to linguistic proficiency. In this project, we come up with a set of properties that LMs should know about numbers, and we build tests to check how well numeracy is captured by current state-of-the-art models. We experiment with GPT 3.5 Turbo, Gemini 1.0 pro and Llama 2 (quantized) and compare their performace on these tests. The code and additional plots are available at `https://github.com/rushikeshpawar22581/NLP_project_EssentialNumberPropertiesForLMs`

1 Introduction

NLP systems neglects numbers. They are treated just like any other word/character tokens even though they are semantically different from texts, and their operations are also different. This affects the ability of models to handle and interact with numbers on numerical tasks such as comparing, sorting, identifying patterns in sequences. Even the large language models, which perform exceedingly well in linguistic tasks, do not show consistency in performance on basic tasks like the ones mentioned above. Although there have been attempts made to check numeracy, they need improvement. We identify properties of numbers that the language models should learn in order to truly understand numbers and perform well on tasks involving numeracy. We develop tests to check if the models are learning these properties. We evaluate the performance of GPT 3.5 Turbo, Gemini 1.0 pro, Llama 2 (quantized) models on these tests.

2 Related Work

Thawani et al. (2021a) show numeracy improves the word prediction ability of a language model and gains are also observed on text without numbers. This shows the need for LMs to know numbers. Testolin (2024) shows that state-of-the-art architectures often fall short when probed with relatively simple tasks designed to test basic numerical and arithmetic knowledge.

Thawani et al. (2021b) have done a survey of the then existing work on numeracy and broken down notion of numeracy into 7 subtasks(tasks to evaluate numeracy like simple arithmetic, numeration, magnitude comparison, arithmetic word problems, exact facts,measurement estimation, Numerical language modeling). Ren & Du (2020) consider the magnitude aspect of numeracy for models. Magnitude concerns how well the embedding of numbers encode their values information. Peng et al. (2023) They propose five numeracy features that can influence the decoding process. These are numerical type (int,decimal,fraction,percent, etc), magnitude, unit, ratio features and digit feature.
Wallace et al. (2019) found that models fail to extrapolate well to numbers outside its training range. The extrapolation problem persisted for tasks like decoding and addition,

while it had minimal effect on the list-maximum finding problem. Pal & Baral (2021) tested T5 models on simple tasks in interpolation and extrapolation settings and found the extrapolation performance to be much worse.

Zhang et al. (2020) identified contextual information in pretraining and numeracy as two key factors affecting their performance. Their results imply that scale representation in contextual encoders is mediated by transfer of magnitude information from numbers to nouns in pretraining and making this mechanism more robust could improve performance. Muffo et al. (2023) compare the performance of two GPT2 models fine tuned to perform arithmetic operations on same dataset, one does operations on decomposed numbers and another without number decomposition. They observe that models with number decomposition do much better than models without number decomposition in both addition and subtraction.

Nogueira et al. (2021) concluded that surface form of a number has a strong influence on a model's accuracy. Sub-word representations don't work because two very similar numbers may be represented very differently. Many studies have proposed using different numeral embeddings. Kim et al. (2021) tries to solve the problem of extrapolation by introducing new surface form called 'E digit'. Jin et al. (2021) proposed a novel NumGPT model in which they use numeral embeddings for the numbers. Each embedding has 2 parts: one for exponent and another for mantissa. Another study Thawani et al. (2021a) also observed that best performing number encoder is Exp. Spithourakis & Riedel (2018) explores different strategies for modelling numerals, such as memorisation and digit by digit comparison and propose a NN that uses continuous pdf to model numerals from an open vocabulary.Sundararaman et al. (2020) proposed a new methodology to learn word embedding called DICE (determinant and independent of corpus) for embedding such that their cosine similarity reflects the actual distance on the number line. Jiang et al. (2020) propose to represent the embedding of a numeral as a weighted average of a small set of prototype of number embeddings which can handle out of vocabulary problem for numerals.Feng et al. (2022) pretrain BERT in 2 proposed ways that encourages the models to understand magnitude 10^x and value(in front of 10^x) and encode the dependency between a number and its context.

We feel that it is important for models to distinguish between special types of numbers, for example, between primes and composites. Recent work has looked into this problem. Chen et al. (2023) show that CoT prompting may increase or decrease the accuracy in primality testing. Zhang et al. (2023) query the primality of 500 randomly chosen primes between $1,000$ and $20,000$. They find that when a model answered incorrectly, upon being asked for justification through factorization, it provided incorrect factors. We decouple primality detection and factorization. This is elaborated upon in the next section.

We notice that in recent times, most of the work in numeracy has been testing the numerical reasoning abilities of the state-of-the-art models on datasets such as DROP (Dua et al. (2019)), NumGLUE (Mishra et al. (2022)) and MATH (Hendrycks et al. (2021)), Testolin (2024). This type of evaluation may be influenced strongly by context. We explore numeracy at a lower level, by evaluating current state-of-the-art models on more basic tasks. Pal & Baral (2021) have conducted such tests, but their dataset is limited to positive integers, while we have included negative numbers, fractions and decimals too in our tests. We propose new tests for integer addition, list min-max and sort, text-to-number ability, division and primality detection. While conducting tests, we have given one prompt just to specify format of response and not as example to learn. This is done because we want to test numerical understanding of out-of-the-box LMs. Our aim is to check numerical understanding and not the few shot abilities to learn numeracy. The task of studying the impact of fine-tuning and incorporating numeral embeddings is left to the readers.

3 Methodology

We conduct tests to understand capture whether models are able to capture following nuances of numbers.

(a) Magnitude: exact value , order of magnitude.

(b) Influence of types of numbers: integer, fraction, decimal, odd/even, positive/negative.

(c) Scaling: in-domain to out of domain performance shift.

(d) Text form: performance shift caused when the numbers are in textual form.

(e) Identification of special types of numbers: whether the model can distinguish primes from composites.

We define certain tests below. We run them on GPT-3.5 Turbo, Gemini-1.0 Pro and LLama 2 (quantized). We access GPT-3.5 Turbo and Gemini 1.0 Pro through APIs and locally host LLama 2 (quantized). We use 'TheBloke/Llama-2-13B-chat-GGUF' from Hugging Face.

3.1 Investigating integer addition ability:

We attempt to quantify the ability of the models to add 2 integers. We generate a balanced dataset (having 24000 samples consisting of 3-element lists of $[a, b, a + b]$, taking inspiration from the addition task and balanced sampling mentioned by Nogueira et al. (2021). We vary the number of digits from 1 to 60. We want to check how scaling influences the model's performance. To check whether a number being positive or negative influences the accuracy of the predictions, we make sure that we have equal representation of all 4 combinations of signs of a and b . We specify our request and response formats to the model via the prompts. Our requests have the format "What is {a} + {b}?". We batch requests together to comply with the rate limits of API access, by concatenating multiple requests into a single request.

We define two metrics to evaluate the performance of the models: hard accuracy and soft accuracy. Hard accuracy measures the percentage of correctly predicted integer addition outcomes compared to the total number of predictions. Soft accuracy is computed as:

$$\text{Soft Accuracy} = 1 - \frac{\text{Edit Distance}}{\text{Answer Length}} \tag{1}$$

where edit distance is the minimum number of operations (insertions, deletions, or substitutions) required to transform the predicted answer into the correct answer. Soft accuracy provides a measure of performance that considers not only exact matches but also the closeness of predicted answers to the correct ones. Soft accuracy helps us determine how the token error rate is influenced by change in magnitude and sign of the numbers being operated on.

3.2 Investigating the ability to compare numbers:

We design tests to investigate the numerical comparison capabilities of the models. We consider two tasks, finding the maximum and minimum elements in a list and sorting a list, taking inspiration from Wallace et al. (2019) and Pal & Baral (2021).

For list min-max, we generate lists of length 3, lists of length 5 and lists of length 10, containing numbers having upto d digits, where d is varied from 1 to 5. We ensure that half the lists are such that all the numbers are positive and half the lists may contain negative numbers (probability of a number being negative is set to 0.5). This helps us check how much the differing magnitudes and signs affect the comparison capability. Our requests for finding the list minimum have the format "Find the minimum number in the list {list}". We expect responses in the format "Min({list}) = {min num}". Again, we batch multiple requests into one. Similar formats are used for finding the maximum. For list sort, we use the same lists we had generated for list min-max. Our requests are of the format "Sort the numbers in the list {list}". We expect the response to be in the format "{Sorted list}".

Again, we define two metrics to evaluate the performance of the models: hard accuracy and soft accuracy. Hard accuracy measures the percentage of perfectly sorted lists compared to the total number of predictions. Soft accuracy is computed as:

$$\text{Soft Accuracy} = \frac{\text{Length of longest increasing sub sequence}}{\text{Length of the list}} \tag{2}$$

3.3 Investigating understanding of numbers represented as text:

We attempt to investigate the change in the models' performance when numbers are expressed in the form of words. Ideally, when performing simple operations (like addition) on numbers, whether they are represented in the form of digits or text should not matter.

We create a balanced dataset in the same way as we had done for integer addition. We vary the number of digits from 1 to only 7 because larger numbers are rarely present in the form of text. Either one or both numbers in each of the examples are in the text form. We convert the sampled numbers to text using the `num2words` package in python. We specify our request and response formats to the model via the prompts. Our requests have the format "What is {a} + {b}?". We expect responses in the format "{a} + {b} = {c}", where c is the predicted sum. For example, a request may be 'What is 2 + three?'. The model's response should be '2 + three = 5' when we ask for the response to be in the form of digits. When we ask for the response in text form, we expect '2 + three = five'.

3.4 Evaluating Integer-Integer Division Proficiency:

We attempt to quantify the effectiveness with which models divide integers of similar magnitudes. We check how well the accuracy and error scale with magnitude of the numbers as well as whether the sign has any effect.

We generate a balanced dataset (having 3400+ samples consisting of 5-element lists $[a, b, a/b$ (precision=4) $, a/b$ (precision=8) $, a/b$ (precision=12)]). A precision of 4 means that the values are accurate up to 4 digits after the decimal point.

Our requests have the format "What is {a} / {b} correct to $\{n\}$ digits after decimal point?". We expect responses in the format "{a} / {b} = {c}", where c is a/b with n digits of precision. For example, a request may be "What is 1 / 2 correct to 4 digits after decimal point?". The model's response should be "1 / 2 = 0.5000".

3.5 Investigate ability to distinguish primes from composites:

We design tests to see if the models possess the ability to distinguish between primes and composites and how different prompts affect the accuracy. We form a list of all primes from 2 to 100,000 and another list containing all composites in the same range. We sample 5000 primes and 5000 composites from their respective lists to construct the dataset.

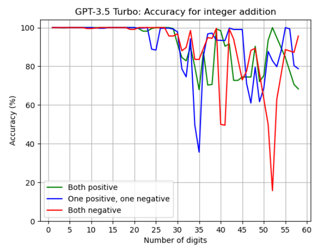
We test with three different prompts. One asks the model whether the number is prime. Another asks if the number is composite. The last one asks for the factors of the number and we check if they imply primality or not.

4 Results

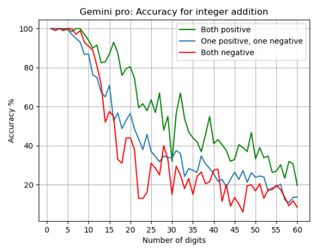
4.1 Investigating integer addition ability:

The models do not always respond in the correct format. Moreover, sometimes the models do not respond to all the requests in a single batched request. We manually figure out all the response formats. We consider only those responses that contained $\{a\}$, $\{b\}$ and $\{c\}$. There happened to be multiple such formats, which were then processed to reconstruct the triplets (a, b, c). The results obtained are shown in figures 1 and 2.

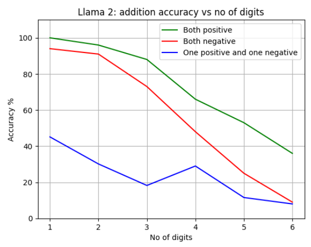
We can see that for all the models, with the increase in number of digits, the accuracy drops. This is consistent with what Nogueira et al. (2021) had observed. As the models encounter smaller numbers more during their training, especially in the context of numerical arithmetic operations, they tend to perform well when adding two numbers comprising a



(a) GPT-3.5 Turbo

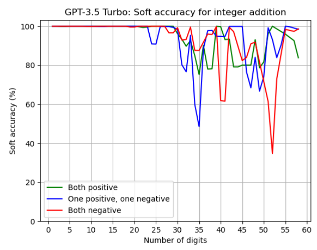


(b) Gemini 1.0 Pro

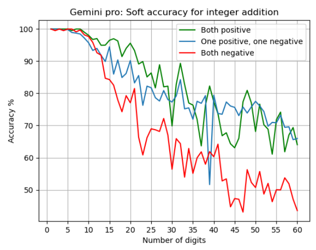


(c) LLama 2

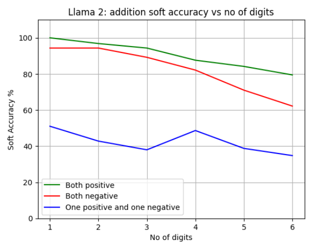
Figure 1: Integer Addition Hard Accuracy



(a) GPT-3.5 Turbo



(b) Gemini 1.0 Pro



(c) LLama 2

Figure 2: Integer Addition Soft Accuracy

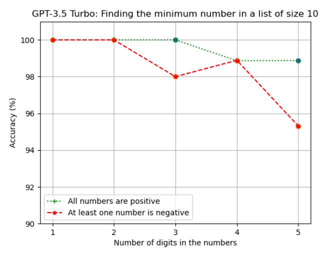
relatively low number of digits. They do not do well when presented with numbers that did not occur or occurred less frequently in its training data Kim et al. (2021). Sometimes, they return garbage values like “$13080003539169249962Andgen9ynomial9257$” or “$Editorial5899179930780095360$”, heavily indicative of its bias towards alphanumeric words in the training corpus. Also, the drop in accuracy is not monotonic, which could be down to the random nature of the models' behaviour.

GPT-3.5 turbo is found to be very accurate when handling numbers till the magnitude of $1e20$, while the Gemini 1.0 is accurate when handling numbers as large as $1e7$. Llama's performance is particularly poor. The accuracy becomes too small at the $1e6$ magnitude mark to even explore numbers beyond that range. Further, we find that the models, especially Gemini, perform slightly better when both the integers are positive. Also, Llama struggles more when the numbers being added have opposite signs.

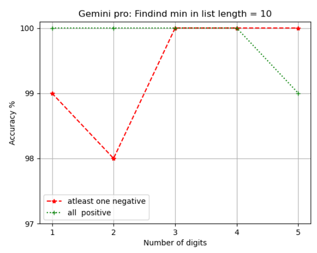
4.2 Investigating the ability to compare numbers:

Figures 3 and 4 show that as the magnitude range (number of digits) of the number increases, the accuracy goes down. Here, even though the numbers encountered are well within the model's training range, it seems to experience some difficulty for the larger numbers. One can imagine the accuracy being much worse when the magnitude range is much bigger than the ones we have tested for. Also, as the size of the list increases, the accuracy falls. The relevant plots for this observation can be found in the repository linked above. One more important observation is that when GPT and Gemini are finding the minimum element of a list containing 10 elements, the accuracy for cases where all the numbers are positive is slightly higher. Conversely, when the model is looking for the maximum element, the presence of negative numbers increases the accuracy. This could suggest that the model understands how to separate positive and negative numbers better than how it understands how to differentiate between numbers only on the basis of magnitude. However, the same cannot be asserted for LLama, which performs worse in the presence of negative numbers irrespective of whether we are finding the maximum or minimum in the list.

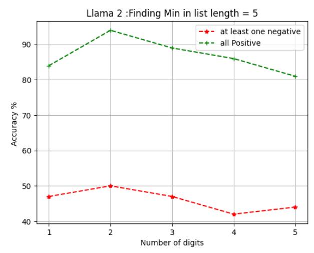
In the case of list-sort, the figures 5 and 6 show that the accuracy clearly drops. As the magnitude range of the number increases, there is a slight drop. But as the size of the list



(a) GPT-3.5 Turbo

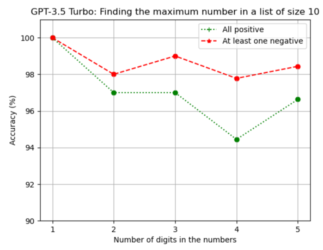


(b) Gemini 1.0 Pro

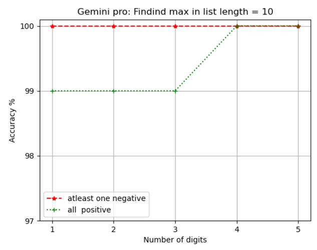


(c) LLama 2

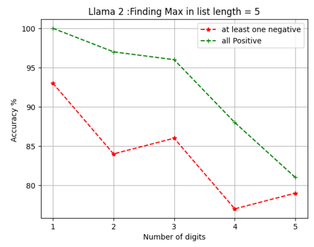
Figure 3: Evaluating list-min accuracy



(a) GPT-3.5 Turbo

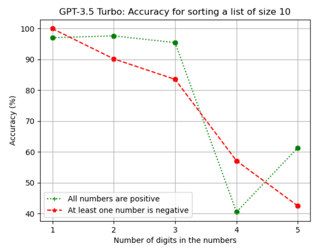


(b) Gemini 1.0 Pro

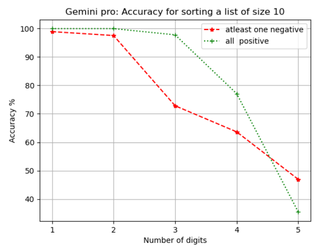


(c) LLama 2

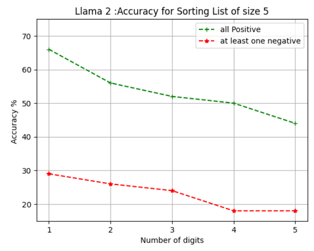
Figure 4: Evaluating list-max accuracy



(a) GPT-3.5 Turbo

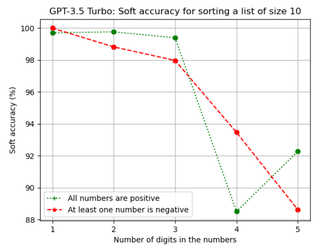


(b) Gemini 1.0 Pro

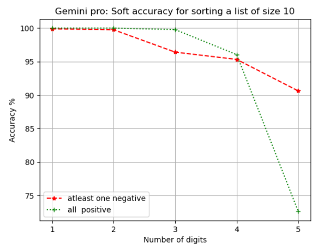


(c) LLama 2

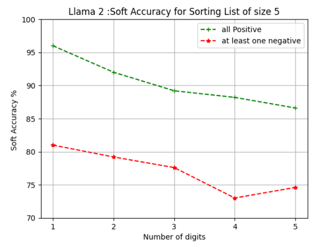
Figure 5: List-sort Hard Accuracy



(a) GPT-3.5 Turbo



(b) Gemini 1.0 Pro



(c) LLama 2

Figure 6: List-sort Soft Accuracy

increases, there is an even more significant drop (graphs for different list sizes in repository). Thus, confirming the fact that the models, at least in zero-shot or one-shot (since we are giving it an example to show the expected response format) settings, are unable to do comparisons between many numbers. GPT and Gemini show comparable performance. However, Llama lags behind. Even for sorting smaller lists, its accuracy is much worse.

Also, the drop in soft accuracy shows that with an increase in magnitude of the integers, the percentage of misplaced numbers in the sorted list increases.

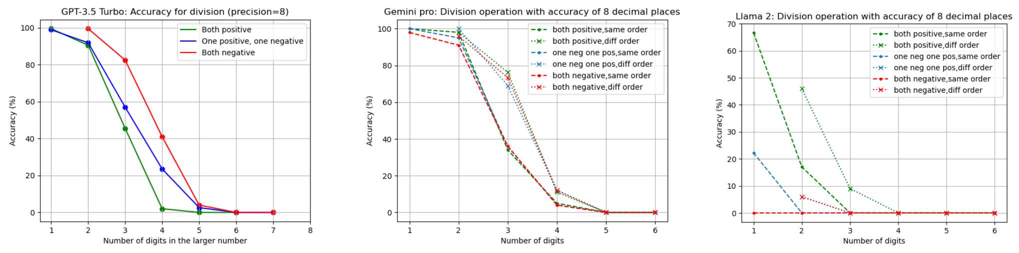
4.3 Investigating understanding of numbers represented as text:

First, we ask the models to convert the integers supplied to them in text form into digit form. We consider integers having magnitude up to $1e7$. We observe that GPT and Gemini do the conversions with over 99% accuracy. Llama's accuracy, however, reduces to 40% when converting 7-digit negative integers.

Next, we perform additions of 2 integers, where one or both may be present in text form. We observe that the accuracy starts falling much quicker than when we add two integers in digit form. For both GPT and Gemini, the fall in accuracy starts at 10^3 to 10^4 range (compared to 10^{20} and 10^7). For Llama, the drop starts from two digit numbers. We suspect that this is because the length of the numbers in text form is much greater than in digit form. Also, whether we ask the model to return the answer in digit form or word form does not make a difference for GPT and Gemini but it does for Llama (when output is in digit form, it is generally more accurate). This is to be expected since digit form to text form conversion is found to be poor for Llama. Plots for this task may be found in the appendix.

4.4 Evaluating integer-integer division proficiency:

We try dividing integers of same order (same number of digits) and of different order. The results obtained when asking for precision of 8 are shown in the figure below. The results for other values of precision may be found in the appendix section.



(a) GPT-3.5 Turbo (b) Gemini 1.0 Pro (c) Llama 2

Figure 7: Accuracy for integer-integer division

For GPT, we observe that the accuracy is higher when both the numerator and denominator are negative, irrespective of the precision we ask for. Also, the relative error is lower for such cases. Further, the relative error (computed as $\frac{\texttt{actual}-\texttt{observed}}{observed}$, is positive when both the numerator and denominator are positive, and negative when both are negative. The relevant plot may be found in the appendix. This suggests that the magnitude of the observed value is generally lower than the magnitude of the true value. Consequently, GPT-3.5 Turbo tends to round down decimal values in terms of magnitude. Also, for GPT, whether the numerator and denominator are of the same sizes or different sizes does not make a difference. For Gemini 1.0 Pro, it does. Llama's performance is comparatively poor. We also observe that when both numbers are negative, for Llama, the accuracy reduces to 0 when we ask for a precision of 8 or more.

4.5 Investigate ability to distinguish primes from composites:

We evaluate the performance of primality detection of each of the models and check how it varies with different prompts. Note that the prompt "What are the factors of {num}?" gives us a list of factors, which we further evaluate to check for primality.

In the case of GPT and Gemini, we observe that when we ask a model whether a certain number is prime, it is more likely to give a response falsely claiming it is a prime. Lower

Prompt	Accuracy	Precision	Recall	F1-Score
Is {num} a prime number?	0.7635	0.7212	0.8592	0.7842
Is {num} a composite number?	0.5653	0.6071	0.3697	0.4595
What are the factors of {num}?	0.6766	0.8502	0.4460	0.5851

Table 1: Testing Primality in GPT-3.5 Turbo

Prompt	Accuracy	Precision	Recall	F1-Score
Is {num} a prime number?	0.6685	0.6920	0.6072	0.6469
Is {num} a composite number?	0.5000	0.5000	0.0002	0.0004
What are the factors of {num}?	0.7728	0.8364	0.6781	0.7490

Table 2: Testing Primality in Gemini 1.0 Pro

Prompt	Accuracy	Precision	Recall	F1-Score
Is {num} a prime number?	0.5017	0.6441	0.0076	0.0150
Is {num} a composite number?	0.5000	NaN	0.0000	NaN
What are the factors of {num}?	0.5426	1.0000	0.0003	0.0006

Table 3: Testing Primality in Llama 2.0

precision values for prompt type a indicates this. Moreover, when we ask a model whether a certain number is composite, it is very likely to falsely call it composite. This behaviour is particularly worse in case of Gemini where almost every response to prompt type b is that the number is composite. Llama classifies almost every integer as composite. We also observe that GPT-3.5 Turbo tends to generate many false factors for the integers, which subsequently leads to them being classified as composites. The low recall value for prompt type c indicates this.

In general, the accuracies are low considering this is a two class classification problem. But this is to be expected since primality testing is a sequential reasoning problem and transformers struggle to generate correct solutions for such problems in a single generation step Merrill & Sabharwal (2023). Moreover, the accuracy decreases slightly for larger integers (see Appendix). This again indicates that scaling is a significant issue for any numerical operations done by LLMs.

5 Conclusion

We saw that GPT-3.5 turbo and Gemini 1.0 Pro performed much better on all the tasks compared to the quantized version of Llama-2. In the integer addition task, as the number of digits increased, the accuracy went downwards. Also, the models became more likely to give garbage responses. All this suggests that the model does not truly understand the addition task. But the signs of the numbers did not influence the addition accuracy. We found that models are not good at dealing with numbers when represented in text form. We also saw that although the list min-max performance reduced with increasing size of list and magnitude of numbers, it was still fairly good; suggesting that the models are good at differentiating one number from the rest. An interesting observation was that the presence of negative numbers actually helped with the list maximum task. This suggests that the model was using the sign information well. For list-sort however, the performance drop was, expectedly, more significant since more comparisons had to be performed. Through the division task, we found that GPT has a tendency to round down (in absolute sense) the fractions. We also found that the prompt given has a significant effect on a model's ability to distinguish between special numbers (like primes and composites). When asked to detect the presence of certain types of numbers (say primes), the models exhibit a bias towards the class that it is asked to detect (more false primes).

Contributions

Pawar Rushikesh Gajanansa:

Literature Review:(9 papers)

Thawani et al. (2021b),Peng et al. (2023),Muffo et al. (2023),Thawani et al. (2021a),Feng et al. (2022),Naik et al. (2019), Jiang et al. (2020),Spithourakis & Riedel (2018),Mishra et al. (2020)

Model Evaluation: All tests and analysis for Gemini 1.0 pro
Dataset creation: Numeration dataset, Different order integer addition dataset

Karan Raj Bagri:

Literature Review:(10 Papers)

Wallace et al. (2019), Jin et al. (2021), Kim et al. (2021), Testolin (2024), Nogueira et al. (2021), Dua et al. (2019), Mishra et al. (2022),Zhang et al. (2023),Chen et al. (2023),Merrill & Sabharwal (2023)

Model Evaluation: All tests and analysis for GPT 3.5 Turbo
Dataset creation: Primality dataset, Integer addition dataset, List dataset for min, max and sort tests

Dhamale Vivek Shrikrishna:

Literature Review:(8 Papers)

Geva et al. (2020),Ren & Du (2020),Al-Negheimish et al. (2021),Andor et al. (2019),Geva et al. (2020),Pal & Baral (2021),Ran et al. (2019),Sundararaman et al. (2020)

Model Evaluation: All tests and analysis for Llama 2 (quantized)
Dataset creation: Division dataset

References

Hadeel Al-Negheimish, Pranava Madhyastha, and Alessandra Russo. Numerical reasoning in machine reading comprehension tasks: are we there yet?, 2021.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. Giving bert a calculator: Finding operations and arguments with reading comprehension, 2019.

Lingjiao Chen, Matei Zaharia, and James Zou. How is chatgpt's behavior changing over time?, 2023.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. Drop: A reading comprehension benchmark requiring discrete reasoning over paragraphs, 2019.

Fuli Feng, Xilin Rui, Wenjie Wang, Yixin Cao, and Tat-Seng Chua. Pre-training and evaluation of numeracy-oriented language model. In Proceedings of the Second ACM International Conference on AI in Finance, ICAIF '21, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391481. doi: 10.1145/3490354.3494412. URL `https://doi.org/10.1145/3490354.3494412`.

Mor Geva, Ankit Gupta, and Jonathan Berant. Injecting numerical reasoning skills into language models, 2020.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset, 2021.

Chengyue Jiang, Zhonglin Nian, Kaihao Guo, Shanbo Chu, Yinggong Zhao, Libin Shen, and Kewei Tu. Learning numeral embeddings, 2020.

Zhihua Jin, Xin Jiang, Xingbo Wang, Qun Liu, Yong Wang, Xiaozhe Ren, and Huamin Qu. Numgpt: Improving numeracy ability of generative pre-trained models, 2021.

Jeonghwan Kim, Giwon Hong, Kyung-min Kim, Junmo Kang, and Sung-Hyon Myaeng. Have you seen that number? investigating extrapolation in question answering models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 7031–7037, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.563. URL `https://aclanthology.org/2021.emnlp-main.563`.

William Merrill and Ashish Sabharwal. The parallelism tradeoff: Limitations of log-precision transformers, 2023.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, and Chitta Baral. Towards question format independent numerical reasoning: A set of prerequisite tasks, 2020.

Swaroop Mishra, Arindam Mitra, Neeraj Varshney, Bhavdeep Sachdeva, Peter Clark, Chitta Baral, and Ashwin Kalyan. Numglue: A suite of fundamental yet challenging mathematical reasoning tasks, 2022.

Matteo Muffo, Aldo Cocco, and Enrico Bertino. Evaluating transformer language models on arithmetic operations using number decomposition, 2023.

Aakanksha Naik, Abhilasha Ravichander, Carolyn Rose, and Eduard Hovy. Exploring numeracy in word embeddings. In Anna Korhonen, David Traum, and Lluís Màrquez (eds.), Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, pp. 3374–3380, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1329. URL `https://aclanthology.org/P19-1329`.

Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks, 2021.

Kuntal Kumar Pal and Chitta Baral. Investigating numeracy learning ability of a text-to-text transfer model, 2021.

Rao Peng, Chuanzhi Yang, Litian Huang, Xiaopan Lyu, Hao Meng, and Xinguo Yu. A numeracy-enhanced decoding fornbsp;solving math word problem. In Natural Language Processing and Chinese Computing: 12th National CCF Conference, NLPCC 2023, Foshan, China, October 12–15, 2023, Proceedings, Part III, pp. 111–122, Berlin, Heidelberg, 2023. Springer-Verlag. ISBN 978-3-031-44698-6. doi: 10.1007/978-3-031-44699-3_11. URL `https://doi.org/10.1007/978-3-031-44699-3_11`.

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. Numnet: Machine reading comprehension with numerical reasoning, 2019.

Yuanhang Ren and Ye Du. Enhancing the numeracy of word embeddings: A linear algebraic perspective. In Xiaodan Zhu, Min Zhang, Yu Hong, and Ruifang He (eds.), Natural Language Processing and Chinese Computing, pp. 170–178, Cham, 2020. Springer International Publishing. ISBN 978-3-030-60450-9.

Georgios Spithourakis and Sebastian Riedel. Numeracy for language models: Evaluating and improving their ability to predict numbers. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). Association for Computational Linguistics, 2018. doi: 10.18653/v1/p18-1196. URL `http://dx.doi.org/10.18653/v1/P18-1196`.

Dhanasekar Sundararaman, Shijing Si, Vivek Subramanian, Guoyin Wang, Devamanyu Hazarika, and Lawrence Carin. Methods for numeracy-preserving word embeddings. In Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (eds.), Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 4742–4753, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-main.384. URL `https://aclanthology.org/2020.emnlp-main.384`.

Alberto Testolin. Can neural networks do arithmetic? a survey on the elementary numerical skills of state-of-the-art deep learning models. Applied Sciences, 14(2):744, January 2024. ISSN 2076-3417. doi: 10.3390/app14020744. URL http://dx.doi.org/10.3390/app14020744.

Avijit Thawani, Jay Pujara, and Filip Ilievski. Numeracy enhances the literacy of language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 6960–6967, Online and Punta Cana, Dominican Republic, November 2021a. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.557. URL https://aclanthology.org/2021.emnlp-main.557.

Avijit Thawani, Jay Pujara, Pedro A. Szekely, and Filip Ilievski. Representing numbers in nlp: a survey and a vision, 2021b.

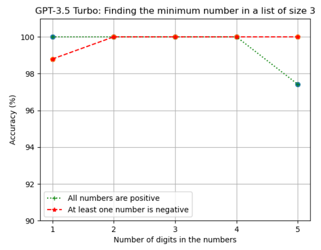
Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings, 2019.

Muru Zhang, Ofir Press, William Merrill, Alisa Liu, and Noah A. Smith. How language model hallucinations can snowball, 2023.

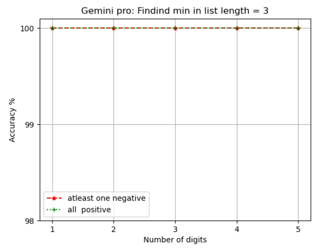
Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. Do language embeddings capture scales?, 2020.

6 Appendix

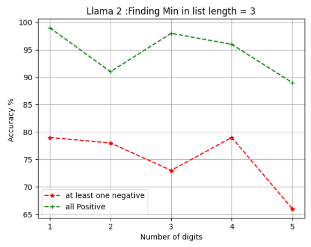
6.1 Investigating the ability to compare numbers:



(a) GPT-3.5 Turbo

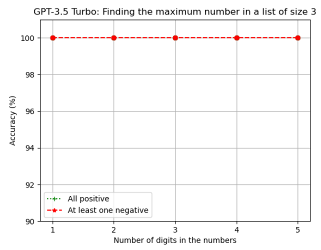


(b) Gemini 1.0 Pro

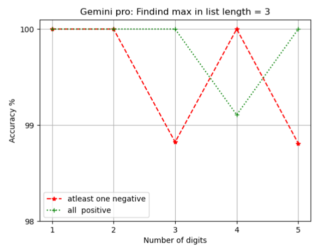


(c) LLama 2

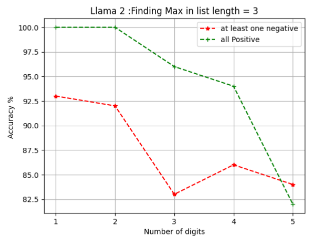
Figure 8: Evaluating list-min accuracy size 3



(a) GPT-3.5 Turbo

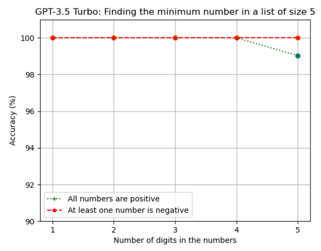


(b) Gemini 1.0 Pro

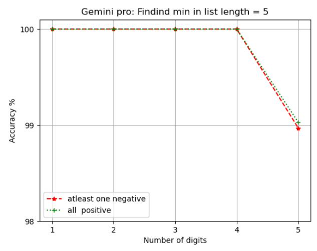


(c) LLama 2

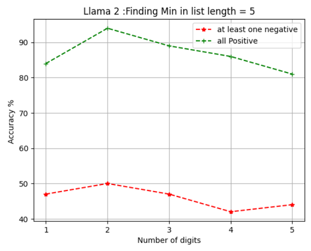
Figure 9: Evaluating list-max accuracy size 3



(a) GPT-3.5 Turbo

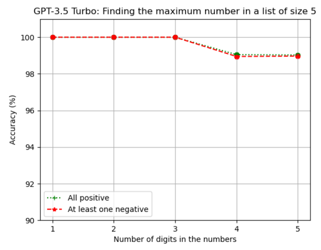


(b) Gemini 1.0 Pro

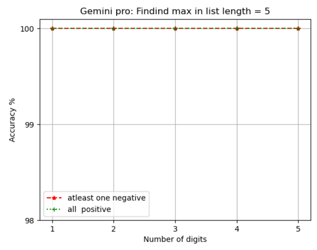


(c) LLama 2

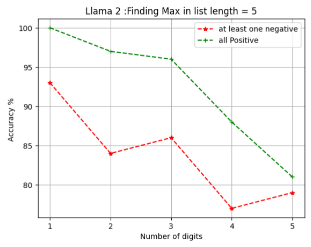
Figure 10: Evaluating list-min accuracy size 5



(a) GPT-3.5 Turbo

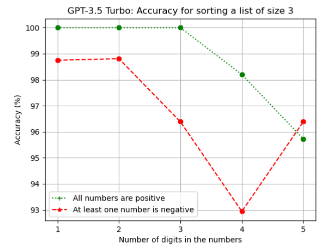


(b) Gemini 1.0 Pro

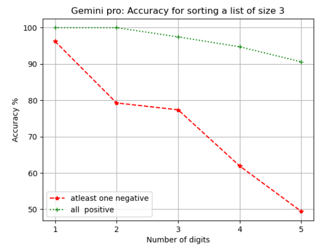


(c) LLama 2

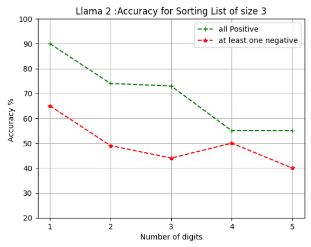
Figure 11: Evaluating list-max accuracy size 5



(a) GPT-3.5 Turbo

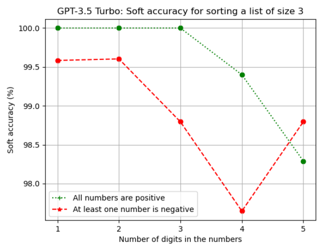


(b) Gemini 1.0 Pro

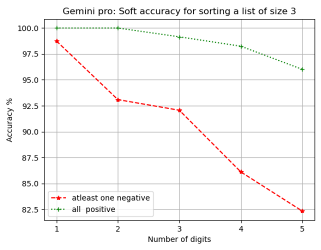


(c) LLama 2

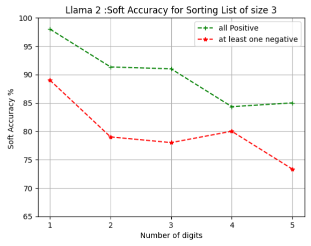
Figure 12: List-Sort Hard Accuracy



(a) GPT-3.5 Turbo

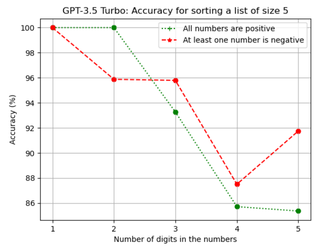


(b) Gemini 1.0 Pro

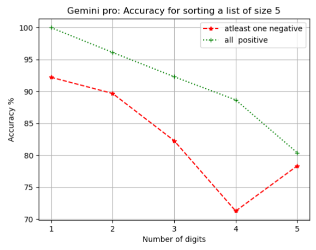


(c) LLama 2

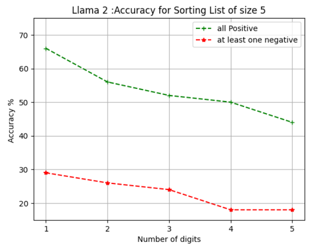
Figure 13: List-Sort Soft Accuracy



(a) GPT-3.5 Turbo

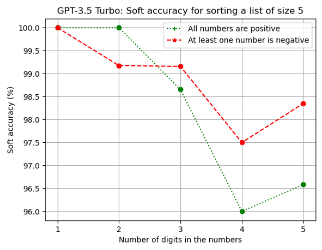


(b) Gemini 1.0 Pro

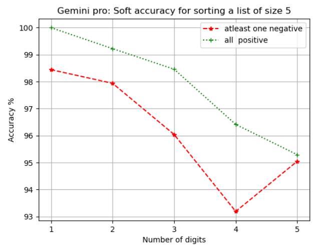


(c) LLama 2

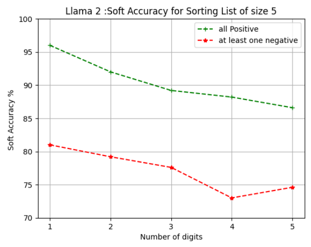
Figure 14: List-Sort Hard Accuracy



(a) GPT-3.5 Turbo



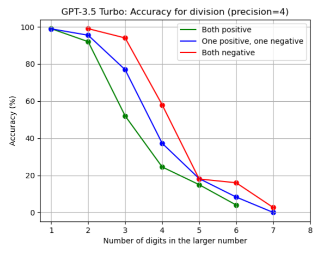
(b) Gemini 1.0 Pro



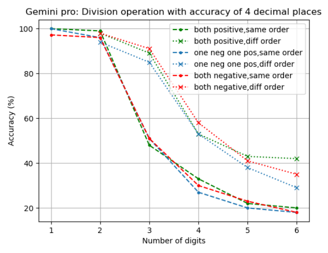
(c) LLama 2

Figure 15: List-Sort Soft Accuracy

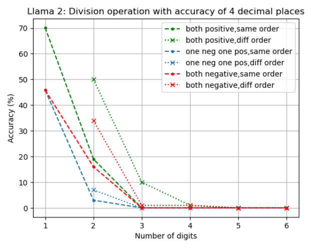
6.2 Evaluating integer-integer division proficiency:



(a) GPT-3.5 Turbo

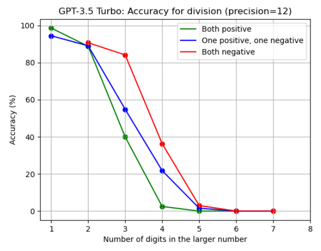


(b) Gemini 1.0 Pro

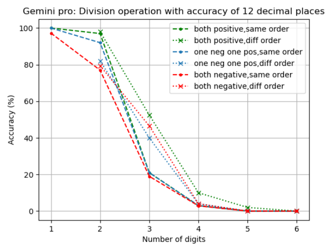


(c) LLama 2

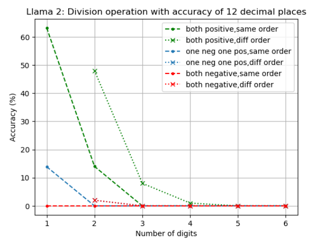
Figure 16: Accuracy for integer-integer division upto 4 digit



(a) GPT-3.5 Turbo

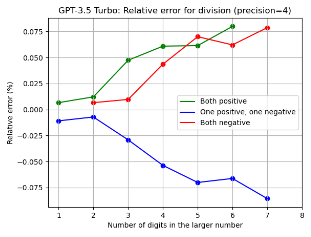


(b) Gemini 1.0 Pro

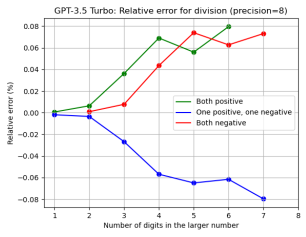


(c) LLama 2

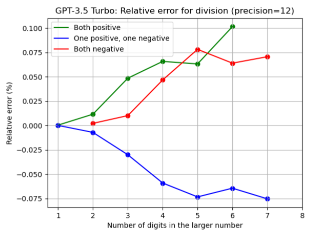
Figure 17: Accuracy for integer-integer division upto 12 digits



(a) 4 digit

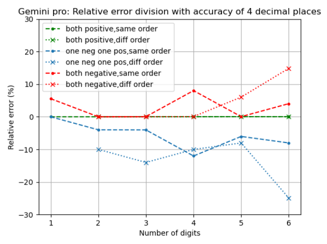


(b) 8 digit

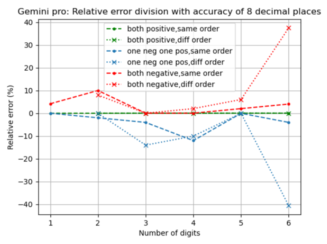


(c) 12 digit

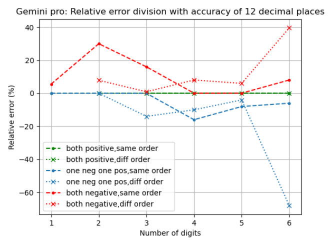
Figure 18: Relative accuracy for integer-integer division for GPT 3.5 turbo



(a) 4 digit



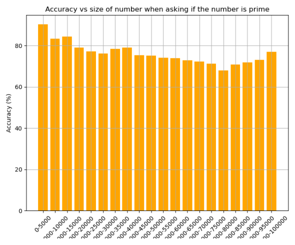
(b) 8 digit

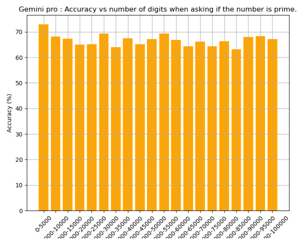


(c) 12 digit

Figure 19: Relative accuracy for integer-integer division fot Gemini 1.0 Pro

6.3 Investigate ability to distinguish primes from composites:


(a) GPT-3.5 Turbo


(b) Gemini 1.0 Pro

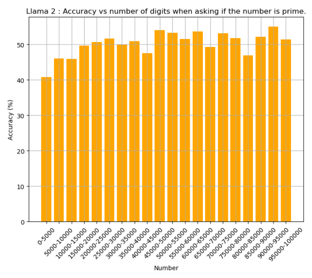
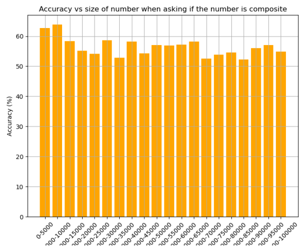
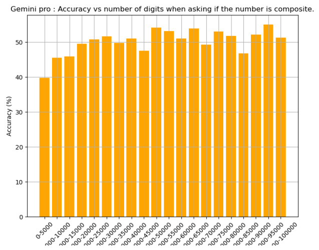

(c) LLama 2

Figure 20: accuracy for is prime test


(a) GPT-3.5 Turbo


(b) Gemini 1.0 Pro

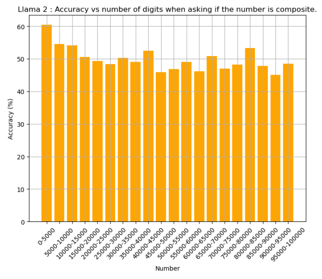
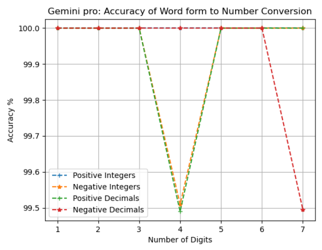
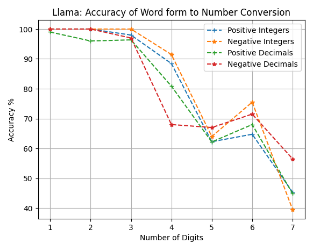

(c) LLama 2

Figure 21: accuracy for is composite test

6.4 Investigating understanding of numbers represented as text:

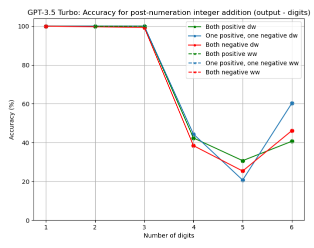


(a) Gemini 1.0 Pro

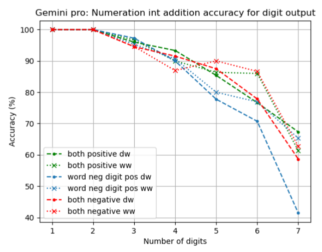


(b) LLama 2

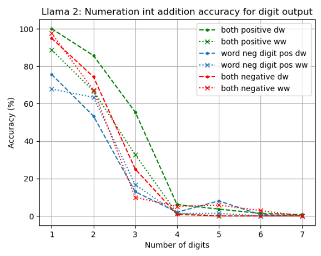
Figure 22: accuracy for word to number test



(a) GPT-3.5 Turbo

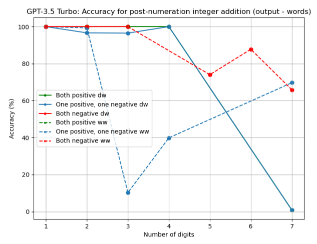


(b) Gemini 1.0 Pro

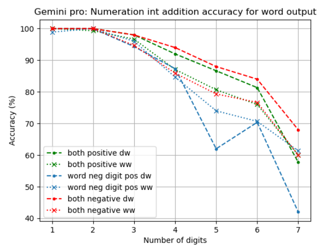


(c) LLama 2

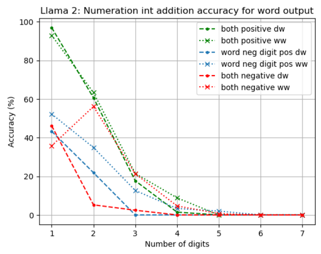
Figure 23: numeration addition accuracy output digit



(a) GPT-3.5 Turbo



(b) Gemini 1.0 Pro



(c) LLama 2

Figure 24: numeration addition accuracy output word