How to Leverage Digit Embeddings to Represent Numbers?

Anonymous ACL submission

Abstract

Apart from performing arithmetic operations, understanding numbers themselves is still a challenge for existing language models. Simple generalisations, such as solving 100+200 instead of 1+2, can substantially affect model performance (Sivakumar and Moosavi, 2023). Among various techniques, character-level embeddings of numbers have emerged as a promising approach to improve number representation. However, this method has limitations as it leaves the task of aggregating digit representations to the model, which lacks direct supervision for this process. In this paper, we explore the use of mathematical priors to compute aggregated digit embeddings and explicitly incorporate these aggregates into transformer models. This can be achieved either by adding a special token to the input embeddings or by introducing an additional loss function to enhance correct predictions. We evaluate the effectiveness of incorporating this explicit aggregation, analysing its strengths and shortcomings, and discuss future directions to better benefit from this approach. Our methods, while simple, are compatible with any pretrained model and require only a few lines of code, which we have made publicly available.[1]

1 Introduction

Numbers play an integral role in language (Thawani et al., 2021), and they are crucial across various domains such as finance (Chen et al., 2018), medicine (Jullien et al., 2023) or even sarcasm (Dubey et al., 2019). Despite, large language models improving their capacity in many tasks, numerical reasoning still poses a challenge (Hong et al., 2024). Recent advancements in enhancing numerical reasoning within language models have predominantly stemmed from using more extensive or higher-quality training datasets (Li et al., 2022a; Yu et al., 2024), scaling up models (Lewkowycz et al., 2022; Kojima et al., 2022), or integrating methods like chain-of-thought reasoning (Wei et al., 2022b; Yue et al., 2024). The effectiveness of such methods is significantly amplified when applied in conjunction with larger model architectures. With smaller models, the improvement shown is often minimal, for example, Wei et al. (2022b) use of chain-of-thought on a 20B parameter model only showed a 2.5% improvement on the MAWPS (Koncel-Kedziorski et al., 2016) dataset whereas it jumps to 14.7% with a 137B parameter model. In addition, many of these solutions are computationally expensive or inaccessible, alternatively we seek a low cost approach that may have minimal impact on small scale models but greater effects on larger models.

One main problem for number understanding is that the widely used tokenisation methods, like Byte-Pair Encoding (BPE) (Sennrich et al., 2016), work well for common words but not for numbers. Specifically, rarer numbers might be broken down into random and meaningless pieces. In light of this, digit tokenisation (Spithourakis and Riedel, 2018) stands out for its simplicity and efficacy at representing numbers. This technique involves breaking down numbers into their individual digits, reducing vocabulary size and ensuring all decimal numbers can be accurately represented enhancing numerical reasoning abilities across various model architectures, tasks, and datasets (Geva et al., 2020; Petrak et al., 2023; Sivakumar and Moosavi, 2023). However, the aggregation of digit embeddings into a complete number representation is implicitly handled by the model, which raises the question: can explicit aggregation using mathematical priors improve numerical understanding? In this paper, we investigate this hypothesis by integrating a mathematically grounded aggregation of digit embeddings explicitly, rather than relying solely on the model's inherent capabilities. We propose a novel approach to number embedding

[1] github repository to be linked here.

that requires no changes to the model's architecture or additional pretraining. Our hypothesis is that an effective aggregation should meet two criteria: (1) it should distinguish between distinct numbers, ensuring unique representations for each value, and (2) the aggregated embedding should reflect natural numerical proximity. We also explore two approaches for this integration: adding a special token before the representation of individual digits to enhance input number representations, and incorporating an additional loss function to improve the representation of output digits.

Our findings show that the integration of explicitly aggregated digit embeddings enhances performance on small-scale models, potentially leading to even greater improvements in larger models. The effectiveness of our integration strategy depends on the size and pretraining of the model used. Our proposed method has promising prospects thus we also enumerate some future directions to further improve number understanding, consequently numerical reasoning.

2 Related Work

Numerical reasoning is the ability to interact with numbers using fundamental mathematical properties and thus model an area of human cognitive thinking (Saxton et al., 2019). Given a maths worded problem, the model needs to interpret the relation between both numbers and the text to then solve the problem by means of arithmetic operations (Ahn et al., 2024). Therefore, an accurate number representation is primordial to both distinguish between different numbers but also predict an accurate answer. The literature focuses on five different areas to better represent numbers.

2.1 Scaling

Increasing the number of parameters of pretrained models has improved their numerical reasoning but it is still nowhere near perfect. For example, Minerva (540B) (Lewkowycz et al., 2022) continued to struggle with higher than seven digit multiplication. Moreover, Frieder et al. (2023) evaluate ChatGPT and GPT4 to conclude that these very large models are inconsistent in their response when answering mathematical questions ranging from arithmetic problems to symbolic maths. This suggest that the models lack fundamental understanding of maths and thus also numbers. One approach to improve number representation is to scale up the vocabulary by having more individual number tokens. For example, GPT3 has unique tokens from the numbers 0-520, whereas GPT4 has them up to 999. Despite general better performance of GPT4, it is not feasible to represent infinitely many numbers in finite memory capacity, making the vocabulary larger would increase the computational costs as well.

2.2 Tokenisation

A more practical approach for representing all numbers is digit tokenisation (Spithourakis and Riedel, 2018; Geva et al., 2020); this separates numbers into a sequence of individual digits. This method improves upon conventional wordpiece tokenisation as shown with GenBERT (Geva et al., 2020) and Mistral-7B (Jiang et al., 2023) by reducing vocabulary size and ensuring precise representation of all numbers. Despite its advantages over conventional tokenisation algorithms, digit tokenisation has limitations. It relies on the model to aggregate digit embeddings into complete number representations, a process for which the model lacks direct supervision. During pretraining, models typically learn to aggregate subword tokens effectively for common words. However, not all numbers are encountered frequently enough during pretraining for the model to learn accurate aggregation. As an example, when the same question is posed with numbers represented differently (once as an integer and once scaled to the thousands), FLAN large with digit tokenisation shows a performance drop of 10% (Sivakumar and Moosavi, 2023). This indicates that the model struggles with numerical consistency and accurate aggregation of digit embeddings.

2.3 Architectural level

Change in model architecture also aids numerical reasoning as shown by NumNET (Ran et al., 2019) and xVAL (Golkar et al., 2024). NumNET extracts the numbers from the input question and passage to create a directed graph with magnitude information about each number present, e.g. which is greater than the others. This information is passed to the model after encoding the input question to supplement it with comparative information about each number so that the model can use this to answer the query. Alternatively, xVAL generates two input encodings, one with the text where numbers are replaced by [NUM], and one with empty space for the text but the actual value of the number in their corresponding positions. From the number preserving encoding, each number is converted to vector embeddings that are composed of themselves at each entry. The product of this vector with the embedding of [NUM] is then injected into the first layer of the transformer for each number in the input sequence. For decoding, a bespoke process is created to extract the predicted number instead of outputting the [NUM] token. Despite the positive contributions of these papers, their methods lack versatility as they are not adaptable off-the-shelf to any pretrained model.

2.4 Loss Functions

Another approach to improve numerical reasoning is for models to intrinsically learn better representation by introducing an inductive bias in the loss function. A simple approach is Wallace et al. (2019)'s use of the mean squared error (MSE) loss across the batch to directly predict floats on a subset of DROP (Dua et al., 2019) which consists of numerical answers. However, this method is limited to datasets that only predict numbers. Contrastive loss is also used to manipulate the representation of numbers, for instance, Petrak et al. (2023) draws nearer the representation generated by BPE and digit tokenisation of numbers through an auxiliary loss when doing extended pretraining to improve arithmetic reasoning in worded problems like DROP but also tables like SciGen (Moosavi et al., 2021). Similarly, Li et al. (2022b) use contrastive learning but on computation trees. They first generate computation trees for the mathematical operations and use contrastive loss to pull nearer the graph representing the same operation, e.g. addition, and push other ones further. This is then integrated in the main loss and improves performance on two maths worded problem datasets, MathQA (Amini et al., 2019) and Math23K (Wang et al., 2017). While these loss functions are adaptable with different models, contrastive training is computationally expensive.

2.5 Input Representation

The most model agnostic method is changing the representation of the numbers in the input text. Wallace et al. (2019) explore worded forms of numbers, but this approach would overly rely on the tokeniser which would split them into subwords. Muffo et al. (2022) decomposes the numbers into place values in reverse order, e.g. 123 = 3 units, 2 tens, 1 hundreds which helps when working with remainders, e.g. when adding. However, this introduces many more tokens which is undesirable as well as either creating new vocabulary for each place value term or the danger of them being split into subword tokens. Zhang et al. (2020) preserves the numerical aspect and converts all numbers into scientific notation, e.g. 314.1 is represented as 3141[EXP]2, improving models' ability to identify the magnitude of a number. Despite providing magnitudinal information, the number before [EXP] still needs to be represented. In fact, all the above strategies require the model to implicitly compute an overall aggregation for the numbers based on their individual components generated by the tokeniser of the model, whether these are digits or subwords. A simple, yet effective method is to introduce pause tokens before predicting the answer (Goyal et al., 2024). This is evaluated by training a 1B parameter transformer model on C4 using [PAUSE] tokens and a 1% improvement is shown on the numerical reasoning dataset, GSM8K (Cobbe et al., 2021). While this method can be used for inference only, they conclude that pretraining is recommended, therefore less applicable to existing models.

Our work is versatile within this line of research. Unlike previous methods that rely on the model to implicitly learn aggregation, we focus on the explicit aggregation of digit embeddings using mathematical priors. This provides direct supervision for the aggregation process, improving the accuracy of number representation. Furthermore, our method ensures that the embedding for a given number aligns with its numerical neighbours, enhancing the model's numerical reasoning capabilities without altering the model architecture or requiring extensive retraining.

3 Aggregation of Digit Embeddings

We explore an approach which is a natural continuation of digit tokenisation as this has demonstrated its efficacy in enhancing numerical reasoning compared to BPE tokenisation. This improvement can be attributed to digit tokenisation's utilisation of pretrained embeddings for individual digits, allowing the model to learn the overall representation through contextualised embeddings. In contrast, BPE may fragment longer and less frequent numbers into random subsequences, resulting in less meaningful aggregations than those achieved through digit tokenisation. However, the implicit aggregation process employed by digit tokenisation remains unclear; specifically, how the model

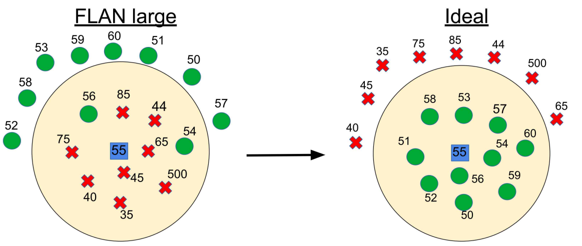

Figure 1: A 2D projection of the neighbourhood of the number token "55" in FLAN large is represented on the left. Ideally, number embeddings should reflect natural numerical proximity. In other words, the embedding for any given number should closely align with those of its immediate numerical neighbours, depicted on the right.

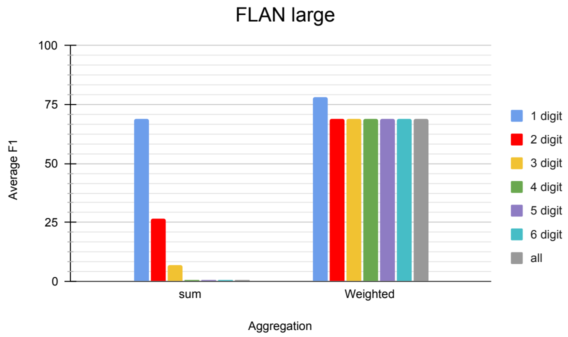

Figure 2: Average F1-score of FLAN large layer 1 numbers using sum and our weighted aggregation function with neighbourhood of 10.

forms the overall aggregation of a number given the embeddings of its individual digits.

In this paper, we investigate a mathematically motivated aggregation that takes into account the relative position of each digit within a number. Our approach generates an overall embedding for the number by considering the positional weight of each individual digit in that number. For example, given "123", the common understanding of numbers as base-10 is "$1 \times 100 + 2 \times 10 + 3 \times 1$", so left most digits are weighted higher as they represent a greater portion of the number.

We design our weighted scheme such that (1) the embeddings of single-digit numbers remain intact, as these embeddings are effectively learned during pretraining, evidenced by the high performance of models on single-digit operations (Sivakumar and Moosavi, 2023), (2) the weights of consecutive place values increase exponentially to reflect base-10, and (3) the weights do not sum to 1, meaning that it is not normalising the sum, allowing for number composed of the same digits, e.g. "111" and "11", to be represented differently. These properties would introduce a bias towards an accurate length of numbers and the correct digits from left to right as the left most digits are amplified, hence preserving natural numerical order.

We propose to calculate the weighted aggregated embedding $\boldsymbol{a}$ with $a_i = \sum w_i \cdot d_i$ for $1 \leq i \leq N$ where N is the number of digits, and the weights w_i are defined as:

$$w_i = 2^{N-i} \times \frac{3(N+1-i)(N+2-i)}{N(N+1)(N+2)}. \quad (1)$$

These weights are designed to satisfy three key properties. **(1) Alignment with single-digit representations:** when $N = 1$, $w_1 = 1$, ensuring compatibility with the model's pretraining on single digits. **(2) Exponential growth:** the exponential component 2^{N-i} mimics the base-10 system, providing an appropriate scale without causing the weights to grow too rapidly. This also ensures that the weights are not normalised. **(3) Regularisation Term:** the fractional component acts as a regularisation term, forming a normalised triangular number sequence. For instance, for a 3-digit number, the sequence is 1,3,6, normalised to 0.1,0.3,0.6. This ensures that the difference between consecutive digit weights increases proportionally, i.e., $w_i - w_{i-1} = w_0 \times i$, replicating the exponential ratio between digit positions in a logarithmic space.

To validate the ability of an aggregated embedding to accurately represent numerical relationships, we use the F1-score to compare natural k-Nearest Neighbours (nkNN) with embedding k-Nearest Neighbours (ekNN). This comparison serves two purposes: firstly, to assess the embeddings' capacity to distinguish between distinct numbers, and secondly, to evaluate how well these embeddings mirror the natural numerical order. By defining nkNN as the set of mathematically adjacent numbers to a given integer n, and ekNN as the set of its closest neighbours in the embedding space, we create a direct measure of the embedding's effectiveness in preserving numerical proximity. The F1-score evaluates the alignment between nkNN and ekNN, penalising both the inclusion of incorrect neighbours and the omission of correct ones. A strong correlation between nkNN and ekNN, as reflected in a high F1-score, indicates that the embeddings faithfully capture the essence of numerical data as illustrated in Figure 1.

We compare our bespoke weighted aggregation function to a more standard aggregation function, sum. For a set of digit embeddings, we apply these functions along each dimension to generate a unique embedding for the number represented by these digits. Figure 2 graphs the F1-score for both functions and different digit length, i.e. 2-digit would be the numbers 10 to 99. Appendix A has results for other aggregation functions: max, min, mean and median; these have the lowest alignment with natural order with an F1-score below 5%. These functions all have a normalising property meaning that the length of the number has no bearing on the aggregated embedding, as the functions only retrieve one entry for each dimension therefore cases like "1111" would be equivalent to both "11" and "1". Contrastingly, sum has better F1-scores for up to 3 digits as it possesses magnitudinal information since all the entries are summed up for each dimension distinguishing, for instance, a 2-digit set from a 3-digit set as it simply adds more numbers. However, it is position agnostic - it assigns equal weight to all the digit irrespective of their relative positions. Therefore, the embeddings generated from permutations of the same digits will always be equivalent, e.g. "85" and "58". Since larger digit numbers have more such permutations, the F1-score reduces as the number of digits increases. Using this metric, the best aggregation is our weighted sum, the average F1-score rounds to 69% for 2 digits onwards suggesting that our weighted sum is closer to the ideal depiction in Figure 1. Undoubtedly, 1-digit F1-score is better as these embeddings are generated from pretraining, but also because the weighted scheme ensures that they are separated from the other number embeddings.

Despite this weighted scheme aligning the number embeddings with their natural order, the weights generated by Equation 1 can become excessively large after a certain point. This behaviour is, however, attenuated by the regularisation term which maintains the high F1-score of 69% for, at least, up to 6-digit long numbers.

4 Integrating Aggregated Embeddings

Given the construction of our mathematically grounded aggregation, we explore two distinct methodologies for enhancing numerical understanding in models, each targeting different aspects of number representation. The first method focuses on enriching the input data by integrating a mathematical aggregation directly into the input embedding as a special token. This approach requires no changes to the model's architecture, making it a flexible solution compatible with various models and suitable for a broad spectrum of tasks.

In contrast, the second approach aims to refine the model's output by improving how numbers are represented in the learned outcomes. This is achieved by incorporating the aggregation in the loss function, encouraging the model to generate number embeddings that align more closely to the correct numerical values. Specifically, this method includes an additional term in the loss calculation, which accounts for the distance between the aggregated embedding of the predicted numbers and that of the true numbers. This targeted intervention is particularly effective in tasks requiring precise numerical predictions, helping the model develop a more nuanced and accurate representation of numbers.

The baseline implementation for both methods is the same as Petrak et al. (2023) with digit tokenisation surrounded by [F] and [/F] tokens to mark the start and end of the number identified using the regular expression "(\d*\.)?\d+".

4.1 Aggregation in Input Embeddings

In our first approach, we enhance the input embedding by incorporating the computed aggregation directly. This is achieved by first digitising numbers and delineating them with special tokens as done by Petrak et al. (2023). Additionally, we introduce a special token, [AGG], positioned as follows where d_i represent the digit tokens: [F] [AGG] [d_1] ... [d_n] [/F]. The embedding for this [AGG] token is initialised with the aggregation of the digit embeddings based on Equation 1.

4.2 Aggregation in Loss Function

Language generation models typically use a cross-entropy loss function ($\mathcal{L}_{CE}$) (Lewis et al., 2020; Raffel et al., 2020). To improve the model's ability to predict numbers accurately, we introduce an auxiliary loss ($\mathcal{L}_{AUX}$) to calculate the mean squared error between the aggregate embedding of the gold and predicted numbers. Understanding and predicting numbers is inherently more complex than predicting a single word or sub-word because they consist of multiple digits, each carrying different significance. For example, in answering the question "Mary's salary is £900 a month, but she pays £579 in rent. How much salary does she have left at the end of each month?", the answers 320, 230, 32, or 456 are all incorrect. However, 320 is more accurate compared to others because its magnitude is closer to the correct answer, 321. Incorporating this new auxiliary loss would help the model predict digits that are closer to the gold answer, enhancing its precision in numerical predictions by recognising the relative significance of each digit within a number.

Given a prediction p and the gold label l, we compute the weighted sum of the digits[2] for both p and l. This process generates two single embedding representations: $W(p)$ for the prediction, and $W(l)$ for the gold label. The distance between these two embeddings is then calculated using the log[3] mean squared error (equivalent to the euclidean distance):

$$\mathcal{L}_{AUX} = \log_2 \left(\|W(p) - W(l)\|_2 \right) \tag{2}$$

The two losses are linearly interpolated by a hyper-parameter, λ:

$$\mathcal{L} = \lambda \times \mathcal{L}_{CE} + (1 - \lambda) \times \mathcal{L}_{AUX} \tag{3}$$

5 Experimental Setup

Both methods are evaluated on two different pre-trained models, BART base (140M) (Lewis et al., 2020) and FLAN base (250M) (Wei et al., 2022a). Additionally, we evaluate on FLAN large (780M) to explore the effect of model size. All of these models are encoder-decoders. BART is pre-trained on five corrupted document tasks from books and Wikipedia data. FLAN is an instruction-finetuned version of T5 (Raffel et al., 2020) which is trained on C4 using transfer learning.

We evaluate our proposed methods on two different test sets: FERMAT (Sivakumar and Moosavi, 2023), and MAWPS (Koncel-Kedziorski et al., 2016). Both FERMAT and MAWPS consist of English maths worded problem that can be tackled by BART and FLAN as shown by Sivakumar and Moosavi (2023) and where the answer is a single number. This enables us to evaluate our method strictly on numerical outputs reducing the interference of other difficulties such as predicting words and units, or extracting spans. FERMAT is a multi-view evaluation set which has different test sets with different number representations while keeping the maths problem fixed. The different test sets distinguish different number types of which we select the ones that separate integers into number lengths, mix integers less than 1000, mix integers greater than 1000, one and two decimal place numbers, and a test set scaled up to more than 4-digit numbers; these allow us to evaluate which number representation the models support better. FERMAT's training set is augmented from templates making it independent to its test sets. MAWPS, on the other hand, has the same domain for both training and testing. It is a widely used dataset to evaluate numerical reasoning, chiefly because it is small and easy to train with small models. We finetune the models on each dataset's respective training data (see Appendix B) using the hyperparameters described in Appendix C.

Accuracy is the general metric used to evaluate these datasets, however, since it is sometimes too stringent and neglects to reflect some improvements of the model, we also use a variation of edit distance (Levenshtein, 1966) as a supplementary metric. Edit distance helps see improvement in the predictions despite being incorrect; it calculates how many insertions, deletions or substitutions is required for the prediction to be transformed into the gold label number on a string level. In this paper, we will use Character Error Rate (CER) which is a character level (digit level) edit distance as a percentage over the string length of the target. The lower the CER, the closer the prediction is to the gold label.

6 Impact of Integrating Aggregations

Table 1 presents the results of our exploration into the effects of integrating mathematical aggregation into the three models across two distinct settings. The bold values indicate the stronger improvement between the two incorporation strategies. For the majority of the test splits, the strongest performance of the examined models is observed when the aggregation is incorporated into the auxiliary loss. This suggests that incorporating aggregation at the output level is more effective than incorporating it in the input embedding. However, this may be due to the fact that adding a new token in the input might require more than just fine-tuning, such as an extended pretraining phase. This aligns with the observations made by Goyal et al. (2024), who found that the addition of the pause token only became effective from pretraining.

FLAN large, on the other hand, has a more bal-

[2] Should the answers not be numerical, the model is penalise by arbitrarily setting $\mathcal{L}_{AUX}$ to 20.

[3] Log base 2 is used to regularise the auxiliary loss.

Incorporating Weights (Accuracy %)		MAWPS	FERMAT													
			Original	Commuted	Integers 0 to 1000	2-digit integers	3-digit integers	4-digit integers	1000+	1000+ same	1dp random	2dp random	a+b	a-b	a*b	a/b
BART base (140M)	Digits	19.20	16.65	8.73	10.26	13.41	10.89	7.74	5.58	10.89	17.82	8.37	40.91	10.62	9.56	11.76
	[AGG] + Digits	**+2.00**	+0.63	+1.53	-1.17	-0.90	-2.16	-0.27	+0.09	**+0.09**	**+1.08**	-0.27	-3.90	-0.74	+1.77	0.00
	Digits + Aux Loss	+1.40	**+1.89**	**+1.80**	**+0.54**	**+0.81**	0.00	**+0.81**	**+1.17**	-1.26	+0.18	**+0.63**	**+2.01**	**+0.19**	**+4.25**	-1.27
FLAN base (250M)	Digits	23.00	28.35	17.82	17.10	22.86	17.37	13.77	10.35	18.72	25.83	18.45	63.38	19.57	12.92	11.27
	[AGG] + Digits	+0.80	**+2.79**	+0.27	+2.52	+0.81	**+1.80**	**+2.79**	**+1.80**	+0.90	+0.45	-0.09	**+4.48**	+3.21	-0.27	+1.08
	Digits + Aux Loss	**+1.80**	+2.25	**+0.36**	**+3.15**	**+2.16**	+1.71	**+2.79**	+0.81	**+3.87**	**+1.89**	-0.18	+3.90	**+5.80**	**+0.27**	**+1.57**
FLAN large (780M)	Digits	28.80	42.39	21.06	25.65	31.32	24.30	21.87	16.47	23.31	36.36	25.83	63.12	39.88	18.23	18.14
	[AGG] + Digits	**+1.20**	+0.45	**+0.45**	+0.81	+2.07	**+2.79**	**+0.99**	+1.35	**+2.88**	+0.27	+0.54	**+6.17**	**+3.83**	**+0.53**	**+1.47**
	Digits + Aux Loss	+1.00	**+0.99**	-0.18	**+1.62**	**+2.88**	**+2.79**	+0.72	**+1.53**	+1.26	**+1.26**	**+0.63**	-0.39	+1.79	+0.18	-1.08

Table 1: Results change from baseline after including aggregate embeddings in input embedding ([AGG] + Digits) and auxiliary loss (Digits + Aux Loss) for BART base, FLAN base and FLAN large. Darker shades of green and red indicate an absolute change greater than 1%.

anced performance but an overall higher improvement when the aggregation is incorporate in the input as shown particularly from all the green cells in the row [AGG] + Digits. Therefore, a certain model size may be required to learn a new token and leverage the information it provides. This reinforces that an aggregated embedding provides useful signal to improve number understanding but how it is integrated is also crucial.

When focusing on smaller integers (columns "Integers 0 to 1000" to "4-digit integers"), incorporating the weighted embedding in the auxiliary loss consistently yields better performance, with all cells being green and showing the highest scores. For smaller integers, models likely already possess a strong implicit representation, making the explicit [AGG] token less impactful. However, at the decoding stage, the auxiliary loss enhances precision by penalising incorrect predictions.

For the 1000+ columns, using accuracy, the pattern is not evident, however, from Appendix D, using the auxiliary loss clearly reduces the CER more than explicitly using the aggregation in the input. The auxiliary loss encourages the model to predict the correct answer as the CER is lower. However, since the weights assigned to each digit position is lower as it gets closer to the units, the auxiliary accounts less for it, reducing precision. As a consequence, despite the CER reducing, since the entire number is not predicted correctly, improvement fails to be reflect in the accuracy.

7 Analysis of Aggregation Embedding in the Input

The first integration method relies on prepending the aggregated embedding token, [AGG], before the digits. The position of the token is before what it represents, similar in nature to BERT's (Devlin et al., 2019) [CLS] token, which is an aggregation token of the entire input. However, Goyal et al. (2024) use a [PAUSE] token posteriori to the digit tokens to act as processing time after concluding that prepending it had less impact. Consequently, we also evaluate our proposed method by appending the aggregation token, i.e. Digits + [AGG]. Table 2 clearly shows that this configuration for both base models underperforms compared to [AGG] + Digit as rows have more red entries. In fact, it performs worse than the baseline with only digit tokenisation. For FLAN large, the results between [AGG] prepended and appended are closer to one another, but prepended, the impact is positive for each test set and on average better by 1% than [AGG] used posteriori. Seeing the token before the digits might provide magnitude information of the overall number which would indicate the importance of each digit to come, whereas having it after might interfere with the representation that the model has already started to create implicitly from seeing the digits first.

Additionally, we test the impact of providing the aggregated token by replacing it with a randomly initialised [PAUSE] token akin to Goyal et al. (2024). From Table 2, we observe that for BART, nor [AGG], nor [PAUSE] have a great positive impact on the performance. This confirms that BART struggles to learn new tokens from fine-tuning alone. The FLAN models are more adaptable to the new tokens as seen by the greener rows. However, the overwhelming bold entries with the [PAUSE] token indicate that both FLAN base and large perform better with a [PAUSE] token acting

Aggregated Embedding (Accuracy %)		MAWPS	FERMAT													
			Original	Commuted	Integers 0 to 1000	2-digit integers	3-digit integers	4-digit integers	1000+	1000+ same	1dp random	2dp random	a+b	a-b	a*b	a/b
BART base (140M)	Digits	19.20	16.65	8.73	10.26	13.41	10.89	7.74	5.58	10.89	17.82	8.37	40.91	10.62	9.56	11.76
	Digits + [AGG]	-1.40	-14.76	-7.74	-8.82	-10.98	-8.73	-6.75	-5.58	-10.35	-14.76	-7.83	-36.82	-9.38	-8.94	-9.51
	[AGG] + Digits	**+2.00**	**+0.63**	**+1.53**	-1.17	-0.90	-2.16	0.27	**+0.09**	**+0.09**	**+1.08**	0.27	3.90	0.74	**+1.77**	0.00
	[PAUSE] + Digits	-1.40	+0.18	-0.45	-0.18	-0.63	-0.90	-0.36	-0.27	-3.87	-0.90	0.00	-8.51	-0.31	+1.68	-2.06
FLAN base (250M)	Digits	23.00	28.35	17.82	17.10	22.86	17.37	13.77	10.35	18.72	25.83	18.45	63.38	19.57	12.92	11.27
	Digits + [AGG]	**+1.80**	-1.53	-2.07	+0.99	-1.89	-0.36	+0.63	+1.35	-0.63	-1.98	-0.99	+0.45	+3.89	-2.39	-0.10
	[AGG] + Digits	+0.80	**+2.79**	**+0.27**	**+2.52**	+0.81	**+1.80**	**+2.79**	+1.80	+0.90	+0.45	-0.09	**+4.48**	+3.21	-0.27	+1.08
	[PAUSE] + Digits	+1.00	+2.07	-0.54	+1.98	**+1.44**	**+1.80**	+2.61	**+2.52**	**+2.16**	**+2.61**	**+1.71**	+3.18	**+5.99**	1.95	**+3.43**
FLAN large (780M)	Digits	28.80	42.39	21.06	25.65	31.32	24.30	21.87	16.47	23.31	36.36	25.83	63.12	39.88	18.23	18.14
	Digits + [AGG]	-2.80	-2.16	**+1.35**	**+1.89**	+1.08	+1.44	+1.62	+2.16	**+5.40**	-1.17	+0.54	**+8.57**	-8.15	-0.97	+1.18
	[AGG] + Digits	**+1.20**	**+0.45**	+0.45	+0.81	+2.07	+2.79	+0.99	+1.35	+2.88	+0.27	+0.54	+6.17	**+3.83**	**+0.53**	+1.47
	[PAUSE] + Digits	-1.40	-0.45	-0.45	**+1.89**	**+3.69**	**+2.88**	**+3.06**	**+2.25**	+5.04	**+1.17**	**+2.61**	+6.17	+1.17	-1.77	**+3.53**

Table 2: Comparing the aggregated embedding at the input level with a pause token and positioning the token after the digits. Darker shades of green and red indicate an absolute change greater than 1%.

as a blank space for the model to process the information. It may also be that the model uses this token to create an implicit representation of the number. Nevertheless, the average improvement between the [PAUSE] and [AGG] differs by less than 0.5% implying that a different aggregation function or a full hyperparameter search could reverse the trend.

8 Future Work

Our proposed aggregation strategy has shown encouraging steps towards better number representation. However, as with observation made in previous work, the effect of new strategies report minimal improvement on smaller models but greater impact on larger models (Cobbe et al., 2021; Wei et al., 2022b). Therefore, an evaluation of our proposed method on larger scale models would verify the scalability of this approach.

The weighting scheme, presented in Equation 1, offers a straightforward method for aggregating digit embeddings. However, as numbers increase in length, their aggregated embeddings tend to drift away from the original numerical embedding space. This divergence could be addressed by enabling the model to adapt to this new embedding space by exploring extended pretraining, or alternative weighting schemes that remain closer to the numerical subspace while satisfying the criteria outlined in Section 3.

Our auxiliary loss, grounded in Mean Squared Error, shows promising results for penalising the model's erroneous predictions and nudging it towards more accurate outcomes. Given that the values resulting from standard cross-entropy and the MSE of the aggregated embeddings may span vastly different value ranges, crafting a loss function that aligns more closely in magnitude with the output of cross-entropy could mitigate the risk of exerting excessive regularisation pressure.

9 Conclusion

Improving numerical reasoning is a challenging task, increasing model sizes or focusing on data augmentation helps but at the cost of a substantial additional training time or computations. Digit tokenisation has been a pioneering work in improving how models encode and decode numbers, however the aggregation of the digit is done implicitly. We advance this idea by explicitly providing an aggregated number embedding that is more mathematically sound. These embeddings are generated as weighted sums of the digit embeddings by accounting for the digits relative position in the number. We then incorporate them in two model agnostic forms: in the input level as an additional token, and in an auxiliary MSE loss. Our promising results demonstrate that, as a proof-of-concept, even a straightforward aggregation with simple incorporation techniques can positively impact number understanding. Therefore, testing it at larger scale, developing sophisticated aggregation functions, and refining the integration of the auxiliary loss presents valuable avenues for future research.

10 Limitations

Some of the limitations of this work is discussed in the Future Work section. However, we give detail of more limitations relating to the size of the models used, and the compatibility and growth of our proposed weighted aggregation function.

Due to financial and resource constraints the hypothesis that the methods for incorporating the aggregated embedding in larger architectures would lead to greater performance based on the improvement observed on smaller model is not verified.

In addition, while the weighted scheme is designed using mathematical priors, it is specifically created for integers, therefore it may not be compatible with decimals or alternative representation of numbers such as 01 for 1. Nonetheless, from Table 5, we note that CER reduces for both 1dp and 2dp therefore our aggregated embedding method has promising scope for all numbers. Lastly, the weights function described in Equation 1 does not converge, therefore for a sufficiently large number of digit it would grow beyond the accuracy provided by the model. However, we explain in Section 3 with the aid of Figure 2 that, for up to 6-digits, the weighted scheme functions well with no signs of deterioration. Moreover, in natural text, very large numbers tend to be shorten using a more appropriate unit, for example, the world population of 8114693010 is more often expressed as 8 billion reducing the numbers of digits needed considerably.

References

Janice Ahn, Rishu Verma, Renze Lou, Di Liu, Rui Zhang, and Wenpeng Yin. 2024. Large language models for mathematical reasoning: Progresses and challenges. *arXiv preprint arXiv:2402.00157*.

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. 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 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Chung-Chi Chen, Hen-Hsen Huang, Yow-Ting Shiue, and Hsin-Hsi Chen. 2018. Numeral understanding in financial tweets for fine-grained crowd-based forecasting. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*, pages 136–143.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

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, Minneapolis, Minnesota. Association for Computational Linguistics.

Dheeru Dua, Yizhong Wang, Pradeep Dasigi, Gabriel Stanovsky, Sameer Singh, and Matt Gardner. 2019. DROP: A reading comprehension benchmark requiring discrete reasoning over paragraphs. 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 2368–2378, Minneapolis, Minnesota. Association for Computational Linguistics.

Abhijeet Dubey, Lakshya Kumar, Arpan Somani, Aditya Joshi, and Pushpak Bhattacharyya. 2019. "when numbers matter!": Detecting sarcasm in numerical portions of text. In *Proceedings of the Tenth Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis*, pages 72–80, Minneapolis, USA. Association for Computational Linguistics.

Simon Frieder, Luca Pinchetti, Alexis Chevalier, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Christian Petersen, and Julius Berner. 2023. Mathematical capabilities of chatGPT. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 946–958, Online. Association for Computational Linguistics.

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.

Sachin Goyal, Ziwei Ji, Ankit Singh Rawat, Aditya Krishna Menon, Sanjiv Kumar, and Vaishnavh Nagarajan. 2024. Think before you speak: Training language models with pause tokens. In *The Twelfth International Conference on Learning Representations*.

Pengfei Hong, Deepanway Ghosal, Navonil Majumder, Somak Aditya, Rada Mihalcea, and Soujanya Poria. 2024. Stuck in the quicksand of numeracy, far from agi summit: Evaluating llms' mathematical competency through ontology-guided perturbations. *arXiv preprint arXiv:2401.09395*.

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.

Maël Jullien, Marco Valentino, Hannah Frost, Paul O'regan, Donal Landers, and André Freitas. 2023. SemEval-2023 task 7: Multi-evidence natural language inference for clinical trial data. In *Proceedings of the The 17th International Workshop on Semantic Evaluation (SemEval-2023)*, pages 2216–2226, Toronto, Canada. Association for Computational Linguistics.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. In *ICML 2022 Workshop on Knowledge Retrieval and Language Models*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

Vladimir I. Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online. Association for Computational Linguistics.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. 2022. Solving quantitative reasoning problems with language models. In *Advances in Neural Information Processing Systems*.

Ailisi Li, Yanghua Xiao, Jiaqing Liang, and Yunwen Chen. 2022a. Semantic-based data augmentation for math word problems. In *Database Systems for Advanced Applications: 27th International Conference, DASFAA 2022, Virtual Event, April 11–14, 2022, Proceedings, Part III*, page 36–51, Berlin, Heidelberg. Springer-Verlag.

Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. 2022b. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 2486–2496, Dublin, Ireland. Association for Computational Linguistics.

Nafise Sadat Moosavi, Andreas Rücklé, Dan Roth, and Iryna Gurevych. 2021. Scigen: a dataset for reasoning-aware text generation from scientific tables. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.

Matteo Muffo, Aldo Cocco, and Enrico Bertino. 2022. Evaluating transformer language models on arithmetic operations using number decomposition. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 291–297, Marseille, France. European Language Resources Association.

Dominic Petrak, Nafise Sadat Moosavi, and Iryna Gurevych. 2023. Arithmetic-based pretraining improving numeracy of pretrained language models. In *Proceedings of the 12th Joint Conference on Lexical and Computational Semantics (*SEM 2023)*, pages 477–493, Toronto, Canada. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Qiu Ran, Yankai Lin, Peng Li, Jie Zhou, and Zhiyuan Liu. 2019. NumNet: Machine reading comprehension with numerical reasoning. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2474–2484, Hong Kong, China. Association for Computational Linguistics.

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Jasivan Sivakumar and Nafise Sadat Moosavi. 2023. FERMAT: An alternative to accuracy for numerical reasoning. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15026–15043, Toronto, Canada. Association for Computational Linguistics.

Georgios Spithourakis and Sebastian Riedel. 2018. 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)*, pages 2104–2115, Melbourne, Australia. Association for Computational Linguistics.

Avijit Thawani, Jay Pujara, Filip Ilievski, and Pedro Szekely. 2021. Representing numbers in NLP: a survey and a vision. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 644–656, Online. Association for Computational Linguistics.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5307–5315, Hong Kong, China. Association for Computational Linguistics.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Jason Wei, Maarten Bosma, Vincent Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V Le. 2022a. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*.

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*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. Metamath: Bootstrap your own mathematical questions for large language models. In *The Twelfth International Conference on Learning Representations*.

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*.

Xikun Zhang, Deepak Ramachandran, Ian Tenney, Yanai Elazar, and Dan Roth. 2020. Do language embeddings capture scales? In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 4889–4896, Online. Association for Computational Linguistics.

Appendix

A Aggregation functions

Figure 3 shows that F1-score for numbers with up to 6-digits across six different aggregation functions. The F1-score for max, min, mean and median are all below 5%.

B Datasets

The datasets' split is given in Table 3. MAWPS is a dataset generated by combining different ones ranging from addition and subtraction to simultaneous equations. The collation of questions is split to create the train, development and test set. FERMAT is a large dataset which has a training and development set automatically generated from 100 templates using different numbers from the following four categories: small integers (less than 1000), large integers (between 1000 and 100000), 1 decimal place and 2 decimal place numbers. The test set is independently generated from two maths worded problem datasets, and then augmented to create 21 test sets of which we use 11.

Datasets	Train	Dev	Test
MAWPS	1500	373	500
FERMAT	200000	1000	1111x11

Table 3: Train, development, and test splits of MAWPS and FERMAT.

C Hyperparameters

All experiments were conducted using an Nvidia Tesla A100 with 80G and with a weight decay of 0.005, warm-up of 100, float32 and 3 generation beams, max input length = 128, max target length=16, and seed=42. Due to limited computational resources, a full grid search of hyperparameter was impossible, however, we do a lambda search in the range 0.4 to 0.8 in 0.05 increments. Specific hyperparameters as well as computation time for dataset and model combinations can be found in Table 4.

D Character Error Rate (CER) Results

Table 5 presents the character error rate (CER) for incorporating the weighted aggregation as an input token and in the auxiliary loss, for all three models.

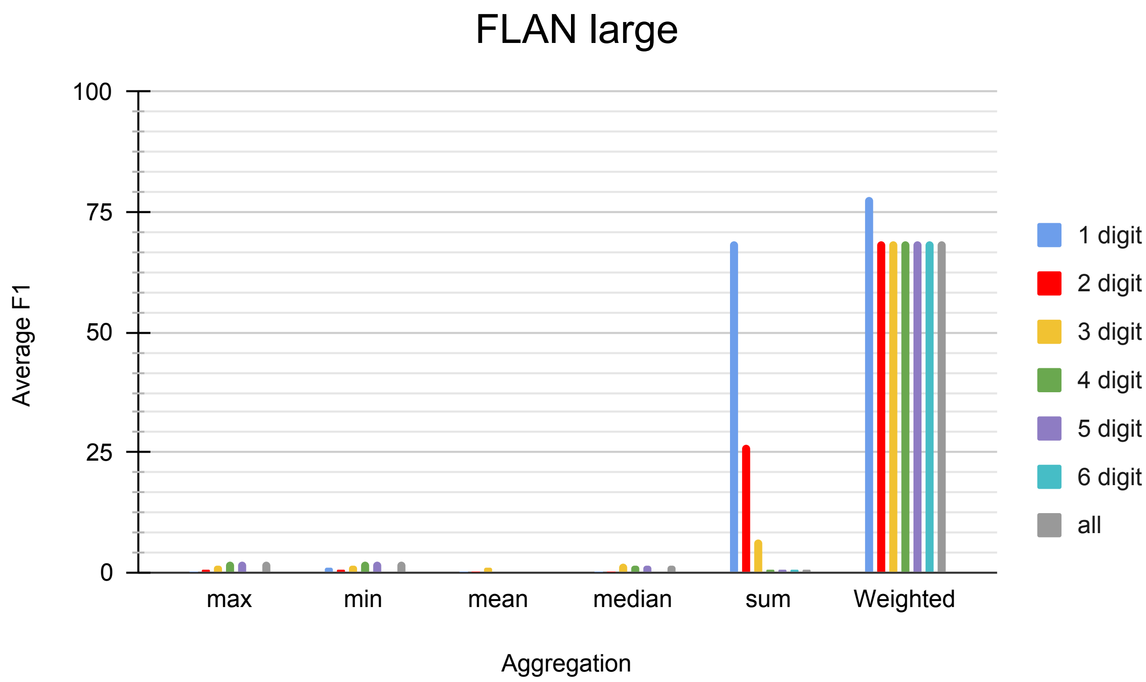

Figure 3: Average F1-score of FLAN large layer 1 numbers using max, min, median, mean sum and our weighted aggregation function with neighbourhood of 10.

Datasets	Models	Learning Rate	Epochs	Batch Size	Lambda	Training Time
MAWPS	BART base	1.00E-04	150	128	0.6	1h
	FLAN base		150	64	0.6	1h
	FLAN large		100	16	0.65	1.5h
FERMAT	BART base	1.00E-05	50	128	0.6	37h
	FLAN base		50	64	0.65	48h
	FLAN large		50	16	0.4	87h

Table 4: Specific hyperparameters for MAWPS and FERMAT based on the models trained. Training time is also provided as a rounded figure.

Incorporating Weights (CER %)		MAWPS	FERMAT													
			Original	Commuted	Integers 0 to 1000	2-digit integers	3-digit integers	4-digit integers	1000+	1000+ same	1dp random	2dp random	a+b	a-b	a*b	a/b
BART base (140M)	Digits	77.73	89.59	90.32	72.87	71.93	72.25	74.04	77.01	50.29	54.42	62.23	50.31	74.12	60.73	75.51
	[AGG] + Digits	**-1.79**	**-12.40**	**-0.83**	+0.46	+0.51	+1.19	-0.16	-0.44	+0.94	-1.38	-1.28	+3.08	**-1.58**	+1.21	**-2.22**
	Digits + Aux Loss	+0.76	-1.88	-0.53	+0.17	+0.20	+0.34	**-1.06**	**-0.53**	**-1.89**	**-1.59**	**-1.78**	**-2.45**	-0.23	**-2.75**	+0.26
FLAN base (250M)	Digits	67.71	75.32	169.52	67.37	67.68	67.94	67.86	68.86	50.95	43.77	47.80	39.84	87.81	60.96	91.52
	[AGG] + Digits	-0.98	**-1.40**	-0.29	**-1.11**	**-1.41**	**-1.19**	**-1.67**	-0.96	+1.26	-1.33	**-0.39**	**-1.64**	-1.94	-0.17	-0.50
	Digits + Aux Loss	**-1.54**	-0.83	**-1.09**	-1.09	-1.15	-0.80	-1.39	**-1.23**	**-2.09**	**-1.82**	-0.30	-1.25	**-3.15**	**-0.72**	**-0.93**
FLAN large (780M)	Digits	63.13	69.71	76.46	63.02	62.69	63.53	63.96	66.67	49.90	37.63	42.31	39.00	58.84	52.84	70.49
	[AGG] + Digits	-2.57	**-44.77**	**-10.81**	-1.02	-0.10	**-1.63**	-0.65	-0.89	+1.78	-0.93	-1.23	**-6.16**	**-7.80**	**-5.49**	**-7.19**
	Digits + Aux Loss	**-3.45**	-45.42	-2.72	**-1.20**	**-0.24**	-1.09	**-1.23**	**-1.31**	**-2.57**	**-1.11**	**-1.27**	-3.47	-6.14	-2.93	-4.74

Table 5: Results in Character Error Rate (CER) as a percentage over the target string with change from baseline after including aggregate embeddings in input embedding ([AGG] + Digits) and auxiliary loss (Digits + Aux Loss) for BART base, FLAN base and FLAN large. With CER, lower CER indicates a better performance, green highlight reduced CER i.e. negative change, and red the opposite. Darker shades of green and red indicate an absolute change greater than 1%.