Automated Frameworks of Semantic Augmentation to Improve Mathematical Word Problem Solving

Kunal Pai
University of California, Davis
kunpai@ucdavis.edu

Nishant Acharya
University of California, Davis
nacharya@ucdavis.edu

Zeerak Babar
University of California, Davis
zebabar@ucdavis.edu

Abstract

Mathematical word problems (MWP) are a challenging task for language models to solve, due to the complex logical structures and the need to remember context for longer periods of time. In this study, we develop a framework to generate and select prompts that utilize digit-level semantics as contextual information to improve accuracy of language models for solving mathematical word problems. We introduce a demonstration selection algorithm based on BLEU scores and Levenshtein distance, further enhanced by a prompt selection model to identify the most similar equation to the question for the 1-shot example. Our prompt generation frameworks build on the inference of previous studies by incorporating digit-level semantics to increase the accuracy of mathematical tasks.

We test multiple prompt generation frameworks and analyze the results on the SVAMP and GSM8K datasets. To evaluate our prompts' effect on the training process for small language models, we fine-tuned TinyLlama using our prompt-enhanced GSM8K questions as training samples, and tested on the SVAMP dataset. Our results show that adding additional digit-level context helps in increasing the accuracy of MWP tasks and generates more accurate sub-steps. Moreover, we found that different operations, due to their varying levels of complexity, require different types of context.

We further validate the importance of parametric knowledge by comparing inference-level and training-level results. We also analyzed the effectiveness of our demonstration selection algorithm, and identified areas for further improvement. Our code can be found here.

1 Introduction

Mathematics poses significant challenges for language models (Patel et al., 2021a). Several factors contribute to this difficulty:

- **Complex logical structures**: Mathematical problems involve intricate logic more so than tasks like language modeling or image classification, causing generalization issues.
- **Extended context retention**: Solving mathematical problems, particularly word problems, requires models to remember context over longer spans.
- **Accurate function learning**: Models must learn to prioritize correct arithmetic functions and avoid arbitrary ones to minimize errors.

A particularly challenging task is solving mathematical word problems, where problems are expressed in natural language and require translation into mathematical equations for solving. This demands a model to understand context clues and apply appropriate arithmetic functions.

Current literature addresses these challenges using various techniques. Methods incorporate external knowledge through encoders and verifiers to grasp mathematical structures (Yu et al., 2021; Cobbe et al., 2021). Cyclic reasoners are employed to retain long contextual information (Zhu et al., 2023). Detailed step-by-step solutions help models learn correct arithmetic functions (Lee et al., 2023). However, these methods often necessitate training or fine-tuning large models, which is costly.

An alternative is Chain of Thought prompting, which has shown success in structural reasoning tasks without extensive training (Wei et al., 2022; Zhou et al., 2023a; Zheng et al., 2023). Our study leverages this technique to **develop an algorithmic, automatic 1-shot prompting framework for solving mathematical word problems**.

We propose a framework that generates prompts, utilizing digit-level semantics, to enhance the prediction accuracy of both small and large language models. This framework aims to:

- **Generate prompts with digit-level semantics** to improve models' ability to construct

sub-steps necessary for operations such as carrying in addition and borrowing in subtraction.

- **Automatically select similar examples in a few-shot setting** to help models generate better sub-steps at the inference level.

The resulting dataset from our framework will support fine-tuning or training models to learn structural parametric knowledge, based on the insight that *models require additional contextual information to correctly learn mathematical functions*.

We will evaluate the efficacy of our framework using two models: PaLM2 by Google (Anil et al., 2023) as our inference-level large language model and TinyLlama by Facebook (Peiyuan, 2023) as our training-level small language model. Our primary datasets for prompt generation and testing are SVAMP and GSM8K (Cobbe et al., 2021). Our analyses aim to:

- Assess the ability of our prompting framework to generate more accurate answers on a pre-trained large model.
- Evaluate the improvement in accuracy of training small language models using our prompt generation strategy.
- Analyze our prompting framework's ability to generate more accurate sub-steps on a pre-trained large model.

Our main *technical contributions* are:

- Developing a 1-shot prompting framework that automatically selects a similar equation to that of the given mathematical word problem, using a mixture of BLEU score (Papineni et al., 2002), Levenshtein distance (Levenshtein et al., 1966), and a pre-trained selector. We aim to enhance the accuracy of solving tasks and digit-level sub-step generation.
- Creating operation-specific generalizable prompt templates with digit-level semantic context, usable at both inference and training levels.
- Analyzing the accuracy of these prompts across different arithmetic operations and their effectiveness in generating better sub-steps.

2 Method

2.1 Preliminaries

Our research aims to **identify and develop an automated prompt selection and generation framework, to enhance the accuracy of language models for mathematical word problem solving**. This involves generating prompts with digit-level sub steps and selecting example prompts with similar complexity as the question, for 1-shot prompting.

2.2 Techniques

Our study aims to create prompts and a prompt selection strategy at inference level, and we further test the efficacy of our prompts at training level, by fine-tuning a small language model. We mainly use two models in our study, PaLM2 (34 Billion Parameters) (Anil et al., 2023) and TinyLlama (1.5 Billion Parameters) (Peiyuan, 2023).

2.2.1 Inference Level

For the inference level experiments, we used PaLM2's default "text-bison-001" model (Anil et al., 2023) by Google.

We analyze different frameworks and validate their accuracy. These frameworks involve prompting the model with additional mathematical context, including the equation to solve the word problem and digit-level semantic information from a manual and a couple of automatic techniques. In doing so, we better the model's ability to generate the answer and sub-steps to get to the answer.

These frameworks can be viewed in Figure 1, and are as follows:

- **Basic**: In this framework, we prompt using just the word problem from the datasets. The query to the model is "`Solve this question: <QUESTION>`". The <QUESTION> is a placeholder for the actual question. This prompt requires no 1-shot example, as the most context that can be provided here is just the answer.

  To validate this strategy, we compare the generated answer against the actual answer to the question.

- **Text**: Developing on the previous approach, we add the additional context of the equation associated with the the word problem. The prompt for this approach is "`Solve this question: <QUESTION>. The`

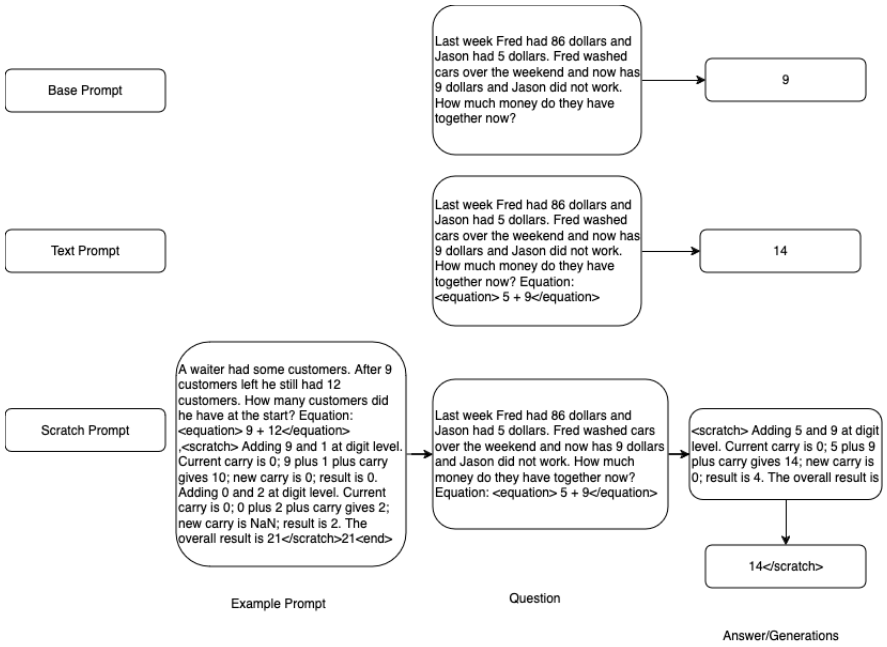

Figure 1: The above figure shows an example of the three different types of prompts. The **base prompt** takes just the question, its example provides no extra context. The **text prompt** provides the equation of the question as an additional prompt. Lastly, the **scratch prompt** provides detailed mathematical information in the example, which is used to generate the same type of information for the actual question, as seen in the second last block, which is then used to generate the answer.

`equation to solve the question is <equation> <EQUATION> </equation>`". The <EQUATION> and <QUESTION> are placeholders for the actual equation and question. The provided equation is the entire equation, each number present in the equation is also present in the question itself. It is not broken down into sub-steps.

This approach is validated by comparing if the model can infer the answer based on the provided equation.

- **Static Simplified Scratch (SSS) 1-Shot**: In this framework, we begin by extracting the word problems and equations from the designated datasets. The equations are then simplified into two operands and a single operator. For example, if the equation required to solve a problem is `5 + 3 + 2`, the *simplified* equation would be `8 + 2` {`5 + 5` is also a valid simplification}.

  The previous step is used to regulate the amount of information in the context, so that the number of sub-steps does not blow up, and to ensure consistency between all equations used in the prompt. The sub-steps are **generated using a arithmetic algorithm**, and highlights the digit-level semantics. The generated sub-steps for the above equation are, "`Adding 8 + 2: 10; Result 0; Carry 1; Adding 1 + 0: 1; Result 1; Final Result 10`". This context will be addressed as the **scratch** section.

  During inference, we use a 1-shot approach where we provide a single example of a similarly complex question, complete with its solution and scratch section. This example, which we refer to as the **gold example**, is selected manually in the SSS framework. We ensure the gold example demonstrates crucial concepts such as carrying and borrowing, and matches the complexity of the target question. After presenting the gold example, we ask PaLM2 to generate the scratch section for the target question based on this example, and then infer the answer based on its generated scratch section. The prompt for this approach looks like: "`Look at the question, equation and scratch section as follows: <QUESTION>. The equation to solve the question is <equation> <EQUATION>`

`</equation>. The scratch section is <scratch> <SCRATCH> </scratch>. Generate a methodology for the following question and equation: <QUESTION>. The equation to solve the question is <equation> <EQUATION> </equation>`". The <EQUATION>, <QUESTION> and <SCRATCH> are placeholders for the actual equation, question and scratch parts. The model then generates the target question's scratch section. The next prompt involves asking it to infer the answer from the provided scratch section, and is as follows: "`Look at this scratch section: <scratch> <SCRATCH> </scratch>. What is the answer to the question?`".

For validation, we compare the target question's scratch section to the scratch section that has been generated by our generation script, using BLEU score. We also check for the accuracy of the answer.

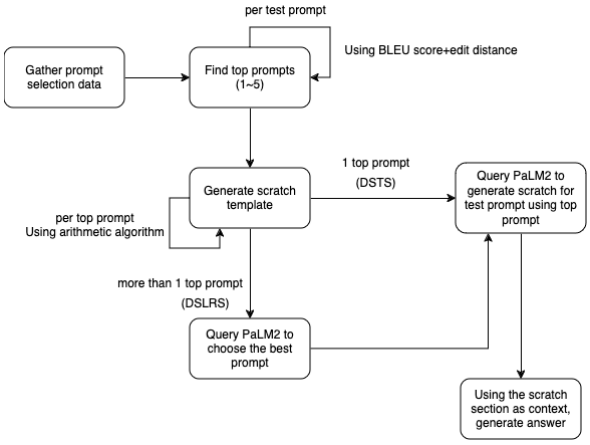

Figure 2: Dynamic Simplified Scratch 1-Shot techniques

- **Dynamic Simplified Scratch 1-Shot**: The previous approach relied on manually chosen example prompts, known as gold examples, which is not scalable. To address this, we introduce a **demonstration selection algorithm** to select a personalized example for each target question. Similar to the previous approach, we extract the question and its equation. But then, we use the demonstration selection algorithm to find the best example for that question instead of manual selection. The rest of the procedure remains the same as in the SSS 1-Shot approach.

  The demonstration selection algorithm works as follows:

  – Per equation, we pre-process it to remove whitespaces, parentheses, and other non-essential characters, resulting in a string of two numbers and an operation.
  – We then append a boolean to each equation, `True` or `False`, based on whether the equation involves carrying or borrowing. This semantic information further aids in understanding the complexity of the equation.
  – We then use a combination of BLEU score (Papineni et al., 2002) and Levenshtein distance (Levenshtein et al., 1966) to compare the *equation* of the question being solved to the other *equations* in the dataset.
  – We then choose the equations with the top 5 highest BLEU scores and lowest Levenshtein distances as the gold examples.

  We tried two variants of the dynamic scratch 1-shot framework, which can also be seen in Figure 2:

  Dynamic Simplified Top Scratch (DSTS) 1-Shot: In this variant, we use the top prompt, out of the top 5 prompts, as the gold example for the 1-shot prompting.

  Besides the gold example, the approach is the same as the SSS 1-shot approach. Therefore, we use the same prompts to guide the model to generate the scratch section and to generate the answer based on the scratch.

  For validation we check the generated scratch's similarity, using BLEU score, to the ground truth generated by our scratch generation script. We also check for the accuracy of the final answer.

  Dynamic Simplified LLM Retrieval Scratch (DSLRS) 1-Shot: To provide some more flexibility in choosing a different gold example, and avoid giving the model very similar equations for 1-shot, we use this variant where we ask a selector model to choose the best example for the target question. In this case, due to resource constraints, we use

PaLM2's "text-bison-001" (Anil et al., 2023) as the selector.

We ask the selector model to choose the closest equation, from the top 5 closest equations, as determined by the aforementioned demonstration selection algorithm. The prompt for that is "`Look at this equation: <equation> <EQUATION> </equation>. Which of the following equations is the closest to the equation? <equation> TOP1 </equation>, <equation> TOP2 </equation>, <equation> TOP3 </equation>, <equation> TOP4 </equation>, <equation> TOP5 </equation>`". The rest of the process and the prompts are similar to SSS 1-shot. Based on the equation picked by the selector, we algorithmically generate its digit-level scratch section, and use that as the gold example, to assist the model in generating its own scratch section, and infer the answer from it.

For validation, similar to previous approaches, we use BLEU score to calculate the scratch generation accuracy. We also check for the final answer accuracy.

2.2.2 Training Level

Another aspect of our study involved evaluating how effectively our frameworks could be used for fine-tuning or training a small language model. We aim to optimize the training process for small language models and develop a resource-efficient solution for learning to solve mathematical word problems. For the training level experiments, we used TinyLlama (Peiyuan, 2023) by Facebook. We used the latest checkpoint for the model, which has 1.5 billion parameters.

We used the same frameworks as the inference level experiments to generate the prompts. All inference on the fine-tuned model was conducted in a zero-shot manner, meaning no examples were selected. This is because the demonstration selection algorithm and manual gold example selection are inference level concepts, and hence, are not feasible for training level experiments. Therefore, we only used the **basic**, **text**, and **scratch** frameworks for the training level experiments.

We fine-tuned six models, 3 per operation, for the training level experiments:

- **Basic Addition/Subtraction**: This model was trained on GSM8K addition and subtraction questions and answers. The training samples did not contain any extra context.
- **Text Addition/Subtraction**: Similar to inference, the text model was trained on questions, the equations to solve those questions and the answer.
- **Detailed Scratch Addition/Subtraction**: Building on top of the previous model, this model also has the scratch section generated from our generation script appended as semantic information after the questions and the equations.The equations are *not simplified*, unlike the inference level experiments.

The models were evaluated for their generalizability on the SVAMP dataset (Patel et al., 2021b), with final answer accuracy as our metric. Due to resource constraints, we implemented several optimization techniques, such as using low-rank weight matrices for training, employing a paged optimizer to speed up gradient recording, and reducing float size to 8 bits for storage, which were then up-sized during usage.

3 Experiments

3.1 Experimental Setup

The inference experiments were conducted on a MacBook Air with an M1 chip. The MacBook Air has 8 cores and 16 GB of RAM. Since the PaLM2 model has an API, we wrote a Python script to interact with the API. The script prompts the model with the different frameworks, and then compares the model's answer to the expected answer. The script also calculates the BLEU score of the model's scratch section, to see if it can generate the correct sub-steps. The Python libraries used in the script are *google.generativeai v0.6.2*, *json v3.10.2*, and *nltk v3.8.1*.

The TinyLlama model was fine-tuned on a Google Colab Pro (Bisong and Bisong, 2019) instance, a mixture of A100 and L4 GPUs were used which have a RAM of 40 GB and 22.4 GB GPU RAM respectively.

3.2 Datasets

We used the following datasets for our experiments:

- **SVAMP (Patel et al., 2021a)**: The SVAMP dataset is a dataset of 1000 challenging math

word problems. There are 195 addition examples and 507 subtraction examples.

- **GSM8K (Cobbe et al., 2021)**: The GSM8K dataset is a dataset of 8000 grade school level math word problems. There are 2371 addition examples and 1979 subtraction examples.

3.3 Results

3.3.1 PaLM2

Our results for the PaLM2 model can be seen in Tables 1 and 2. The results for the BLEU scores of the scratch section generation can be seen in Tables 3 and 4.

Table 1: PaLM2 Accuracy on SVAMP Dataset

Operation	Basic	Text	SSS 1-Shot	DSTS 1-Shot	DSLRS 1-Shot
Addition	59%	80%	80%	**81%**	68%
Subtraction	70%	81%	**89%**	77%	79%

Table 2: PaLM2 Accuracy on GSM8K Dataset

Operation	Basic	Text	SSS 1-Shot	DSTS 1-Shot	DSLRS 1-Shot
Addition	38%	**87%**	66%	44%	44%
Subtraction				82%	80%

Table 3: PaLM2 BLEU Scores on SVAMP Dataset Scratch Section Generation

Operation	SSS 1-Shot	DSTS 1-Shot	DSLRS 1-Shot
Addition	0.80	**0.87**	0.85
Subtraction	0.82	**0.89**	0.88

3.3.2 TinyLlama

Our results for the TinyLlama model can be seen in Table 5.

3.4 Analysis

PaLM2: Our Scratch prompting frameworks beat the Basic and Text based prompting frameworks for PaLM2. For SVAMP the highest accuracy for subtraction prompts was 89% using the SSS 1-shot framework, and for addition was 81% DSTS 1-shot framework. For GSM8K, our prompting frameworks did better than the baseline for both subtraction and addition, but worse than the Text framework.

Our demonstration selection technique improved the accuracy of addition prompts when selecting the most similar equation from the top 5 options. For subtraction, accuracy improved when a prompt selector (PaLM2 in our study) was used to choose the most similar equation from the top 5.

Table 4: PaLM2 BLEU Scores on GSM8K Dataset Scratch Section Generation

Operation	SSS 1-Shot	DSTS 1-Shot	DSLRS 1-Shot
Addition	0.25	**0.45**	**0.45**
Subtraction	0.25	**0.47**	0.46

Table 5: TinyLlama Accuracy on SVAMP Dataset

Operation	Basic	Text	Detailed Scratch
Addition	1.025%	0%	**77.659%**
Subtraction	0.7%	4.133%	**26.967%**

We believe that the drop in accuracy for GSM8K was due to PaLM2 already having parametric knowledge of GSM8K. Adding additional context in the prompt led to overfitting and hallucination of answers.

Our DSTS 1-shot framework improved the accuracy of addition. We believe this is due to addition being a simpler function to learn. We theorize that the DSTS 1-shot framework could not improve the accuracy of subtraction since it is comparatively harder, and needs more information to capture the complexity of it. The concept of carrying in addition is straightforward: if the sum of two numbers is greater than 9, then 1 is carried over to the next digit. However, the concept of borrowing is more complex: if the number being subtracted is greater than the number it is being subtracted from, the model borrows 1 from the next digit. This reduction due to borrowing can lead to more complex situations, such as reducing a 0 (treated as 10) to 9, which requires a multi-step process. Additionally, the direction of operations is important; carrying works in the same direction as digit-level addition (right to left). In contrast, borrowing works from left to right, while digit-level subtraction happens in the opposite direction, which can cause the model to be confused when trying to infer information from the context, making subtraction harder for it.

Our current methodology for choosing a demonstration uses digit-level semantic similarity through BLEU scores and Levenshtein distance over the equation. However, this may not be enough to capture the complexity of a subtraction equation.

From picking the gold example in SSS 1-shot, we found that using examples similar in complexity, rather than exactly the same, provides much better accuracy for subtraction. While this is possible in SSS 1-shot due to manual demonstration selection, automating this requires a more robust metric that captures the complexity of the problem, not just semantic similarity.

Moreover, we believe that due to the model having some parametric knowledge of arithmetic operations, less complex prompts were supplemented by this parametric knowledge. This may explain why the DSLRS 1-shot framework performed slightly better on subtraction questions. This hypothesis is further validated by the fact that this behavior is not observed in TinyLlama, where no parametric knowledge is present, since we are training the model.

Our DSTS 1-shot framework also improves the BLEU score of the scratch section generation for both addition and subtraction operations on the SVAMP dataset and the GSM8K dataset. The model was able to generate more correct sub-steps for the scratch section using the most similar equation to it. This makes sense, as seeing more similar digit-level operations being performed, assist the model in generating more accurate sub-steps. We noticed BLEU scores of 0.87 for addition and 0.89 for subtraction on the SVAMP dataset, and BLEU scores of 0.45 for addition and 0.47 for subtraction on the GSM8K dataset. The other frameworks did not perform as well as the DSTS 1-shot framework, because the model had to rely on its own structural understanding to construct the sub-steps, which it may not have been able to do as well as a case where it sees the operation being performed on similar numbers.

Our results indicate that having a single best prompt works the best for generating digit-level sub steps for solving a mathematical word problem. We also found that addition is semantically simpler and easier to generalize than subtraction. We believe that, in the future, using a metric to capture the mathematical complexity of a problem for demonstration selection is better suited to solve subtraction problems. To merge multiple arithmetic functions, a non linear metric needs to created, which means that a multi-layer feed forward network, or similar class of models, can be used for improving demonstration selection.

TinyLlama: The fine-tuned TinyLlama model had the best accuracy for scratch framework, both for addition and subtraction, with the highest accuracy of 77.659% on addition questions, and 26.967% on subtraction questions. This suggests that our technique of algorithmically generating digit-level contextual sub-steps significantly enhances the model's ability to learn these functions compared to using no context or minimal context.

It is imperative to use the *entire* detailed equation, as it helps the model better connect the numbers in the question to the numbers in the equation. We do see a discrepancy in the Text framework experiments for TinyLlama, where addition has a 0% accuracy. We believe this is a byproduct of answer hallucination.

3.5 Takeaways from Alternative Frameworks

To further analyze the prompt selection and generation strategies, we experimented with various approaches. Our insights from these experiments are as follows:

- We attempted a dynamic Python code based framework to generate contextual information and answers. In this framework we ask the model to generate Python code to solve the question, and then run the code to obtain the answer. This approach was unsuccessful, likely due to insufficient connections between the question, the equation, and the code, which led to model hallucinations. The accuracy for this approach was 57% for addition, 66% for subtraction on SVAMP, for just using code. Adding additional context in the form of an equation decreased the accuracy by a few point, 52% for addition and 65% for subtraction.

- We used the Text framework on the SVAMP dataset for multiplication and division. Adding equations as context increased the accuracy from 58% to 64% for multiplication and from 85% to 91% for division. We also applied the SSS 1-shot framework to the same examples. For multiplication, accuracy increased to 69%, while for division, accuracy decreased to 66%. We believe this decrease is due to division being similarly complex to subtraction, with the model having a better parametric understanding of this operation. Therefore, this approach may not generalize as well for division as it might for multiplication.

4 Related Work

Mathematical Word Problem Solving: Understanding Mathematical Word Problems (**MWP**) is a critical task, and research in this area using Large Language Models (LLMs) has advanced significantly. Parametric knowledge and embeddings are popular methods for learning mathematical structures in LLMs. Yu et al. introduced the idea of using an external knowledge encoder to better learn the structure of the MWP (Yu et al., 2021). Another avenue is to increase the reasoning capabilities of the LLM, for example, Zhu et al. proposed a reasoner to mimic human reasoning processes when solving MWPs (Zhu et al., 2023). Similarly, Cobbe et al. introduced the concept of verifiers to evaluate generated answers and use this metric in training (Cobbe et al., 2021). Patel et al. explored the complexity of the current benchmark datasets, and how treating MWPs like a bag of words does not work on challenging questions (Patel et al., 2021b). Lee et al. developed a chain of thought training set to train small language models arithmetic, with the key idea being having informative digit-level information (Lee et al., 2023).

Chain of Thought prompting: LLMs have been shown to be successful in natural language processing, and multi-step prompts can be used to guide a LLM towards the correct answer, given several examples. Wei et al. explored how generating intermediate chain of thought steps in LLMs during inference increased reasoning task accuracy (Wei et al., 2022). Zhou et al. examined using chain of thought prompting to make LLM generation more contextually faithful (Zhou et al., 2023b). To increase the multi-modal reasoning accuracy over images and text, Zheng et al. introduced DDCoT, a zero-shot prompting strategy to break the question into simpler steps to generate the correct answer (Zheng et al., 2023).

5 Conclusion and Future Work

Conclusion: In this project we developed and explored frameworks to automatically generate and select digit-level semantic context for prompting, to increase the accuracy of mathematical word problem solving tasks. Our motivation was to create a framework that could effectively direct a model to solve mathematical word problems. To this end, we tested our framework at the inference level using PaLM2 and at the training level using TinyLlama, primarily utilizing the GSM8K and SVAMP datasets for testing and fine-tuning.

Our results indicate that the DSTS 1-shot framework, which employs BLEU score and Levenshtein distance for demonstration selection, performs the best in generating the closest correct sub-steps for addition and subtraction, and the correct answers to word problems for addition. The results for subtraction suggest that similarity alone is insufficient; we need more dimensions for domain selection. Specifically, the metric should capture the complexity of the problem and the borrowing process while minimizing the overfitting aspect of prompting.

Another takeaway is that different mathematical operations have varying levels of complexity. Addition is observed to be the simplest, followed by multiplication (which can be rewritten as a series of additions), then subtraction, and finally division. As mentioned above, a more complex metric needs to be created for demonstration selection for increasing the accuracy on these tasks. We believe one such avenue can be to use models from the same family as a feed forward network with non-linear activation functions to model the different metric for each operation into a single step-wise metric.

Future Work: In the future, we plan to explore:

- **Update Demonstration Selection Algorithm**: We aim to improve the demonstration selection algorithm to better choose the gold example for the dynamic scratch 1-shot frameworks. This involves using a feed-forward network or FFN-based demonstration selector that incorporates current and additional metrics to model mathematical complexity similarity.

- **Semantic Information**: We plan to update our generation techniques to ensure semantic information is consistent for subtraction and division. One approach is to reverse the digit-level arithmetic, so that borrowing and the operation are in the same direction.

- **Different Operations**: Our ultimate goal is to develop a prompt selection and creation framework that enhances the accuracy of a range of arithmetic operations. To this end, we will integrate more arithmetic operations, such as multiplication, division, modulo, and others, into our framework.

References

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint arXiv:2305.10403*.

Ekaba Bisong and Ekaba Bisong. 2019. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems.

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

Vladimir I Levenshtein et al. 1966. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021a. Are NLP models really able to solve simple math word problems? In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2080–2094, Online. Association for Computational Linguistics.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021b. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Zhang Peiyuan. 2023. Tinyllama. https://github.com/jzhang38/TinyLlama.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Weijiang Yu, Yingpeng Wen, Fudan Zheng, and Nong Xiao. 2021. Improving math word problems with pre-trained knowledge and hierarchical reasoning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3384–3394, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Ge Zheng, Bin Yang, Jiajin Tang, Hong-Yu Zhou, and Sibei Yang. 2023. Ddcot: Duty-distinct chain-of-thought prompting for multimodal reasoning in language models. *Advances in Neural Information Processing Systems*, 36:5168–5191.

Wenxuan Zhou, Sheng Zhang, Tristan Naumann, Muhao Chen, and Hoifung Poon. 2023a. Continual contrastive finetuning improves low-resource relation extraction. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13249–13263, Toronto, Canada. Association for Computational Linguistics.

Wenxuan Zhou, Sheng Zhang, Hoifung Poon, and Muhao Chen. 2023b. Context-faithful prompting for large language models. *arXiv preprint arXiv:2303.11315*.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Yongfeng Huang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. 2023. Solving math word problems via cooperative reasoning induced language models. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4471–4485, Toronto, Canada. Association for Computational Linguistics.