Learning Multi-Step Reasoning by Solving Arithmetic Tasks

Tianduo Wang and **Wei Lu**
StatNLP Research Group
Singapore University of Technology and Design
{tianduo_wang,luwei}@sutd.edu.sg

Abstract

Mathematical reasoning is regarded as a necessary ability for Language Models (LMs). Recent works demonstrate large LMs' impressive performance in solving math problems. The success is attributed to their Chain-of-Thought (CoT) reasoning abilities, i.e., the ability to decompose complex questions into step-by-step reasoning chains, but such ability seems only to emerge from models with abundant parameters. This work investigates how to incorporate relatively small LMs with the capabilities of multi-step reasoning. We propose to inject such abilities by continually pre-training LMs on a synthetic dataset **MsAT** which is composed of **M**ulti-**s**tep **A**rithmetic **T**asks. Our experiments on four math word problem datasets show the effectiveness of the proposed method in enhancing LMs' math reasoning abilities.[1]

1 Introduction

Making Language Models (LMs) perform mathematical reasoning is a valuable, yet challenging research objective (Hendrycks et al., 2021; Cobbe et al., 2021). Recently, we have witnessed large-scale LMs' impressive performance on a series of reasoning tasks via *chain-of-thought* prompting (Wei et al., 2022). This method elicits large LM's ability to decompose a complex problem into several intermediate steps. However, it is believed that such ability only emerges from sufficiently large models (empirically more than 100B parameters) (Wei et al., 2022). In this paper, we examine how to incorporate moderate-sized LMs, e.g., RoBERTa (Liu et al., 2019), with such multi-step reasoning ability via continual pre-training to improve the performance on math problems.

Correctly understanding numbers is a prerequisite of mathematical reasoning abilities. But Wallace et al. (2019) shows that medium-sized LMs have a deficiency in numerical comprehension. To overcome this issue, previous works inject numerical reasoning skills into LMs following two approaches. The first is masking numbers with special tokens, and generating symbolic expressions with a structured neural decoder (Xie and Sun, 2019; Jie et al., 2022). An example of such expression is provided in Figure 1. The second strategy continually pre-trains LMs on synthetic numerical tasks, which requires models to learn how to perform computation involving numbers (Geva et al., 2020; Pi et al., 2022).

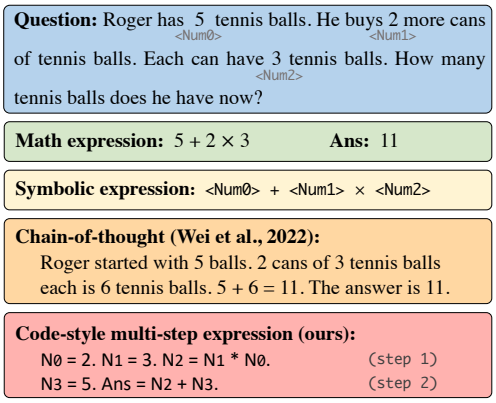

Figure 1: A math word problem example with different kinds of answers. In **Question**, <Num0>, <Num1>, and <Num2> are special tokens used for masking numbers.

However, both approaches suffer from critical limitations. For symbolic methods, they neglect the information carried by the numbers, which could provide crucial hints for solving math problems (Wu et al., 2021; Liang et al., 2022). As for continual pre-training methods, LMs' arithmetic skills are not reliable. Previous works indicate that such skills are highly influenced by the training data (Razeghi et al., 2022) and hard for extrapolation (Wallace et al., 2019).

Motivated by these shortcomings, we propose to first pre-train moderate-sized LMs on a synthetic dataset called MsAT (Multi-step Arithmetic Tasks)

[1] Our code and data are released at https://github.com/TianduoWang/MsAT.

Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics
Volume 2: Short Papers, pages 1229–1238

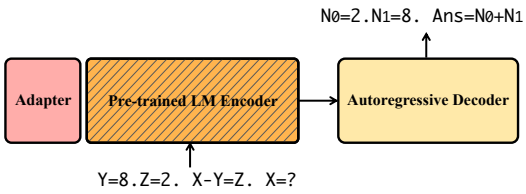

Figure 2: An illustration of the continual pre-training process on our Seq2Seq model. We attach adapter modules to each layer of LM encoder and fix LM's parameters (shaded area) during pre-training. Tokens N_0, N_1, and Ans in the output are the variable names only used by the decoder. Our DAG structured model is similarly pre-trained with the only difference on the decoder part.

before downstream task fine-tuning. To make sure LMs capture the information carried by the numbers, we keep the numbers in the questions instead of masking them during both pre-training and fine-tuning. Instead of making LMs conduct computation internally, MSAT encourages LMs to generate a series of intermediate steps leading to the answer. Experiments on four math word problem datasets with two backbone models demonstrate the effectiveness of our method in enhancing LMs' math reasoning performance.

2 Method

Our method essentially appends a continual pre-training stage before fine-tuning LMs on downstream tasks. The continual pre-training serves two purposes: first, we tokenize numbers digit-by-digit to improve LMs' numerical comprehension; second, we make LMs learn multi-step reasoning skills from the proposed synthetic task.

2.1 Digit tokenization for numbers

Sub-word tokenization methods, e.g., byte pair encoding (BPE) (Sennrich et al., 2016), is one of the reasons why moderated-sized LMs poorly understand numbers (Wallace et al., 2019). BPE-based tokenizers split text based on the token frequency in the training corpus, which can be counter-intuitive when dealing with numbers. For example, numbers "520" and "521" will be tokenized into ["520"] and ["5", "21"] respectively by the `RoBERTaTokenizer`[2] of the `Transformers` library (Wolf et al., 2020). Such inconsistent tokenization strategy for numbers undermines LM's numerical understanding ability. Hence, we tokenize numbers digit-by-digit for both pre-training and fine-tuning.

[2] https://huggingface.co/docs/transformers/model_doc/roberta

2.2 Multi-step Arithmetic Tasks (MSAT)

The core of our method is the synthetic task MSAT where LMs can learn multi-step reasoning skills. Like MWP tasks, MSAT can be formulated as a Seq2Seq task: the input of a MSAT example describes an arithmetic question, while the output is a reasoning chain leading to the answer. Specifically, each input sequence is composed of three components: *question context*, *equation*, and *question variable*. Equation is a sequence of symbols and operators ($+$, $-$, $\times$, $\div$, $=$) that builds equality relationship between symbols. Given an equation, only one of the symbols is set as the question variable, while other symbols will be listed in question context with their numerical values.

The output sequence of MSAT is constructed in a code-style multi-step reasoning format. Each step consists of two sub-steps: *variable assignment* and *calculation*. In variable assignment, numbers appear in the input sequence are assigned to the variable names that are exclusive for decoder. In calculation, a new variable is generated from the calculation of the existing variables. This makes our outputs become executable Python code so that the numerical answer can be calculated by an external Python interpreter. Both inputs and outputs of MSAT are generated purely automatically. Details about the construction of MSAT are provided in Appendix A.1.

2.3 Pre-training via adapter-tuning

Directly training on synthetic data that are largely different from the natural language corpus harms LMs' language prowess (Geva et al., 2020). Therefore, we adopt a two-stage tuning strategy (Wang and Lu, 2022) to inject reasoning skills into LMs. Specifically, we perform adapter-tuning (Houlsby et al., 2019) on MSAT and then jointly fine-tune adapter and LM backbone on downstream tasks. It mitigates catastrophic forgetting because LM's original parameters are largely preserved during adapter-tuning (Houlsby et al., 2019).

We consider two backbone models to verify the effectiveness of our method. In particular, we select a sequence-to-sequence (Seq2Seq) model (Lan et al., 2021) and a directed acyclic graph (DAG) structured model (Jie et al., 2022) that both adopt $\text{RoBERTa}_{\text{base}}$ to encode the input questions. More details of these models are provided in §3.1. Figure 2 shows an overview of the proposed pre-training method.

Model	MAWPS		ASDiv-A		SVAMP		SVAMP (hard)	
	Acc.	Δ	Acc.	Δ	Acc.	Δ	Acc.	Δ
Large language models	(PaLM 540B)		(code-davici-002)		(PaLM 540B)			
w/ Chain-of-Thought prompting	93.3		80.4		**79.0**		-	
Seq2Seq models								
RoBERTaGen (Lan et al., 2021)								
w/ symbolic masks	88.4		72.1		30.3		30.3♡	
w/ digit tokenization	84.1	(−4.3)	71.9	(−0.2)	27.6	(−2.7)	19.6	(−10.7)
MSAT-RoBERTaGen (Ours)	**91.6**	(+3.2)	**81.8**	(+9.7)	**39.8**	(+9.5)	**36.2**	(+5.9)
DAG structured models								
DeductReasoner (Jie et al., 2022)								
w/ symbolic masks	92.0		85.0		45.0		45.0♡	
w/ digit tokenization	91.6	(−0.4)	84.1	(−0.9)	44.4	(−0.6)	42.8	(−2.2)
MSAT-DeductReasoner (Ours)	**94.3**	(+2.3)	**87.5**	(+2.5)	**48.9**	(+3.9)	**48.2**	(+3.2)

Table 1: Accuracy (%) comparison between large language models (LLMs), backbone model baselines, and our method. Δ: performance gap compared with the symbolic mask baselines. ♡: For baselines with symbolic masks, performance on SVAMP (hard) is the same as SVAMP because the actual numbers are replaced by symbolic tokens. The results of LLMs with chain-of-thought prompting are from Wei et al. (2022).

3 Experiments

Now we investigate whether our pre-training method facilitates models on Math Word Problem (MWP) solving tasks. All results are averaged over three different runs.

3.1 Experimental setup

Existing datasets We consider three commonly-used MWP datasets: MAWPS (Koncel-Kedziorski et al., 2016), ASDiv-A (Miao et al., 2020), and SVAMP (Patel et al., 2021). The statistics of these datasets is provided in Table 2. More details can be found in Appendix A.2. We report five-fold cross-validation results for both MAWPS and ASDiv-A and test set accuracy for SVAMP following previous practice (Lan et al., 2021; Jie et al., 2022).

SVAMP (hard) We find more than 85% of the numbers in the above datasets are smaller than 10^2. To investigate the extrapolation performance of the models trained with MSAT, we create SVAMP (hard) from the original SVAMP dataset by replacing the numbers with much larger ones inspired by Gao et al. (2022). More details about SVAMP (hard) and number distribution of the existing datasets are provided in Appendix A.3.

Dataset	# Data	Avg. input length	Avg. output reasoning steps
MAWPS	1,987	30.3	1.4
ASDiv-A	1,217	32.3	1.2
SVAMP	1,000	34.7	1.2

Table 2: Existing dataset statistics.

Models We consider both sequence-to-sequence (Seq2Seq) models and directed acyclic graph (DAG) structured models as our backbone models. For Seq2Seq model, we choose RoBERTaGen (Lan et al., 2021), an encoder-decoder model with $\text{RoBERTa}_{\text{base}}$ as the encoder combined with a Transformer decoder. For DAG structured model, we choose DeductReasoner (Jie et al., 2022) that combines $\text{RoBERTa}_{\text{base}}$ with a DAG decoder. In their original implementation, both models replace numbers with symbolic mask tokens. Hence, we additionally consider a baseline for each backbone model that uses actual numbers with digit tokenization. We name the models that are based on these two backbone models and pre-trained with our method as MSAT-RoBERTaGen and MSAT-DeductReasoner respectively. We also compare our models to large LMs, e.g., PaLM (Chowdhery et al., 2022) and Codex (Chen et al., 2021), with chain-of-thought prompting (Wei et al., 2022). All models are evaluated via greedy decoding. More implementation details, e.g., training hyper-parameters, are provided in Appendix B.

3.2 Main results

Table 1 compares our models with backbone model baselines and large LMs. On all datasets, digit tokenization baselines consistently perform worse than their symbolic mask counterparts, indicating the deficiency of the numeracy comprehension of the original RoBERTa model. However, the models trained with MSAT surpass both baselines by a large margin, which demonstrates the effectiveness of our pre-training method.

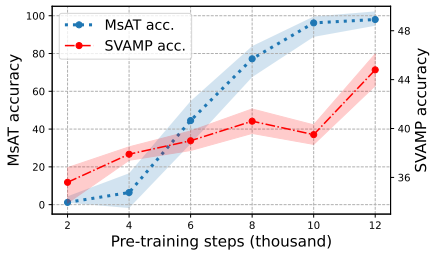

Figure 3: Performance on MSAT and SVAMP with respect to the pre-training steps. Results are obtained from 3 different runs.

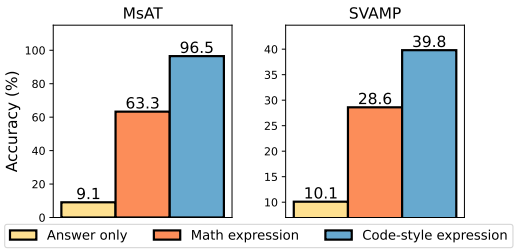

Figure 4: Comparison between different output expression formats. Results are obtained from our Seq2Seq model (with code-style expressions) and its variants.

SVAMP (hard) We can observe that, on SVAMP (hard), the accuracies of digital tokenization baselines decrease dramatically (10.7 points drop for ROBERTAGEN and 2.2 points drop for DEDUCTREASONER) compared with baselines with symbolic masks, while the models trained with MSAT still outperforms symbolic mask baselines by 5.9 and 3.2 points respectively. This shows that not only does our models obtain better results than the baselines on the existing tasks, but it is also more robust in handling out-of-distribution numbers.

Compare with large language models We also observe that, on relatively simple tasks, i.e., MAWPS and ASDiv-A, RoBERTa-based models can outperform large LMs. But for the more challenging task SVAMP, there is still a large performance gap. We believe this is because SVAMP requires models to have a better understanding of natural languages. Jie et al. (2022) also reports that varying LM encoders results in significant performance disparities on SVAMP, indicating that SVAMP performance is closely tied to model's natural language capabilities.

4 Pre-training analysis

In this section, we provide a careful analysis of our pre-training method from various perspectives to understand why it works.

4.1 Pre-training task performance

We visualize how the performance of pre-training task MSAT and one of the MWP tasks SVAMP changes with pre-training steps in Figure 3. It can be observed that the performance on both synthetic and natural language tasks tends to improve gradually as the number of pre-training steps increases. Figure 3 demonstrates that LMs are capable of learning multi-step reasoning gradually from the synthetic task MSAT. The acquired multi-step reasoning ability can subsequently be transferred to the downstream MWP solving tasks, enhancing performance during the fine-tuning phase.

4.2 Reasoning format of MSAT

The reasoning format of MSAT dictates the specific reasoning skills that LMs will acquire during pre-training. We demonstrate the superiority of our code-style multi-step reasoning format by comparing it with two different reasoning expressions.

Effect of producing intermediate steps While it is a common practice to train LMs towards directly producing the numerical answers of the arithmetic questions (Geva et al., 2020; Pi et al., 2022), a recent work shows that LMs' arithmetic skills are not reliable (Razeghi et al., 2022). To explore whether LMs can learn reasoning skills from MSAT without intermediate steps, we pre-train LMs on a variant of MSAT by replacing step-by-step output sequences with only numerical answers. Figure 4 compares this model (answer only) with our model (code-style). Its poor performance on both MSAT and SVAMP confirms the necessity of producing intermediate reasoning steps during pre-training.

Structured code-style expression We next investigate the importance of applying the structured code-style reasoning expressions by comparing it with the less formatted math expressions. We argue that, compared with math expressions that only contain numbers and operators, our code-style expressions are more suitable for multi-step reasoning due to the structure information in the output sequences. Our experiments in Figure 4 demonstrate the superiority of the code-style output expressions. We can see that models with math expressions perform consistently worse than models with code-style multi-step reasoning format on both pre-training task MSAT and MWP solving task SVAMP.

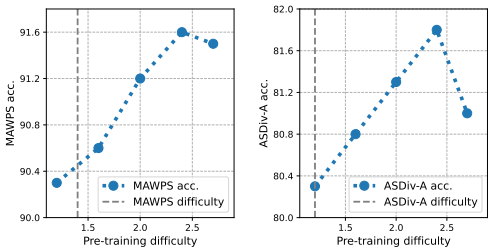

Figure 5: Performance on MAWPS and ASDiv-A with respect to pre-training difficulty. The difficulty levels of two MWP tasks are also added for reference.

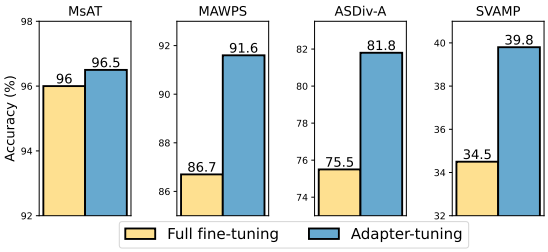

Figure 6: MSAT and downstream task performance comparison between full fine-tuning and adapter-tuning during pre-training.

4.3 Difficulty level of MSAT

Leveraging synthetic data for pre-training provides the advantage of enabling highly customizable difficulty levels for the training data. Here we define the difficulty level of a reasoning task as the averaged reasoning steps that are required to solve the problems. From Figure 5, we see that pre-training LMs on MSATs that are harder than downstream tasks generally leads to better results. It's important to note that, broadly speaking, the difficulty level of a reasoning task, particularly those involving natural language, is not solely determined by the number of reasoning steps. One example is that, though both ASDiv-A and SVAMP have an averaged reasoning steps of 1.2 (see Table 2), SVAMP is considered more difficult as it requires high-level natural language understanding (Patel et al., 2021).

4.4 Perform adapter-tuning on MSAT

Tuning all parameters of LM encoders on synthetic data that are largely different from the pre-training corpus may lead to catastrophic forgetting (Geva et al., 2020). To explore the importance of performing adapter-tuning on MSAT, we create a variant of our method in which we perform full fine-tuning on MSAT. We compare this variant with our models in Figure 6. It can be observed that both full fine-tuning and adapter-tuning can achieve good performance on MSAT, but adapter-tuning outperforms fine-tuning on all downstream MWP datasets, which demonstrates the benefits of performing adapter-tuning on MSAT.

5 Related Work

In this work, we focus on improving moderate-sized LM's MWP performance by injecting multi-step reasoning ability. Hence, our work closely relates to both reasoning ability injection (Geva et al., 2020; Pi et al., 2022) and MWP solving (Xie and Sun, 2019; Patel et al., 2021; Jie et al., 2022).

Reasoning skills injection This technique refers to continually pre-training LMs on certain intentionally-crafted tasks to enhance their reasoning abilities. GenBERT (Geva et al., 2020) pre-trains LMs on templated-based synthetic data to inject numerical skills into the LMs. PoET (Pi et al., 2022) improves LMs' reasoning ability by pre-training them on tabular data towards imitating program executors. Both methods involve training LMs to produce numerical answers directly, which can be unreliable (Razeghi et al., 2022). Our work focuses on injecting into LMs the capability for solving complex arithmetic problems step-by-step.

Solving MWP with specialized architectures One of the research lines of MWP solving focuses on designing specialized achiectures for math reasoning (Xie and Sun, 2019; Lan et al., 2021; Jie et al., 2022). For example, Lan et al. (2021) combines RoBERTa (Liu et al., 2019) with a Transformer (Vaswani et al., 2017) decoder, and Jie et al. (2022) augments encoder-only LMs with a directed acyclic graph decoder. One of the shortages of such models is the information loss caused by masking actual numbers in the questions with symbolic tokens (Wu et al., 2021). In this work, we propose to represent actual numbers with digit tokenization, and improve models' multi-step reasoning ability by pre-training them on a synthetic task MSAT.

6 Conclusion

We propose a novel synthetic pre-training task, MSAT, to incorporate LMs with multi-step reasoning skills that improve performance on MWP tasks. This pre-training task encourages LMs to generate intermediate reasoning steps instead of predicting final numerical answers directly. Our experiments show that the proposed method is effective in improving the moderate-sized LM's performance on MWP solving tasks.

Limitations

Limited number of operators considered Following previous methods (Lan et al., 2021), we only consider binary operators ($+$, $-$, $\times$, and $\div$). As we adopt a code-style output format, it is possible to introduce other non-binary operators supported by the Python interpreter, e.g., `sum()` and `max()`. However, obtaining labeled data with such operators may require laborious efforts. We believe it is an interesting research question on exploring how to teach models to solve practical questions e.g., math word problems, by writing code in a low-resource setting (Jie and Lu, 2023).

Limited performance due to greedy decoding All the results we report in this work are produced via greedy decoding. A recent work (Wang et al., 2023) reports that making large LMs generate multiple answers and selecting the answer with the most votes can boost performance by a large margin. However, performing beam search for symbolic neural reasoners, e.g., DeductReasoner, can be challenging in that searching space increases exponentially with the number of variables in the question (Jie et al., 2022). Designing effective beam search strategies for symbolic neural reasoners is a promising direction.

Acknowledgements

We would like to thank the anonymous reviewers, our meta-reviewer, and senior area chairs for their insightful comments and support with this work. We would also like to thank members of our StatNLP research group for helpful discussions. This research/project is supported by the National Research Foundation Singapore and DSO National Laboratories under the AI Singapore Program (AISG Award No: AISG2-RP-2020-016), and Ministry of Education, Singapore, under its Academic Research Fund (AcRF) Tier 2 Programme (MOE AcRF Tier 2 Award No: MOE-T2EP20122-0011)

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2022. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.

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*.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. 2022. Pal: Program-aided language models. *arXiv preprint arXiv:2211.10435*.

Mor Geva, Ankit Gupta, and Jonathan Berant. 2020. Injecting numerical reasoning skills into language models. In *Proceedings of ACL*.

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *Proceedings of ICML*.

Zhanming Jie, Jierui Li, and Wei Lu. 2022. Learning to reason deductively: Math word problem solving as complex relation extraction. In *Proceedings of ACL*.

Zhanming Jie and Wei Lu. 2023. Leveraging training data in few-shot prompting for numerical reasoning. In *Findings of ACL*.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. Mawps: A math word problem repository. In *Proceedings of NAACL*.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2021. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. *arXiv preprint arXiv:2109.00799*.

Zhenwen Liang, Jipeng Zhang, Lei Wang, Wei Qin, Yunshi Lan, Jie Shao, and Xiangliang Zhang. 2022. MWP-BERT: Numeracy-augmented pre-training for math word problem solving. In *Findings of NAACL*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *Proceedings of ICLR*.

Shen-yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2020. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of ACL*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of NeurIPS*.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are NLP models really able to solve simple math word problems? In *Proceedings of NAACL*.

Xinyu Pi, Qian Liu, Bei Chen, Morteza Ziyadi, Zeqi Lin, Yan Gao, Qiang Fu, Jian-Guang Lou, and Weizhu Chen. 2022. Reasoning like program executors. In *Proceedings of EMNLP*.

Yasaman Razeghi, Robert L Logan IV, Matt Gardner, and Sameer Singh. 2022. Impact of pretraining term frequencies on few-shot reasoning. In *Proceedings of ICML*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of ACL*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NeurIPS*.

Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. 2019. Do NLP models know numbers? probing numeracy in embeddings. In *Proceedings of EMNLP-IJCNLP*.

Tianduo Wang and Wei Lu. 2022. Differentiable data augmentation for contrastive sentence representation learning. In *Proceedings of EMNLP*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *Proceedings of ICLR*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. In *Proceedings of NeurIPS*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, et al. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of EMNLP*.

Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuanjing Huang. 2021. Math word problem solving with explicit numerical values. In *Proceedings of ACL-IJCNLP*.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *Proceedings of IJCAI*.

A Additional information about datasets

In this section, we provide additional details about the datasets that we used in the experiments.

A.1 Construction of MSAT

The proposed MSAT is a synthetic Seq2Seq task where the inputs describe arithmetic questions and outputs are the solutions represented by a code-style multi-step reasoning format. Both inputs and outputs of MSAT can be generated automatically. To construct an example of MSAT, we first generate the input sequence and then produce the output solution accordingly. In all, we generate 85,000 examples and split them into 80,000 and 5,000 for training and evaluation respectively.

Input sequence construction We start by preparing a set of equation templates and each equation template contains no more than 3 binary operators ($+$, $-$, $\times$, and $\div$). By enumerating the possible combinations of operators, we obtain $4+4^2+4^3 = 84$ equation templates in total. The first step to construct an input arithmetic question is to instantiate an equation from an equation template. For example, given an equation template "`<Num0> + <Num1> = <Num2>`", we assign each variable a value that makes the equality hold and a variable name selected from the capitalized letters. The numbers in the questions are sampled from `0` to `10,000`. The last step is to randomly pick a variable as the question variable. Therefore, the resulting input arithmetic question may look like: "`A=1. C=3. A+B=C. B?`"

Output sequence construction Given an equation and a question variable, the output is first constructed as a math expression leading to the value of the question variable. Notice that an equation can be represented as a binary tree where the variables are the terminal nodes and operators are the non-terminal nodes. Hence, the output can be produced by a "tree inversion" algorithm (see Figure 7) from an equation and a question variable.

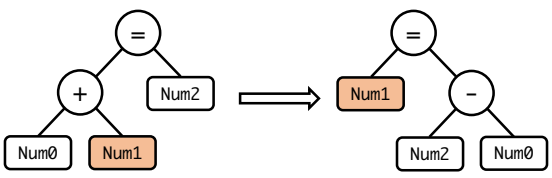

Figure 7: An illustration of the "tree inversion" algorithm that produces an output expression from an arithmetic question. The question variable is highlighted.

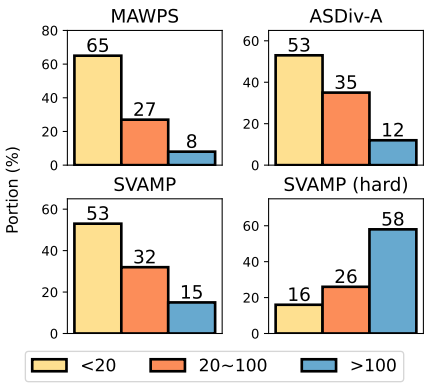

Figure 8: Number distribution for different datasets.

A.2 Existing datasets

MAWPS (Koncel-Kedziorski et al., 2016) It is a popular benchmark dataset for math word problems. We use the five-fold split provided by Lan et al. (2021) for evaluation.

ASDiv-A (Miao et al., 2020) This is an English math word problem task containing various linguistic patterns and problem categories. We obtain the data and five-fold split from Patel et al. (2021).

SVAMP (Patel et al., 2021) It is a challenge set created for MWP model robustness evaluation. The examples in SVAMP are from ASDiv-A with deliberately designed variations. Such variations include: changing questions, adding irrelevant information, etc. Following the evaluation protocol suggested by Patel et al. (2021), we train our models over 3,138 training examples from a combination of MAWPS and ASDiv-A.

A.3 SVAMP (hard)

SVAMP (hard) is used to evaluate models' extrapolation ability on the out-of-distribution numbers. We sample numbers from from `10` to `10,000`, a significantly different range from the original one, to replace the original numbers in SVAMP. Every question in SVAMP (hard) corresponds to a question in SVAMP. Although it is straightforward to sample a large number and use it to replace the numbers, we expect the created questions to make sense. We achieve this by making sure the new numerical results have the same type as the original ones. For example, if the original numerical answer is a positive integer, then we make sure the new numerical answer is also a positive integer. We compare the number distribution of existing MWP datasets and SVAMP (hard) in Figure 8.

B Implementation details

Our method is implemented in Python 3.8 with HuggingFace's Transformers (Wolf et al., 2020) and PyTorch (Paszke et al., 2019) libraries. All experiments can be conducted on one NVIDIA RTX 6000 GPU with 22 GB memory.

B.1 Backbone Model implementation

For our MSAT-ROBERTAGEN and MSAT-DEDUCTREASONER, we build the backbone models following the implementation provided by Lan et al. (2021) and Jie et al. (2022) respectively. The encoders for both models are initialized with the pre-trained weights of $\text{RoBERTa}_{\text{base}}$. The adapter modules (Houlsby et al., 2019) are added to each layer of the encoders with a bottleneck dimension of 64. More details about the mdoel architectures are provided in Table 3.

	ROBERTAGEN	DEDUCTREASONER
# Params.	139.71 M	142.40 M
# Attention heads	8	-
Hidden dim.	768	768
Feedforward dim.	1024	768
# Layers	2	-
Activation	ReLU	ReLU
Dropout	0.1	0.1
Label smoothing	0.05	-
# Constants	17	17

Table 3: Hyperparameters of model architectures.

B.2 Training configurations

	PRE-TRAINING	FINE-TUNING
Batch size	32	16
Max steps	10,000	50,000
Optimizer	AdamW (Loshchilov and Hutter, 2019)	
Weight decay	0.01	0.01
Max grad norm	0.1	1.0
Learning rate	3e-5	1e-5
LR scheduler	Linear	Linear

Table 4: Pre-training and fine-tuning hyperparameters.

ACL 2023 Responsible NLP Checklist

A For every submission:

☑ A1. Did you describe the limitations of your work?
Limitations

☑ A2. Did you discuss any potential risks of your work?
Limitations

☑ A3. Do the abstract and introduction summarize the paper's main claims?
1

☒ A4. Have you used AI writing assistants when working on this paper?
Left blank.

B ☑ Did you use or create scientific artifacts?

2, 3

☑ B1. Did you cite the creators of artifacts you used?
3, Appendix

☑ B2. Did you discuss the license or terms for use and / or distribution of any artifacts?
Appendix

☑ B3. Did you discuss if your use of existing artifact(s) was consistent with their intended use, provided that it was specified? For the artifacts you create, do you specify intended use and whether that is compatible with the original access conditions (in particular, derivatives of data accessed for research purposes should not be used outside of research contexts)?
Appendix

☒ B4. Did you discuss the steps taken to check whether the data that was collected / used contains any information that names or uniquely identifies individual people or offensive content, and the steps taken to protect / anonymize it?
We mainly focus on dealing with mathematical problems and in this work.

☑ B5. Did you provide documentation of the artifacts, e.g., coverage of domains, languages, and linguistic phenomena, demographic groups represented, etc.?
Appendix

☑ B6. Did you report relevant statistics like the number of examples, details of train / test / dev splits, etc. for the data that you used / created? Even for commonly-used benchmark datasets, include the number of examples in train / validation / test splits, as these provide necessary context for a reader to understand experimental results. For example, small differences in accuracy on large test sets may be significant, while on small test sets they may not be.
Table 2, Appendix

C ☑ Did you run computational experiments?

3

☑ C1. Did you report the number of parameters in the models used, the total computational budget (e.g., GPU hours), and computing infrastructure used?
Appendix

The Responsible NLP Checklist used at ACL 2023 is adopted from NAACL 2022, with the addition of a question on AI writing assistance.

☑ C2. Did you discuss the experimental setup, including hyperparameter search and best-found hyperparameter values?
Appendix

☑ C3. Did you report descriptive statistics about your results (e.g., error bars around results, summary statistics from sets of experiments), and is it transparent whether you are reporting the max, mean, etc. or just a single run?
3, 4

☑ C4. If you used existing packages (e.g., for preprocessing, for normalization, or for evaluation), did you report the implementation, model, and parameter settings used (e.g., NLTK, Spacy, ROUGE, etc.)?
Appendix

D ☒ Did you use human annotators (e.g., crowdworkers) or research with human participants?

Left blank.

☐ D1. Did you report the full text of instructions given to participants, including e.g., screenshots, disclaimers of any risks to participants or annotators, etc.?
No response.

☐ D2. Did you report information about how you recruited (e.g., crowdsourcing platform, students) and paid participants, and discuss if such payment is adequate given the participants' demographic (e.g., country of residence)?
No response.

☐ D3. Did you discuss whether and how consent was obtained from people whose data you're using/curating? For example, if you collected data via crowdsourcing, did your instructions to crowdworkers explain how the data would be used?
No response.

☐ D4. Was the data collection protocol approved (or determined exempt) by an ethics review board?
No response.

☐ D5. Did you report the basic demographic and geographic characteristics of the annotator population that is the source of the data?
No response.