A Systematic Survey on Math Word Problem Solvers Based on Large Language Models

Yicong Liang[1] and Debby D. Wang[1]

[1]School of Science and Technology, Hong Kong Metropolitan University, Hong Kong SAR
yliang@hkmu.edu.hk

***Abstract*— Nowadays, there is growing interest in improving the reasoning capabilities of large language models (LLMs), represented by designing an LLM-based solver for math word problems (MWPs). This survey provides a comprehensive overview of recent LLM- based methods that aim to solve MWPs. In particular, we first introduce some preliminaries on MWPs and their connection with LLMs. Then, we examine the capabilities of each reviewed method by analyzing its architecture and prompting technique. Finally, we discuss the limitations of LLM-based MWP solvers and provide potential directions for future research.**

***Index Terms*— Large Language Model, Math Word Problem, Prompt Engineering, Mathematical Reasoning**

I. INTRODUCTION

The development of automatic AI systems for math word problems has a long-standing history, dating back to the 1960s [1], [2]. A tool that can generate step-by-step solutions to math word problems has the potential to offer personalized guidance for students and assist educators in curriculum development. However, automatically solving math word problems (MWPs) is challenging, as the solver needs to combine arithmetic skills with commonsense reasoning.

Before pre-trained large language models revolutionized most NLP tasks, some small-scaled models with handcrafted neural networks were proposed to solve MWPs. For example, previous studies [3], [4] consider MWP as a generation task and usually leverage LSTM-based sequence-to-sequence models to learn the mapping from source sequences (i.e., question texts) to target sequences (i.e., math expressions). These neural MWP solvers without using pretrained language models (PLMs) have been surveyed in [5]. However, the main disadvantage of previously proposed neural solvers is that they must be trained from scratch for different MWP datasets, making them unscalable for other downstream tasks.

In recent years, language models (LMs) have reshaped the landscape of the NLP field and demonstrated impressive performance across diverse downstream tasks [6]. The pretrained models, such as BERT [7], RoBERTa [8], DeBERTa [9], BART [10] and GPT [11], have learned world knowledge by parsing a vast amount of texts, which benefits the question answering (QA) task consequently (e.g. commonsense QA [12], answering math word problems [6] and assisting with theorem proving [13]).

Large language models (LLMs) have an improved performance on diverse NLP downstream tasks. However, scaling up the size of language models alone has not demonstrated their effectiveness on some mathematical reasoning tasks, such as MWPs and theorem proving [14]. The dataset GSM8K [15] containing step-by-step reasoning is proposed to evaluate the LLM's reasoning ability by checking the effectiveness of its generated natural language solutions. Cobbe et al. [15] proposed to finetune GPT3 [6] on GSM8K to help the language model generate multi-step rationales and train a neural component to verify the correctness of the model-generated solution. This generate-and-verify framework could enhance the model's capacity of generating accurate answers.

Recently, some studies [16], [17], [18] have reported that language models can demonstrate the emergent ability of performing complex multi-step reasoning tasks when they are large enough (e.g., over 100B parameters). In particular, the breakthrough method, chain-of-thought (CoT) [16] prompting strategy, can unlock the reasoning ability of LLMs when provided with a few examples without any parameter update. A series of intermediate natural language reasoning steps is generated before giving the final answer.

II. MATH WORD PROBLEM

A. *Difference between MWP and other QA task*

Machine reading comprehension is one of the central tasks in natural language understanding, especially for the task of question answering (QA). In the context of knowledge-based QA tasks, the system leverages knowledge-aware methods to respond with the corresponding answer, e.g., knowledge triple retrieval [19].

The MWP task can be considered as a special case of QA task. However, the MWP task is different from the traditional text-based QA tasks with the following challenges: (1) A MWP needs to parse the human-readable words into machine-understandable mathematical logic to perform quantitative reasoning; (2) A MWP requires complex reasoning scenarios, and MWP solvers need to be capable of mathematical

calculation and mathematical reasoning; (3) Unlike natural language understanding, MWPs usually have a single correct numeric answer, which increases the difficulty for the solver to accurately generate the solution.

Math reasoning tasks include arithmetic problems and math word problems. Arithmetic problems mainly correspond to mathematical calculation consisting of arithmetic representation and arithmetic calculation [20]. However, there exist some differences between arithmetic problems and math word problems. In particular, arithmetic problems focus on pure mathematical operations and numerical manipulation, where the input problems rarely contain semantic textual elements.

On the other hand, math word problems are usually presented as verbal descriptions instead of explicit mathematical equations in arithmetic problems. In addition, MWP is related to the task of mathematical reasoning, where the solver can interpret and generate step-by-step natural text before giving the final answer.

Traditional span-based methods of extractive question-answering tasks (e.g., SQuAD [21]) cannot be directly applied to solve MWPs since the answer is usually the result of some computation and is generally not a span in the question or context. A MWP solver generates a numerical math expression and feeds this expression to an external symbolic calculator to obtain the final answer.

B. Preliminary

The math word problems include the following main components: (1) A textual description related to the math problem; (2) A set of known quantities mentioned in the problem text; (3) An unknown quantity whose value needs to be solved. In addition, MWP can be further grouped into the different levels: (1) one-unknown variable to be solved, or multi-unknown variables; (2) linear equations or non-linear equations. In this survey, the reviewed methods only aim to solve math problems solved by linear equations towards one unknown variable.

The solution to the problem denoted as A can be represented in the following format: (1) A single numerical value, e.g., GPT3 [6] leverages standard prompting to output the final answer for a given MWP question; (2) A mathematical expression, as an input to an external tool like Python calculator [22]; (3) a series of textual reasoning steps including the final numeric answer, e.g., chain-of-thought prompt [16] requires LLM to generate rationales before giving the final answer. Previous studies [23], [24] address to generate the solution to a math word problem as a mathematical expression, and these models try to map the problem text to a symbolic space with numbers and operators.

We introduce preliminaries of generating solutions to MWPs with LM prompting based on large language models, and the notations for modeling are listed in Table I. Mathematically, a math word problem is represented in the form of a text sequence $< w1, w2, \cdots, wn >$. There are some known quantities mentioned in the text and one unknown variable that the system needs to solve. How to extract the quantities is a preprocessing problem, and some works simply adopt the string pattern matching method [25] to recognize the numeric entries.

Based on the backbone of large language models, solving MWP can be transformed into text generation tasks. Therefore, the language model objective can be used in the MWP task for solution generation. In the following, we will introduce how to incorporate a pretrained language model to solve the MWP task.

Vanilla QA:

$$p(\mathcal{A}|\mathcal{Q}) = \prod_{i=1}^{|\mathcal{A}|} p(a_i|\mathcal{Q}, a_{<i}) \tag{1}$$

Given a textual question Q, the solution in terms of a single numeric answer can be generated by the LLM. In addition, a prefixed prompt will be added before generating the final answer, e.g., "The answer is " [6].

In-context learning:

$$p(\mathcal{A}|\mathcal{Q}, \mathcal{D}) = \prod_{i=1}^{|\mathcal{A}|} p(a_i|\mathcal{Q}, \mathcal{D}, a_{<i}) \tag{2}$$

TABLE I
NOTATIONS USED IN THIS PAPER

Notations	Descriptions
$\mathcal{Q}$	a question related to MWP
w	a token in the text sequence
$\mathcal{A}$	a textual solution to the question
$\mathcal{Y}$	a numerical answer of the question
c	one reasoning path
$\mathcal{D}$	A set of exemplars for in-context learning

To enhance the quality of the response from the LLM, in-context learning (ICL) is incorporated into the natural language processing (NLP) tasks. The ability of analogy with respect to ICL is embedded in LLMs, which can be learned from a few examples D in the context [26]. Each exemplar consists of one question and its corresponding solution. Meanwhile, a list of exemplars is concatenated to form the demonstration prompt, which is fed into the LLM as an augmented context for prediction on the testing question. The improvement by leveraging ICL relies on the training stage and inference stage [26], but this survey mainly focuses on the inference stage for the MWP task.

Reasoning-enhanced QA:

$$p(\mathcal{A}|\mathcal{D}, \mathcal{Q}) = \sum_c p(\mathcal{A}, c|\mathcal{D}, \mathcal{Q}) = \sum_c p(\mathcal{A}|\mathcal{D}, \mathcal{Q}, c)p(c|\mathcal{D}, \mathcal{Q}) \tag{3}$$

Scaling up the size of LLMs can elicit some reasoning abilities that can improve the accuracy of MWP solvers by generating a reasoning path c before giving the final numeric answer. In particular, reasoning abilities can be unlocked by prompting engineering [16], which generates a solution to MWP in the step-by-step format.

III. Surveyed Methods

The principles for selecting reviewed papers in this work are as follows: (1) The proposed methods are based on the pretrained language models (e.g., GPT-3 [6], Codex [28]) as their fundamental backbone, as shown in Table II; (2) The tasks aim to solve math word problems, e.g., the datasets in their experiments are related to MWP; (3) The reviewed articles are published in top venues or with a citation count over 50. The surveyed models are listed in Table III.

This survey first analyzes the compared models to determine whether the parameters of their foundation language models are updated. Accordingly, the reviewed methods are grouped into two mainstreams: finetune-based and prompt-based methods.

A. Finetune-based Models

Numerical reasoning skills are challenging when the LMs are only trained on the objective of the vanilla language model. Geva et al. [23] proposed a multi-task training strategy to inject mathematical reasoning skills into PLMs. Notably, the proposed framework GenBERT [23] incorporates automatic data generation in the pretraining task and is trained on textual data in the form of question-passage pairs. GenBERT is a BERT-based model for generating arbitrary output tokens. It can handle the extractive QA task, where the answer is a text span among the question or context and contains a generative head that can output the numeric answer.

The MWP solver trained on the language model objective may make mistakes in generating mathematical expressions. The models trained to learn the mapping from problem text to math expressions may have unsatisfactory performance because it is difficult for them to learn to distinguish between ground-truth and predictive expressions with minor mistakes [24]. To handle this limitation, Shen et al. [24] additionally introduce a ranker to explicitly train the model to distinguish between accurate and inaccurate expressions. The proposed model Generate & Rank [24] within transformer-based encoder-decoder architecture BART [10] first generates expression candidates and then ranks the candidates to make the final prediction.

It is challenging for autoregressive models to accomplish mathematical reasoning tasks since they cannot correct their errors when a generation is generated. Similar to the idea in [24] that the solutions generated need to be evaluated, Cobbe et al. [15] proposed training verifiers to check the correctness of the candidate solutions. In addition, the proposed model [15] developed reasoning steps in natural language such that the produced solutions were more interpretable by humans instead of producing a math expression in [24]. The generator based on GPT3 [6] is finetuned on a curated math dataset GSM8K [15] for generating rationales to form the full natural language solution. The experimental results suggest that it is essential to allow the MWP solver to generate a natural language solution

TABLE II
Pretrained Language Models used in math word problem.

Pretrained LM	Size
GPT2 [11]	1.5B
GPT3 [6]	175B
GPT-J [27]	6B
Codex [28]	12B
PaLM [17]	540B
BERT [7]	340M
BART [10]	406M
DeBERTa [9]	304M

TABLE III
Surveyed papers.

Model	Venue	Language model	Parameters update	External component
GPT3 [6]	Arxiv	GPT3	No	Nill
CoT [16]	NeurIPS	GPT3	No	Nill
Zeroshot-CoT [29]	NeurIPS	GPT3	No	Nill
Auto-CoT [30]	ICLR	GPT3	No	Sentence-BERT
PAL [31]	ICML	Codex	No	Nill
PoT [32]	TMLR	Codex	No	SymPy
Self-consistency [18]	ICLR	PaLM	No	Nill
Least-to-most [33]	ICLR	GPT3	No	Nill
MathPrompter [25]	ACL	GPT3	No	Calculator
DECLARATIVE [22]	NeurIPS	Codex	No	SymPy
Plan-and-Solve [34]	ACL	GPT3	No	Nill
Complexity [35]	ICLR	GPT3	No	Nill
GenBERT [23]	ACL	BERT	Yes	Nill
Generate & Rank [24]	EMNLP	BART	Yes	Trained Ranker
Verifier [15]	Arxiv	GPT3	Yes	Trained Verifer
STaR [36]	NeurIPS	GPT-J	Yes	Nill
DIVERSE [37]	ACL	DeBERTa	Yes	Trained Verifier
CoRe [38]	ACL	GPT-J	Yes	Trained Verifier

before giving the final answer, and the performance dropped dramatically if directly outputting a final numeric answer without any intermediate steps [15].

Generating intermediate reasoning steps improves LM performance on complex tasks like MWP, but fine-tuning the generator requires massive training examples with rationales. To address this limitation, Zelikman et al. [36] developed a self-taught reasoner (STaR) by iteratively bootstrapping the reasoning ability to generate rationales. In particular, starting with a small prompt dataset that contains intermediate rationales, StaR adopts in-context prompting to annotate each example in the large dataset for further finetuning the language model [36]. To improve the robustness, rationalization is applied [36] to reason backward for problems that the model fails to solve, i.e., given the correct answer as a hint, let the model generate the rationale accordingly. Finally, the finetuning process will be repeated on the enlarged dataset with previously generated rationales.

Few-shot learning for solving MWP is a challenging task that requires the LMs to elicit intermediate reasoning steps. Even equipped with the LLMs like GPT3 (175B) [6] and PaLM (540B) [17], the reasoning abilities are still limited. To further improve the reasoning capacity of PLMs, Li et al. [37] designed a diverse verifier to aggregate different sampled reasoning paths to solve the MWP. Specifically, the proposed model DIVERSE [37] first samples different demonstration exemplars for constructing diverse prompts and then feeds them OpenAI PLMs (e.g., text-davinci-002) to generate various reasoning paths. Second, DIVERSE trains a step-aware verifier by finetuning DeBERTa [9] to score the quality of each path and uses a weighted voting mechanism [37] to obtain the final answer.

Directly prompting PLMs to solve MWPs often does not yield satisfactory results, as the generation process lacks the level of supervision and adaptivity that humans possess. Zhu et al. [38] argued that the human-like reasoning framework could be modeled using a dual system approach, with a generator for immediate reactions and a verifier for more nuanced reasoning. Their proposed model CoRe [38] leverages the direct interaction between the generator and verifier to improve the generalization ability of LLMs. First, the verifier can provide reliable feedback to supervise the generator for rationale generation. Second, the verifier leverages Monte Carlo Tree Search (MCTS) [39] to score the tokens of reasoning paths produced by the generator. Finally, the proposed strategy of self-thinking can provide informative self-produced data to teach the generator and verifier.

B. Prompt-based Models

Based on the transformer-based framework with self-attention technique [40], numerous downstream tasks have been transferred into text generation problems by following the "pretrain, prompt and predict" paradigm. In this paradigm, instead of finetuning the PLMs to adapt to a new task, solving text-based tasks is reformulated to the original language model pretraining with the help of an appropriate textual prompt [41]. Inspired by the idea that humans can perform a new language task from a few demonstration examples or simple task instructions, Brown et al. [6] leverage in-context learning within few-shot settings and prompt the language model GPT3 to solve the target tasks by providing some task examples as additional context during the inference stage without any gradient update [6]. The significant advancement of this setting is that the users can prompt the model with a few input-output demonstration exemplars instead of finetuning a separate LM checkpoint for each new task. The experimental results [6] show that scaling up LMs greatly improves task-agnostic, few-shot performance.

The critical limitation of a finetune-based MWP solver is that it is expensive to collect a large set of training data with high-quality rationales. The traditional few-shot prompting technique used in [6] has been successful for a series of simple QA tasks but performs poorly on tasks requiring reasoning ability even with increasing LM scale [42]. Wei et al. proposed chain-of-thought prompting [16] to elicit the LLMs' reasoning ability for complex tasks, e.g., commonsense reasoning and arithmetic. This approach involves a sequence of intermediate reasoning steps expressed in natural language, leading to the final output. CoT prompting can be considered as a special type of in-context learning where each exemplar includes the reasoning thought process instead of just a single final answer. The experimental results suggest that CoT prompting improves performance by allowing the sequential reasoning steps embodied in the generation [16].

Pretrained large language models are well-known as excellent few-shot learners with task-specific exemplars and can generate complex rationales via step-by-step solution examples. Kojima et al. [29] show that PLMs are also decent zero-shot learners by leveraging a simple but effective prompt "Let's think step by step" before giving the final answer. The proposed model Zero-shot-CoT [29] does not require handcrafted task-specific exemplars and outperforms zero-shot LLMs on diverse downstream reasoning tasks, e.g., arithmetic math word problems.

According to the prompting design regarding the number of exemplars, CoT prompting can be classified into two significant paradigms: few-shot CoT and zero-shot CoT. In general, few-shot CoT with task-specific rationales demonstrations outperforms zero-shot CoT in most cases. However, manually constructing exemplars with step-by-step reasoning chains for a specific task or even each testing question is nontrivial. Randomly or heuristically selecting in-context examples in CoT prompting may have a high risk of unstable performance in reasoning tasks. To address this limitation, Zhang et al. [30] proposed automatically constructing demonstrations with questions and reasoning chains instead of handcrafting in-

context exemplars. Prompting LLM to generate reasoning chains for each exemplar directly often comes with mistakes. The proposed solver Auto-CoT [30] increases the diversity of demonstration questions to help the LLM lower the mistakes in generating reasoning chains. Particularly, the questions are clustered based on their representation obtained by Sentence-BERT [43], and a representative question from each cluster is selected.

LLMs have demonstrated their effectiveness in diverse downstream reasoning tasks by using CoT and in-context learning. However, LLMs often make arithmetic mistakes in the solutions, even if the generated rationales are logically correct. Program-Aided Language model (PAL) [31] addressed the underlying problem that LLMs often struggle with performing arithmetic operations and proposes to use an external tool (i.e., Python program) to deal with the calculation work. PAL leverages the LLM (i.e., Codex [28]) to read a testing problem and then generates a program [31] instead of natural language rationales [16], [29], [30] as intermediate reasoning steps to assure the calculation accuracy by using the Python interpreter.

Previous methods [16], [30], [29] output the chain-of-thought reasoning steps in natural language and let the PLMs do reasoning and computation jobs simultaneously. Chen et al. [32] argued that LMs could express reasoning steps as programs in a few lines of code, and the external language interpreter (e.g., Sympy [44]) can finish the computation work. The proposed model Program of Thoughts (PoT) decouples complex computation from reasoning and language under- standing. To compare PoT with vanilla CoT [16] and Zero-shot CoT [29], PoT leverages the program of thoughts in each exemplar and does not require an extra step to extract the answer from the reasoning steps since the generated program can be executed by the interpreter to return the final answer. Intuitively, there exist multiple different ways of solution leading to its unique correct answer for a complex reasoning problem. Inspired by that, Wang et al. [18] designed a novel decoding strategy, self-consistency, to sample multiple solutions instead of greedily decoding only one as used in vanilla CoT [16]. In particular, self-consistency first samples a diverse set of reasoning paths and then aggregates them by using a majority voting scheme to select the most consistent answer. Compared to the sample-and-rank methods [15], [37], self-consistency does not require training an additional component or finetuning the backbone LM to verify the correctness probability of the generated solution.

In-context learning and CoT have been widely adopted in prompting engineering to help LLMs solve various reasoning tasks. However, LLMs will have poor performance in the case when the examples in the ICL demonstration prompt are easier than the testing problem[1]. To address this issue, Zhou et al. [33] introduced a new prompting strategy, Least-to-most, to help LLMs improve easy-to-hard generalization. Specifically, least-to-most decomposes a complex problem into a series of simpler subproblems and then solves them sequentially. In both stages, the decomposition and subproblem solving are accomplished by the LM without any parameter update. In addition, Least-to-most can be incorporated with ICL and CoT to further enhance the performance in reasoning tasks.

In the zero-shot setting, Zero-shot CoT [29] has demonstrated its remarkable performance in mathematical reasoning tasks. However, Imani et al. [25] pointed out two limitations in CoT-based prompting methods: (1) lack of checking the validity of reasoning steps; (2) lack of confidence in the predictions. The proposed model MathPrompter [25] first transforms the question into an Algebraic template with value-variable mapping and then sends it to the LLM to generate two different solutions in Algebraic and program ways for cross-checking. Afterward, MathPrompter evaluates the two generated solutions using multiple randomly selected values to check consensus among the answers. Repeat the above steps several times to extract the most frequent value observed for the answer.

Some methods [31], [32] offload the calculation to a language interpreter to eliminate the arithmetic error that LLMs often make in generating the solutions. However, these program-based models favor those problems with simple procedures and are less effective for problems requiring declarative reasoning. He-Yueya et al. [22] proposed the approach DECLARATIVE to perform mathematical declarations. DECLARATIVE prompts the LLM to formalize MWPs as a set of variables and equations incrementally and solves the equations by passing them to Sympy [44].

Zero-shot CoT can lower the effort to manually handcrafted step-by-step reasoning exemplars, but it still suffers from missing-step and calculation errors. To address the missing-step limitation, Wang et al. [34] manually craft the Plan-and-solve prompt to guide the LM to devise a plan to decompose the entire task into several subtasks and solve each subtask sequentially. To address the arithmetic limitation, the researchers add a detailed instruction prompt [34] to ask the LM to pay more attention to variables and calculation results.

Demonstration exemplars with reasoning steps can improve the prediction performance for new inputs. However, different exemplars may influence the testing question differently when making inferences. Which reasoning exemplars make the most the most effective in-context learning prompts becomes a critical question. Fu et al. [35] proposed complexity-based prompting, a novel example selection scheme, for CoT multi-step reasoning. Specifically, select complex instances[2] with CoT reasoning steps in the in-context learning prompt before the testing question. The experimental results suggest that selecting complex questions as in-context learning exemplars improves the performance on math word reasoning tasks [35].

[1] A MWP can be simply measured its difficulty by the number of solving steps [33].

[2] The complexity indicator is the number of steps to solve the question.

IV. Challenges and Future Directions

When the large language models (e.g. GPT [45] and LLaMA [46]) are released to the public, the trends show that fewer models will be proposed for outputting math expression only, and more research will focus on leveraging the LLMs directly to generate a natural language solution for MWPs, including the rationales and numeric final answer. The possible reason may be that, due to the difference between natural language text sequences and mathematical expressions, one minor mistake will change the semantics and lead to an incorrect answer, whereas natural language generation is more robust to these tiny mistakes [24]. In addition, the performance of MWP solvers for generating mathematical expressions will only degrade quickly when the expression gets longer [4].

Generating stepwise rationales can enhance the performance of language models on complex reasoning tasks. However, inducing the rationale generation from LMs requires constructing a large amount of data containing reasoning steps. It is very expensive to construct such datasets manually to finetune the LLMs. To the best of our knowledge, only GSM8K [15] and MATH [14] provide full step-by-step solutions to finetune the LLMs, in order to generate the solution for a testing problem. In addition, the language model finetuned on one specific dataset may not generalize well on another MWP dataset.

Another line of method to elicit LLMs to generate reasoning steps automatically is to adopt some prompting strategies, including instructions, trigger sentences, and in-context learning. LLMs have shown promising performance in solving new reasoning problems by simply conditioning on a few demonstration examples (e.g., few-show learning). However, small variations in prompt configuration have been known to affect few-shot performance dramatically [35]. Handcrafting instructing prompts for different reasoning MWP datasets needs much expertise and annotation work. The order of exemplars listed in the demonstration, the complexity of problems provided in in-context learning, and the relation (e.g., similarity) between demonstration examples and the testing problem may affect the prediction performance.

Based on the challenges illustrated above, we will outline some future directions for improving the reasoning ability of MWP solvers. As a formula is not only a simple sequence of mathematical symbols but also has strong logical and se- mantic relation with its context [47], selecting the appropriate formulas is the key step to solve an MWP. Hence, training the representation of the formula is essential for the neural solver. In addition, it is interesting to check the relatedness between the formula and the supporting generated rationale in each step and in different steps by jointly training with the formula and its surrounding context, which could further improve the interpretability of the solver system for users.

Combining Chain-of-thought prompting with in-context learning has shown the efficiency in unlocking the reasoning capability of LLMs without any gradient update or finetuning models [16], [18], [30], [6]. There is still some work to be done to optimize the selection of demonstration examples. Given an exemplar base with a reasoning-step solution for each question, train a small-scaled neural module to select in-context exemplars automatically for the testing problem.

Although LLMs have decent performance in many NLP tasks, building an LLM-based education system is still challenging. Li et al. [48] argue that LLMs basically need to integrate five educational abilities to address students' concerns for their studies. Besides automatically solving MWPs, the solver can be constructed as a multi-functional education system. For example, given a problem that a student cannot solve by herself the first time, the system can generate other questions with similar principles for her to make more practice. On the other hand, instead of directly giving a complete solution to the question, it is more helpful to generate some hints based on the student's partial solution. The traditional MWP solver can be transferred to an LLM-based MWP assistant to meet students' various requirements.

V. Conclusion

Automatically generating high-quality and step-by-step solutions to math word problems has numerous applications in education. This paper presents an overview of the current state of knowledge on math word problems based on LLMs. We divided the reviewed models into two groups, namely finetune-based and prompt-based, and examined their fine- tuning framework and prompting strategies. Finally, we issued the limitations of existing LLM-based MWP solvers and highlighted the future directions worth working on. We hope this survey can highlight the current state of MWP research and provide some insight into future work in this direction.

Acknowledgment

The work described in this paper was supported by the Katie Shu Sui Pui Charitable Trust — Academic Publication Fellowship (Project Reference No. KSPF/2023/05).

References

[1] E. A. Feigenbaum, J. Feldman et al., *Computers and thought*. New York McGraw-Hill, 1963, vol. 37.

[2] E. Charniak, "Computer solution of calculus word problems," in *Proceedings of the 1st international joint conference on Artificial Intelligence*, 1969, pp. 303-316.

[3] J. Li, L. Wang, J. Zhang, Y. Wang, B. T. Dai, and D. Zhang, "Modeling intra-relation in math word problems with different functional multi-head attentions," in *Proceedings of the 57th annual meeting of the association for computational linguistics*, 2019, pp. 6162-6167.

[4] Z. Xie and S. Sun, "A goal-driven tree-structured neural model for math word problems." in *Ijcai*, 2019, pp. 5299-5305.

[5] D. Zhang, L. Wang, L. Zhang, B. T. Dai, and H. T. Shen, "The gap of semantic parsing: A survey on automatic math word problem solvers," *IEEE transactions on pattern analysis and machine intelligence*, vol. 42, no. 9, pp. 2287-2305, 2019.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877-1901, 2020.

[7] J. Devlin, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[8] Y. Liu, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[9] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[10] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[11] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[12] A. Talmor, J. Herzig, N. Lourie, and J. Berant, "Commonsenseqa: A question answering challenge targeting commonsense knowledge," *arXiv preprint arXiv:1811.00937*, 2018.

[13] Y. Wu, A. Q. Jiang, W. Li, M. Rabe, C. Staats, M. Jamnik, and Szegedy, "Autoformalization with large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32 353-32 368, 2022.

[14] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, Song, and J. Steinhardt, "Measuring mathematical problem solving with the math dataset," *arXiv preprint arXiv:2103.03874*, 2021.

[15] K. Cobbe, V. Kosaraju, M. Bavarian, M. Chen, H. Jun, L. Kaiser, M. Plappert, J. Tworek, J. Hilton, R. Nakano *et al.*, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[16] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, "Chain-of-thought prompting elicits reasoning in large language models," *Advances in neural information processing systems*, vol. 35, pp. 24 824-24 837, 2022.

[17] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann *et al.*, "Palm: Scal- ing language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1-113, 2023.

[18] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[19] X. Wang, T. Gao, Z. Zhu, Z. Zhang, Z. Liu, J. Li, and J. Tang, "Kepler: A unified model for knowledge embedding and pre-trained language representation," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 176-194, 2021.

[20] W. Liu, H. Hu, J. Zhou, Y. Ding, J. Li, J. Zeng, M. He, Q. Chen, B. Jiang, A. Zhou *et al.*, "Mathematical language models: A survey," *arXiv preprint arXiv:2312.07622*, 2023.

[21] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *arXiv preprint arXiv:1606.05250*, 2016.

[22] J. He-Yueya, G. Poesia, R. E. Wang, and N. D. Goodman, "Solving math word problems by combining language models with symbolic solvers," *arXiv preprint arXiv:2304.09102*, 2023.

[23] M. Geva, A. Gupta, and J. Berant, "Injecting numerical reasoning skills into language models," *arXiv preprint arXiv:2004.04487*, 2020.

[24] J. Shen, Y. Yin, L. Li, L. Shang, X. Jiang, M. Zhang, and Q. Liu, "Generate & rank: A multi-task framework for math word problems," *arXiv preprint arXiv:2109.03034*, 2021.

[25] S. Imani, L. Du, and H. Shrivastava, "Mathprompter: Mathematical reasoning using large language models," *arXiv preprint arXiv:2303.05398*, 2023.

[26] Q. Dong, L. Li, D. Dai, C. Zheng, Z. Wu, B. Chang, X. Sun, J. Xu, and Z. Sui, "A survey on in-context learning," *arXiv preprint arXiv:2301.00234*, 2022.

[27] B. Wang and A. Komatsuzaki, "GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model," https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

[28] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[29] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large language models are zero-shot reasoners," *Advances in neural information processing systems*, vol. 35, pp. 22 199-22 213, 2022.

[30] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic chain of thought prompting in large language models," *arXiv preprint arXiv:2210.03493*, 2022.

[31] L. Gao, A. Madaan, S. Zhou, U. Alon, P. Liu, Y. Yang, J. Callan, and G. Neubig, "Pal: Program-aided language models," in *International Conference on Machine Learning*. PMLR, 2023, pp. 10 764-10 799.

[32] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.

[33] D. Zhou, N. Scharli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. Le et al., "Least-to-most prompting enables complex reasoning in large language models," *arXiv preprint arXiv:2205.10625*, 2022.

[34] L. Wang, W. Xu, Y. Lan, Z. Hu, Y. Lan, R. K.-W. Lee, and E.-P. Lim, "Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models," *arXiv preprint arXiv:2305.04091*, 2023.

[35] Y. Fu, H. Peng, A. Sabharwal, P. Clark, and T. Khot, "Complexity- based prompting for multi-step reasoning," in *The Eleventh International Conference on Learning Representations*, 2022.

[36] E. Zelikman, Y. Wu, J. Mu, and N. Goodman, "Star: Bootstrapping reasoning with reasoning," *Advances in Neural Information Processing Systems*, vol. 35, pp. 15 476-15 488, 2022.

[37] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen, "Making large language models better reasoners with step-aware verifier," *arXiv preprint arXiv:2206.02336*, 2022.

[38] X. Zhu, J. Wang, L. Zhang, Y. Zhang, Y. Huang, R. Gan, J. Zhang, and Y. Yang, "Solving math word problems via cooperative reasoning induced language models," *arXiv preprint arXiv:2210.16257*, 2022.

[39] L. Kocsis and C. Szepesvari, "Bandit based montecarlo planning," in European conference on machine learning. Springer, 2006, pp. 282- 293.

[40] A. Vaswani, "Attention is all you need," *Advances in Neural Information Processing Systems*, 2017.

[41] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing," *ACM Computing Surveys*, vol. 55, no. 9, pp. 1-35, 2023.

[42] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young *et al.*, "Scaling language models: Methods, analysis & insights from training gopher," *arXiv preprint arXiv:2112.11446*, 2021.

[43] N. Reimers, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[44] A. Meurer, C. P. Smith, M. Paprocki, O. Certik, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh *et al.*, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, 2017.

[45] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730-27 744, 2022.

[46] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.A. Lachaux, T. Lacroix, B. Rozie`re, N. Goyal, E. Hambro, F. Azhar et al., "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[47] S. Peng, K. Yuan, L. Gao, and Z. Tang, "Mathbert: A pre- trained model for mathematical formula understanding," *arXiv preprint arXiv:2105.00377*, 2021.

[48] Q. Li, L. Fu, W. Zhang, X. Chen, J. Yu, W. Xia, W. Zhang, R. Tang, and Y. Yu, "Adapting large language models for education: Foundational capabilities, potentials, and challenges," *arXiv preprint arXiv:2401.08664*, 2023.