Specialized Mathematical Solving by a Step-By-Step Expression Chain Generation

Wenqi Zhang, Yongliang Shen, Guiyang Hou, Kuangyi Wang, and Weiming Lu

***Abstract*—Math Solving requires both semantic understanding and relation reasoning. Most current approaches treat it as a translation task from natural language to mathematical symbols, generating tokens one by one. However, token-level generation is usually vulnerable when confronted with diverse annotations and complex reasoning. We consider the equation is an ordered combination of multiple sub-expressions, and math reasoning should be performed on the sub-expression level rather than the token level. We treat sub-expression as a minimum generative unit and minimum reasoning node. At each step, candidate sub-expression nodes are generated in parallel, and the whole reasoning chain is deduced by combining multiple nodes in order. Besides, we can obtain multiple valid reasoning chains by sub-expression searching, further improving interpretability and precision. Experiments on multilingual datasets show our method significantly outperforms the baselines. Additionally, our approach is more stable and efficient when faced with the challenges of diverse annotation and complex reasoning with limited resources. Moreover, our approach can be seamlessly integrated with large language models (LLMs), enhancing LLM's mathematical reasoning capabilities at a minimal cost. Experiments show a synergistic collaboration between general-purpose LLM and our specialized model yields superior performance.**

***Index Terms*—Math word problem, math reasoning, expression generation, collaboration, step-by-step.**

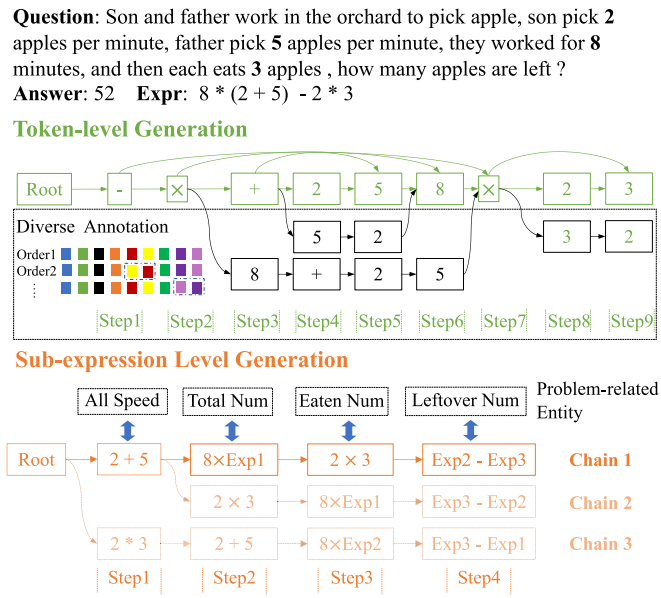

Fig. 1. Token-level method generates tokens one by one, affected by diverse labeling order. We generate a sub-expression in parallel, continuously deducing reasoning chains. Each sub-expression corresponds to a problem-related entity, useful for solving step by step.

I. Introduction

THE math word problem (MWP) aims to identify mathematical relations between quantity words from plain text and generate math equations. A great deal of previous work [1], [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] has focused on designing appropriate decoding structures, e.g., sequence-to-sequence (seq2seq) or sequence-to-tree (seq2tree). Seq2seq methods treat the MWP task as a translation task from human language to mathematical symbols and generate mathematical tokens one by one from left to right. Seq2tree methods view MWP as a binary tree generation task that recursively produces mathematical tokens in prefix order. However, most methods only focus on decoding at the token level, without considering sub-expression granularity.

There are two specific challenges in the token-level generation: **Diverse annotation** and **Complex reasoning**. Diverse annotation means a mathematical equation contains several sub-expressions (2+5, 2×3 in Fig. 1), which are quite diverse due to mathematical laws. These diverse equation cause inconsistent labeling order (e.g., $\times + 2\ 5\ 8$ and $\times 8 + 5\ 2$). However, token-level generation is sensitive to token order and vulnerable in such diverse annotations. In contrast, generating at the sub-expression level could reduce the dependence on token order and mitigate the effect of diversity.

Furthermore, complex math reasoning cannot be simply treated as a sequence generation, since it requires both understanding the relation between two numbers for a sub-expression and capturing the deductive logic between different sub-expressions. The token-level method does not explicitly model the relations between sub-expressions, but only predicts the token sequence in prefix order, which limits its reasoning capabilities. However, sub-expression is naturally suitable for

Manuscript received 3 November 2023; revised 13 April 2024; accepted 16 May 2024. Date of publication 10 June 2024; date of current version 21 June 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62376245, in part by the Key Research and Development Program of Zhejiang Province, China, under Grant 2024C03255, in part by the National Key Research and Development Project of China under Grant 2018AAA0101900, and in part by the MOE Engineering Research Center of Digital Library. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Boyang Li. *(Corresponding author: Weiming Lu.)*

The authors are with the College of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China (e-mail: zhangwenqi@zju.edu.cn; luwm@zju.edu.cn).

Digital Object Identifier 10.1109/TASLP.2024.3410028

reasoning step-by-step, since each sub-expression can correspond to a specific problem-related entity as in Fig. 1.

Except for token-level methods, [13], [14] adopted a Transformer [15] to generate an expression pointer, avoiding the generation at token-level. Besides, [8], [16], [17], [18] proposed a non-generative method using relation extraction or sequence labeling via M-tree coding. However, these models are not a general generative framework for fully utilizing the PLMs, and also difficult to explicitly model the interactions between different steps. In contrast, recent research on chain-of-thought [19], [20] has shown large PLMs can generate sub-expressions step-by-step like humans.

To handle these challenges, we revisit the process of human mathematical solving and find human reasoning is also performed on sub-expression as the basic unit. In this paper, we propose a sub-expression level reasoning chain generative method. As shown in Fig. 1, we treat sub-expression as a minimal generating unit and a basic reasoning step. A complete sub-expression is output in a single step as a reasoning node, and multiple ordered nodes form a valid sub-expression reasoning chain. The introduction of sub-expression generation alleviates the challenge of diverse annotations by extracting sub-expression in parallel, and better handles complex reasoning challenges via sequentially producing different sub-expressions.

To achieve parallel extraction within a step and sequential generation between different steps, we design a query decoder and a state converter for two processes. Firstly, the decoder extracts multiple sub-expressions in parallel from a set of queries, each detecting a mathematical relation. The generated sub-expression is added as a new node in the reasoning chain. Then state converter updates the problem representation and queries for the next step. Finally, all nodes in the chain are treated as a prefix prompt to guide the new generation, enabling the interaction between different reasoning steps. Furthermore, a sub-expression search algorithm can be plugged into our framework, deducing multiple valid reasoning chains simultaneously.

Besides, our method constructs the entire equation by iteratively generating the "expression token", a strategy that aligns closely with how current large language models (LLMs) address reasoning problems in a step-by-step manner. Consequently, we devise a method for collaborative synergy between specialized smaller models (ours) and large language models (e.g., GPT-4). Concurrently, [21] mimics the human problem-solving process by proposing an expansion of thought, which incrementally generates candidate expressions towards a comprehensive solution, sharing some similar insights with ours. However, our approach also incorporates multiple learnable queries for parallel extraction of sub-expressions and employs a sub-expression level beam search to generate multiple reasoning chains, enabling seamless collaboration with LLMs. Experiments indicate that through collaboration with smaller models, the specialized capabilities of LLMs can be further enhanced. Our contributions are threefold:

- We treat sub-expression as the minimum generative unit, and generate reasoning chains at the sub-expression level: parallel extraction for valid sub-expressions at one step and sequential generation for different steps.
- We design an iterative multi-query decoding manner for step-by-step reasoning. Multiple **math relation queries** are converted into candidate **sub-expression slots** under the guidance of the previous steps (**prefix prompt**), improving interpretability and accuracy.
- Extensive experiments on multiple datasets show our method significantly outperforms existing baselines and is more stable and efficient against diverse annotation and complex reasoning scenarios with limited resources.

II. Related Work

MWP tasks have been extensively studied for a long time. One of the earliest approaches was rule-based or templates-based methods [22], [23]. These methods [24], [25] were effective at simple or well-defined problems but struggled with complex and open-ended problems. With the development of deep learning [15], [26], the performance has been improved dramatically [1], [5], [7], [11], [27], [28]. We divide the methods into three categories: generative method, contrastive generation method, and discriminative method.

A. Generative Method for MWP

MWP was treated as a translation task from human language into mathematical language [27], [29]. Many seq2seq-based methods were proposed with an encoder-decoder framework. [30] introduced a group attention to enhance seq2seq performance. [31] utilized a transformer model for equation generation. Except for seq2seq methods, some researchers [1], [2], [3], [5] studied the decoding structures and proposed a tree-based decoder using prefix sequence and a graph encoder. [13] proposed an expression pointer generation manner for MWP. [32], [33], [34], [35] introduced mathematical knowledge to solve the complex math reasoning. [36] improved the interpretability by explicitly retrieving logical rules. These generative approaches significantly improved the accuracy, but most of them were token-level generation, ignoring the fact that mathematical equations are combined by multiple sub-expressions.

B. Contrastive Generation for MWP

The contrastive generation introduces a contrastive method in the generative framework, bringing considerable improvements. [6] improved accuracy by knowledge distillation between the teacher and student. [37] proposed a prototype learning through contrasting different equations. [38], [11] also used contrastive learning at the semantic and symbolic expression level. Besides, [8] aligned the representations of multiple traversal orders for consistency.

C. Discriminative Method for MWP

Recently, the discriminative approach has opened up a new perspective for MWP solving. [16] introduced a DAG structure to extract two quantities from bottom to top. [17] further treated the MWP task as an iterative relation extraction to discriminate the relation between quantities. [18] converted equation generation into a sequence labeling through an M-tree coding.

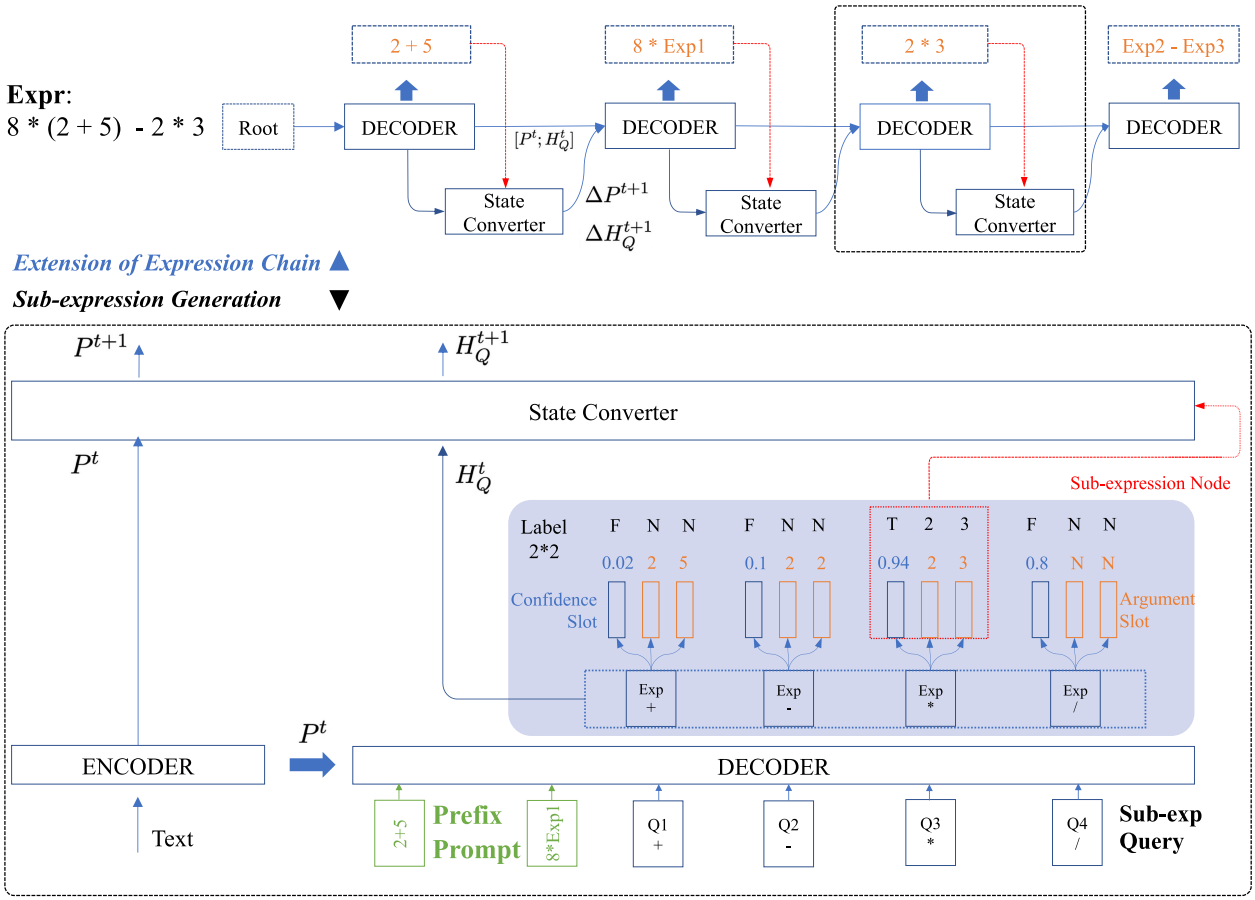

Fig. 2. Our method contains two processes: sub-expression generation (Bottom) and extension of the reasoning chain (Top). Firstly, the decoder converts multiple queries and prefix prompt into multiple sub-expressions, each containing two arguments and a confidence slot for a math relation. Then, we fill the slot with numbers and select the best sub-expression as a new node. Finally, the state converter updates the reasoning state for the next step.

These methods transform equation generation into a discriminative task and achieve impressive performance. In contrast, our approach is a versatile generative framework that fully utilizes PLMs to reason step by step and deduces multiple valid sub-expression chains like humans.

III. Methodology

A. Overview

The MWP is to generate the equation Y based on a problem description, which contains some quantity words. The vocabulary of Y contains two parts, operators and operands (number word). The first part is the mathematical operator, which contains several simple mathematical operations $V_{op} = \{+, -, \times, \div, \cdots\}$ and the second part, number words V_n are either from the original text or the pre-defined constants $\{1, \pi, \cdots\}$. The current token-level methods generate a token at each step from left to right ("8", "×", "(", "2", "+", $\cdots$) or following pre-order traversal ("-", "×", "8", "+", "2", $\cdots$). Our method generates multiple candidate sub-expressions directly at one step by extracting all arguments in parallel (Step 1 in Fig. 1,"$2 + 5$" and "2×3") and then selects the sub-expression with the highest confidence for the next iteration.

As shown in Fig. 2, our framework contains two processes: parallel extraction within a sub-expression and sequential generation for the sub-expression chain. To extract a sub-expression at one step, we convert the multiple queries into corresponding sub-expression slots in parallel (Section III-B), where each sub-expression is assigned to a specific mathematical relation. After filling in the slots with a specific number, we obtain a valid sub-expression for the current step. Then a state converter is introduced to update problem representation and queries for the reasoning chain extension (Section III-C). The two processes run alternately with a sub-expression search algorithm (Section III-D), deducing multiple valid reasoning chains simultaneously.

B. Sub-Expression Generation

We devise a problem encoder, a decoder and multiple learnable queries, where each query is to produce a sub-expression by the decoder. Every sub-expression contains two argument slots and a confidence slot for a particular mathematical relation.

Problem Encoder: Given a text description X with N_n number words, we adopt the pre-trained language model [26], [39] to obtain the contextualized problem representations $P = \{h_i\}_{i=1}^{N}$. We also obtain number vocabulary $V_n = \{h_{n_i}\}_{n_i=1}^{N_n}$

from P, which denotes the embeddings of the N_n number tokens in problem text. The problem and number representations will be updated at each step in the future.

Query Decoder: The decoder is designed for extracting all candidate sub-expression in parallel based on problem representation. Firstly, we adopt N_q learnable embeddings as query $Q = \{q_i\}_{i=1}^{N_q}$, where N_q means the number of the mathematical operators ($N_q = \text{Num}(V_{op})$). Every query is responsible for detecting a specific mathematical relation. The query decoder generates N_q candidate sub-expression embeddings from N_q input queries, each sub-expression also corresponds to a specific mathematical relation, e.g., as shown in Fig. 2, $exp^+ = (\underline{n_l}, \underline{n_r}, +)$ for addition relation detection and $(\underline{n_l}, \underline{n_r}, \times)$ for the multiplicative relation, where both $\underline{n_l}$ and $\underline{n_r}$ are numbers to be filled.

Specifically, the query decoder is the non-autoregressive model which contains a stack of identical Transformer layers. The query embeddings Q and the problem embeddings P are fed into the decoder as inputs. In each decoder layer, different relation queries interact with each other by self-attention mechanism, and the cross-attention mechanism fuses information from P into Q:

$$H_Q = \text{Decoder}(Q; P) \tag{1}$$

where $H_Q = \{h_Q^i\}_{i=1}^{Nq} \in \mathbb{R}^{N_q \times d}$ and d is the size of the hidden layer. We obtain N_q sub-expression embeddings (h_Q) for the current step, where each is assigned to a specific mathematical relation.

Argument Slot: As shown in Fig. 2, each sub-expression is filled with two operands for its corresponding math relation. So we design two argument slots for each sub-expression h_Q^{op} in H_Q:

$$s_l^{op} = \text{MLP}^l(h_Q^{op}); s_r^{op} = \text{MLP}^r(h_Q^{op}) \tag{2}$$

where h_Q^{op} denotes the sub-expression embedding for relation op in H_Q and $op \in \{+, -, \times, \dots, \}$, MLP^l and MLP^r are two multi-layer perceptrons for two argument slots. We obatin $2 \times N_Q$ argument slots (s_l^{op} and s_r^{op}) in total for all sub-expressions.

Then we fill the argument slots with two number tokens for every sub-expression. The number token is selected from the Number Vocabulary V_n:

$$p_l^{op}(n_i) = \text{Softmax}(s_l^{op} h_{n_i}), \forall h_{n_i} \in V_n \tag{3}$$

$$p_r^{op}(n_j) = \text{Softmax}(s_r^{op} h_{n_j}), \forall h_{n_j} \in V_n \tag{4}$$

where h_{n_i} is the i-th number embedding of the V_n. $p_l^{op}(*)$ and $p_r^{op}(*)$ denotes the two probability distributions for the first and the second operands in $(\underline{n_l}, \underline{n_r}, op)$.

Sub-expression Confidence: In addition to the argument slot, we also design a confidence slot for each sub-expression to score itself. Firstly, we merge all number embeddings using two number distributions $\{p_l^{op}(n_1), p_l^{op}(n_2), \cdots\}$ and $\{p_r^{op}(n_1), p_r^{op}(n_2), \cdots\}$ in (3) and (4):

$$m_\lambda^{op} = \sum_{i=1}^{N_n} p_\lambda^{op}(n_i) h_{n_i}, \lambda \in \{l, r\}, h_{n_i} \in V_n \tag{5}$$

and then we calculate the confidence slot s_c^{op} from sub-expression embedding h_Q^{op}, and fuse two vectors m_l^{op} and m_r^{op} with s_c^{op} by another MLP:

$$s_c^{op} = \text{MLP}^c(h_Q^{op}) \tag{6}$$

$$h_c^{op} = \text{MLP}^m([s_c^{op}; m_l^{op}; m_r^{op}; s_c^{op} \circ m_l^{op}; s_c^{op} \circ m_r^{op}; m_l^{op} \circ m_r^{op}]) \in \mathbb{R}^d \tag{7}$$

where $[;]$ means the concatenation of the vectors, $\circ$ denotes the element-wise multiplication. Finally, we calculate the confidence of this sub-expression by a binary classification MLP^c:

$$logit_c(\underline{n_l}, \underline{n_r}, op) = \text{MLP}^c(h_c^{op}) \tag{8}$$

C. Extension of Reasoning Chain

At step t, we obtain N_q candidate sub-expressions and their confidences (e.g., 2×3: 0.94, N $\div$ N[1]: 0.8 in Fig. 2). Each sub-expression represents a candidate node, and we select the node with the highest confidence (n_i^*, n_j^*, op^*) from N_q nodes:

$$h_{node}^t = \text{MLP}(s_c^{op^*} + V_n(n_i^*) + V_n(n_j^*)) + \mathbf{E}(t) \tag{9}$$

where $\mathbf{E}$(t) is a learnable embedding for step t that is shared by all data. The h_{node}^t is added to the reasoning chain as a new node and also as a new number for Vocabulary V_n^{t+1}. Then we use a state converter to model sequential node generation.

State Converter: Mathematical reasoning is similar to a Markov decision process, i.e., each step takes the optimal action based on the current state and then transfers it to the new state for the next iteration. So we design a state converter to update the reasoning state for the new step. We treat problem representation P^t and sub-expression representation H_Q^t as reasoning state and previously selected reasoning node h_{node}^t as action. Specifically, from step t to step $t+1$, we use Gated Recurrent Unit (GRU) [40] to update P^t and H_Q^t based on action h_{node}^t. The GRU predicts the increment of P^t and H_Q^t at step t, and then we add the predicted increment with the original input to obtain the new representations.

$$[\Delta P^{t+1}; \Delta H_Q^{t+1}] = \text{GRU}([P^t; H_Q^t] \mid h_{node}^t) \tag{10}$$

$$P^{t+1} = \Delta P^{t+1} + P^t; H_Q^{t+1} = \Delta H_Q^{t+1} + H_Q^t \tag{11}$$

where $\text{GRU}(*|h_{node}^t)$ means h_{node}^t is fed into GRU as the hidden state to predict the state increments. The incremental prediction updates the reasoning state at each step, as well as ensures the initial problem can be aware by all steps. Finally, we update the Number Vocabulary $V_n^{t+1} = \{h_{n_i}^{t+1}\}_{n_i=1}^{N_n} \cup h_{node}^t$ using H_Q^{t+1} and new number h_{node}^t.

Prefix Prompt: In the next reasoning step, we treat the new sub-expression embeddings H_Q^{t+1} as the new queries to feed into decoder as previous did ((1) to (8)). However, to enhance the interaction between different reasoning steps, we additionally add all previous sub-expressions as a prefix prompt for decoding

[1]We add Null number to Number Vocabulary V_n

Algorithm 1: Search on Sub-Expression Level.

Input:
S: Stack for sub-exp path $S_0 \leftarrow \{\langle 0, \varnothing \rangle\}$
$\mathcal{H}_t$: All possible valid sub-exp at step t
K: Max search size. T_m: Max sub-exp number
for $t \in \{1, \ldots, T_m\}$ **do**
  $B \leftarrow \varnothing$
  for $\langle path, score \rangle \in S_t$ **do**
    $H^{1:t}_{prefix} \leftarrow path$
    $\mathcal{H}_t = \{Decoder([H^{1:t}_{prefix}; H^t_Q], P^t)\}^K_1$
    for $exp \in \mathcal{H}_t$ **do**
      $s = score * exp.logit_{conf}$
      $path \leftarrow path \cup exp$
      $B.add(\langle path, s \rangle)$
    end for
  end for
  $S_t \leftarrow B.top(K)$
end for
Output: $B.max()$

new sub-expression, i.e., the embedding of all previous sub-expressions are fed into the decoder along with the new queries, which is different to the (1):

$$H^{1:t}_{prefix} = [h^1_{node}; h^2_{node}; \cdots; h^t_{node}] \tag{12}$$

$$H'^{t+1}_Q = \text{Decoder}([H^{1:t}_{prefix}; H^{t+1}_Q]; P^{t+1}) \tag{13}$$

The prefix prompt enables the decoding process to be aware of all previously generated sub-expression by self-attention. Finally, a new sub-expression is calculated based on H'^{t+1}_Q as the previous did (Section III-A). The design of the state converter and prefix prompt ensures the reasoning state is generated sequentially, while at the same time enabling more interactions between reasoning steps.

Multi-chain Search: Except for selecting the optimal node at each step for reasoning chain extension (9), we also obtain multiple reasoning chains by searching on sub-expression. As the example in Fig. 1, it produces three valid reasoning chains that bring higher interpretability and accuracy. Specifically, the decoder predicts multiple candidate nodes based on the current state, and then the top K valid nodes are reserved, forming K different reasoning chains. For the next step, each chain generates K subsequent nodes, and we select the top K results from the K^2 candidate nodes. The algorithm is described in Algorithm 1. Multi-chain Search is a sub-expression level beam search. We only use it as a plug-and-play module.

D. Training and Inference

We use the teacher forcing training strategy [41] as other generation method. As shown in Fig. 2, we assign a label $(c^{op}_t, l^{op}_t, r^{op}_t)$ to every sub-expression (detecting relation op) at step t, denoting confidence label (c^{op}_t) and two argument labels (l^{op}_t, r^{op}_t) respectively, e.g., *(F, N, N)* for the first sub-expression $(\underline{n_l}, \underline{n_r}, +)$ and *(T, 2, 2)* for the third sub-expression $(\underline{n_l}, \underline{n_r}, *)$, where *"F / T"* means the confidence is 0% or 100% and *"N"*, *"2"* means the Null number or "2" should be assigned to the proposed sub-expression. The loss can be written as:

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \sum_{j=1}^{N_q} [CE(logit^{op_j}_c, c^{op_j}_t) + CE(p^{op_j}_l, l^{op_j}_t) + CE(p^{op_j}_r, r^{op_j}_t)] \tag{14}$$

where θ denotes all parameters, and CE is the cross-entropy loss for binary classification (confidence) and multi-classification (argument). T denotes the equation has a total of T steps.

During the inference, we search for K nodes at each step and stop the generation if the confidences of all proposed sub-expressions are lower than 50% or the number of steps is longer than T_{max}.

IV. Experiments

Datasets: We evaluate our model on three standard datasets across two languages: MathQA [45], Math23K [27], and MAWPS [46]. We follow [17] to preprocess datasets. The statistics of datasets are reported in Section IV.

Baselines: We compare our method with three types of baselines: generative methods (Gen), discriminative methods (Dis), and contrastive generation (CL-Gen).

Generative method (Gen): [30] (**GroupAttn**) applied a multi-head attention approach using a seq2seq model. [1] proposed a seq2tree generation (**GTS**). [3] (**G2T**) introduced a graph encoder. [42], [47] added a PLMs encoder to GTS and G2T (**PLM-GTS, BERT-T**). [43] proposed a multilingual model (**mBERT**). [31] utilized Transformer for generation (**BERT-Gen**). [44] proposed a multi-task method (**Rank**). [34] introduced a neural symbolic method (**Symbol-Dec**). [35] extracted hierarchical features for encoder (**H-Reasoner**). [36] designed logical rules to guide decoding (**Logic-Dec**). [9] introduce adaptive symbolic compiler model (ELASTIC ♣) for numerical reasoning. On MathQA, ELASTIC adopted a different preprocessing procedure, incorporating a variety of operators. We replicated their results using their codebase. [21] (**ATHENA**) proposes a thought expansion network to generate equation thought-by-thought.

Discriminative method (Dis): [16] used a bottom-up DAG construction method (**DAG**). [17] introduced a relation extraction method (**RE-Ext**). [18] treated MWP as tagging annotation by M-Tree coding (**M-Tree**).

Constrasive generation (CL-Gen): [37] proposed a prototype learning (**Prototype**). [6] adopted a teacher model for discrimination (**T-Dis**). [38] distinguished examples with similar semantics but different logics (**Textual-CL**). [11] adopted an analogy identification to improve the generalization (**Ana-CL**). [8] aligned the representation of different traversal order for consistency (**M-View**).

In addition to the aforementioned models, we conduct evaluations on state-of-the-art large language models (LLMs), including GPT-3.5, GPT-4,[2] LLAMA-2-7B, and LLAMA-13B.[3]

[2][Online]. Available: https://openai.com/research/gpt-4
[3][Online]. Available: https://ai.meta.com/llama/

TABLE I
STATISTICS FOR THREE STANDARD DATASETS

Dataset	#Train/#Valid/#Test	#Avg. Token	#Avg. Sub-exp	#Max. Sub-exp
MathQA	16191 /2415/1606	39.6	4.17	12
Math23K	21162/1000/1000	26.6	2.26	20
MAWPS	1589/199/199	30.3	1.42	7

Utilizing the Chain-of-Thought prompting, we enabled these **Large Language Models (LLMs)** to autonomously generate a step-by-step solving process and the final answer.

Collaboration With LLMs: We devise a simple collaboration mechanism, orchestrating a collaborative solving process between the large language model (GPT-4) and the specialized smaller model (Ours). Specifically, we employ our sub-expression chain generation combined with multi-chain search to generate multiple candidate equations. These candidate equations, concatenated with the original question, are then used as a prompt for the LLMs. The detail of the prompt is as follows: *Please solve the given math question step-by-step. The input question is {Input query}. The possible equation solution are {Equ1, Equ2,...}. Please start.*.

Dataset Statistics: MathQA and Math23 K are two widely used large datasets that contain 20 k English mathematical problems and 23 k Chinese mathematical problems, respectively, and MAWPS contains 1.9 k English problems. The statistics of the dataset are shown in Table I. MathQA is the most difficult of the three datasets, both in terms of text length and the number of sub-expressions.

Training Details: Following most previous works [8], [17], we report the average accuracy (five random seeds) with standard deviation for Math23 K and MathQA, and 5-fold cross-validation for Math23 K and MAWPS. The test-set accuracy is chosen by the best dev-set accuracy step. We provide results for $K = 1$ (w/o search) and $K = 3$ and 8 (w/ multi-chain search). Besides, we adopt Roberta-base and Chinese-BERT as the encoder from HuggingFace[4] for multilingual datasets. We consider five mathematical operators, containing *Addition, Subtraction, Multiplication, Division, Exponentiation*, and various constants ($\{\pi, 1, 0, \cdots\}$) as previously. Our query decoder is a standard transformer decoder with 6 layers, each having 768 hidden units. We use an AdamW optimizer with a 5e-5 learning rate, batch size of 32 for MathQA and 26 for Math23 K. We set the T_{max} as 11. The other parameters are set as previous works [8], [17]. All experiments were set up on an NVIDIA RTX A6000.

Sampling Strategies For Sub-expression Token: Within our framework, each sub-expression is treated as a basic token and is generated one by one. Analogous to natural language generation, we can integrate various sub-expression sampling strategies to enhance the diversity of the generated results. Specifically, during each decoding step, we compute the probabilities for all candidate sub-expression tokens (as shown in 8) and normalize these probabilities. Subsequently, we employ *top-k* or *top-p* sampling to obtain the sub-expression at each step. In our experimental process, we set k to 3 for *top-k* sampling and p to 0.9 for *top-p* sampling, and report their respective results.

[4][Online]. Available: https://github.com/huggingface/transformers

TABLE II
RESULTS ON MATHQA

	Model	Test Acc.
Gen	GroupAttn[30]	70.4
	GTS [1]	71.3
	G2T[3]	72.0
	E-pointer†[13]	73.5
	BERT-T[42]	73.8
	mBERT[43]	77.1
	ELASTIC ♣ [9]	80.3
CL-Gen	T-Dis†[6]	73.1
	Prototype [37]	76.3
	Textual-CL†[38]	78
	Ana-CL[11]	79.6
	M-View♢[8]	80.6
	M-View♢† (w/o *Augmentation*)	79.5
Dis	M-Tree†[18]	76.5
	RE-Ext[17]	78.6
LLMs	GPT-3.5	42.5
	GPT-4	59.6
	LLAMA-2-Chat-7B	18.6
	LLAMA-2-Chat-13B	35.2
Gen	Ours (*w/o search*)	**81.7** ±0.36
	Ours (*top-k sampling*)	81.6 ±0.44
	Ours (*top-p sampling*)	81.4 ±0.39
	Ours (*w/ search K=3*)	**81.9** ±0.27
	Ours (*w/ search K=8*)	**82.5** ±0.33
	Ours (*w/ GPT-4*)	**84.1** ±0.67

† means our reproduction. ♢ and ♢† means M-View using augmentation dataset (report) and standard dataset (reproduction) respectively. ♣ means elastic uses a different data pre-processing process and operator set, so we reproduce their method with our settings.

A. Overall Performance

As shown in Table II, III and IV, our method achieves SoTA performance on all datasets, especially on the most difficult MathQA with +2.1% gains. Similarly, we gain +0.7% (Test) and +0.8% (5-fold) improvements on Math23 K and a slight improvement (+0.1%) on the MAWPS, although MAWPS is small and simple. Additionally, since most generative methods already adopt beam search for results, we can further improve the accuracy (+0.2%) and stability (−0.1% Std.) by sub-expression search.

From the view of three types of methods, our method is more stable and effective. Specifically, the discriminative and contrastive methods have a noticeable advantage over the conventional generation, since their models are more sophisticated and complex. However, our method is a purely generative method at the sub-expression level, but achieves superior performance, with +2.1% and +3.1% gains against best contrastive generation (Ana-CL) and discriminative baseline (RE-Ext) on MathQA. Similar phenomena are observed on other datasets. Furthermore, compared with the E-pointer, which also generates

TABLE III
TESTING AND FIVE-FOLD ACC. ON MATH23 K

	Model	Test	5-fold
Gen	GroupAttn[30]	69.5	66.9
	GTS [1]	75.6	74.3
	G2T[3]	77.4	75.5
	E-Pointer†[13]	78.7	76.5
	mBERT[43]	75.1	-
	Symbol-Dec[34]	-	75.7
	BERTGen[31]	76.6	-
	PLM-Gen [31]	76.9	-
	H-Reasoner[35]	83.9	82.2
	BERT-T[42]	84.4	82.3
	Rank†[44]	85.4	-
	Logic-Dec[36]	83.4	-
	ELASTIC ♣ [9]	84.8	82.9
	ATHENA[21]	84.4	-
CL-Gen	T-Dis[6]	79.1	77.2
	Prototype [37]	83.2	-
	Textual-CL [38]	85.0	82.6†
	Ana-CL [11]	85.6	83.2†
	M-View♢[8]	87.1	85.2
	M-View♢†*(w/o aug)*	85.6	83.3
Dis	DAG [16]	77.5	75.1
	M-Tree[18]	82.5	80.8†
	RE-Ext[17]	85.4	83.3
LLMs	GPT-3.5	55.1	54.5
	GPT-4	68.6	68.1
	LLAMA-2-Chat-7B	24.3	22.7
	LLAMA-2-Chat-13B	37.9	33.2
Gen	Ours (*w/o search*)	86.3 ± 0.35	84.1 ± 0.65
	Ours (*top-k sampling*)	86.5 ± 0.32	83.9 ± 0.58
	Ours (*top-p sampling*)	86.3 ± 0.38	84.2 ± 0.63
	Ours (*w/ search K=3*)	86.5 ± 0.25	84.3 ± 0.53
	Ours (*w/ search K=8*)	**86.9** ± 0.28	**84.6** ± 0.50
	Ours (*w/ GPT-4*)	83.5 ± 0.71	80.4 ± 0.68

We compare with reproduction M-View♢†, since original M-View used the augmentation dataset, which is not fair.

TABLE IV
FIVE-FOLD CROSS-VALIDATION RESULTS ON MAWPS

	Model	5-fold Acc.
Gen	GroupAttn[30]	76.1
	GTS [1]	82.6
	E-Pointer [13]	83.4
	G2T[3]	85.6
	Rank[45]	84.0
	BERTGen[31]	86.9
	PLM-Gen [31]	88.4
	PLM-GTS [42]	88.5
	PLM-G2T [42]	88.7
	H-Reasoner[35]	89.8
	ELASTIC ♣ [9]	91.9
	ATHENA[21]	92.2
CL-Gen	T-Dis[6]	84.2
	Prototype† [37]	89.6
	Textual-CL† [38]	91.3
	Ana-CL†[11]	91.8
	M-View♢ [8]	92.3
	M-View♢†*(w/o aug)*	92.1
Dis	M-Tree[18]	82.0
	RE-Ext[17]	92.2
LLMs	GPT-3.5	92.2
	GPT-4	**97.6**
	LLAMA-2-Chat-7B	57.1
	LLAMA-2-Chat-13B	78.9
Gen	Ours (*w/o search*)	92.3 ± 0.43
	Ours (*top-k Sampling*)	92.5 ± 0.45
	Ours (*top-p Sampling*)	92.2 ± 0.36
	Ours (*w/ search K=3*)	92.3 ± 0.31
	Ours (*w/ search K=8*)	92.7 ± 0.35
	Ours (*w/ GPT-4*)	96.6 ± 0.74

sub-expressions, our advantage is significant (+8.2%, +7.6%, +8.9%).

These experiments indicate our method benefits both from the design of generating sub-expressions as a unit, and iterative multi-query parallel decoding, which make our model more effective against the challenges of diverse and complex reasoning.

B. Different Sampling Strategies

Furthermore, we also demonstrate the efficacy of sub-expression level beam search and different sampling strategies. While the effect is relatively modest when the beam size is set to 3, it still reduces the standard deviation. Upon configuring the beam size to 8, we observed a more significant boost in performance (e.g., achieving a +0.63 improvement on the MathQA), albeit at the cost of extended inference durations. For different sampling strategies, such as top-p or top-k, we observe no significant differences. The results are generally similar to those obtained using greedy sampling. We hypothesize that this may be due to the limited number of candidate sub-expressions available.

Additionally, we also observe that for questions necessitating complex reasoning, the efficacy of sub-expression level beam search becomes more pronounced. Specifically, we divided the Math23 K dataset into two subsets based on the number of sub-expressions (#sub-exp$<$3 and #sub-exp$>$=4) and evaluated the effect of sub-expression level beam search (K=3) on each subset separately. We found that for the simple subset, the accuracy improved only by +0.15, whereas for the subset with #sub-exp$>$=4, the accuracy improved by +0.47. These results show that expression-level beam search is more effective for problems entailing more intricate reasoning processes.

C. Collaboration With LLMs

Beyond supervised learning paradigms, our model demonstrates significant advantages over Large Language Models (LLMs). Experimental results reveal that even the state-of-the-art GPT-4 struggles to achieve satisfactory performance on challenging datasets like MathQA and Math23 K, with a performance gap of approximately 20% on average. This suggests that for such intricate mathematical reasoning challenges, the LLMs might not be adequately equipped to address them.

TABLE V
ABLATION ON MATHQA (W/O SEARCH) BY 5 SEEDS

Decoder			State Converter		Acc. Avg(std)
Multi-query	Prefix Prompt	Sub-exp	Incre -pred	State Update	
✓	✓	✓	✓	✓	81.7 ±0.36
✗	✓	✓	✓	✓	80.5 ±0.31
✓	✗	✓	✓	✓	81.2 ±0.18
✗	✗	✗	✓	✓	78.5 ±0.42
✓	✓	✓	✗	✓	80.6 ±0.19
✓	✓	✓	✗	✗	80.9 ±0.12

TABLE VI
ABLATION EXPERIMENT STATISTICAL SIGNIFICANCE

Comparison	p-value	t-value(abs)
Full vs. w/o Multi-query	0.0002	6.44
Full vs. w/o Prefix-prompt	0.012	3.23
Full vs. w/o Sub-exp	6e-7	14.16
Full vs. w/o Incre-pred	0.0001	7.12
Full vs. w/o State Update	0.0007	5.33
w/o Multi-query vs. w/o Sub-exp	1.3e-5	9.44
w/o Prefix-prompt vs. w/o Sub-exp	5.1e-7	14.46
w/o Incre-pred vs. w/o State Update	0.0087	3.45

TABLE VII
THE EVALUATION OF TWO DIVERSE DATASETS, WHICH ARE RANDOMLY DEFORMED FROM TWO STANDARD DATASETS

Model	Diverse MathQA	ΔAcc.	Diverse Math23K	ΔAcc.
M-View	76.8	-2.7	84.7	-0.9
RE-Ext	76.7	-1.9	84.8	-0.6
M-Tree	76.3	-0.2	82.4	-0.1
Ours	**80.8**±0.38	-0.9	**85.9** ±0.4	-0.4

In Table II, IV, a considerable improvement is observed on both the MAWPS (+4.3) and MathQA (+2.4) when our model is integrated with GPT-4 (Ours+GPT4). It confirms that specialized small models (ours), through collaboration with general-purpose Large Language Models (GPT-4), can synergize and enhance both accuracy. However, we also observe an anomalous situation in Table III, where integrating GPT-4 actually led to a decrease in accuracy from 86.3% to 83.5%. Upon careful examination, we discovered that on Math23 K, GPT-4 occasionally makes erroneous selections, choosing the incorrect candidate sub-expression from our specialized small model. We suspect this is because Math23 K is a Chinese mathematical reasoning dataset that includes some semantically ambiguous problems, leading Large Language Models (LLMs) to misinterpret mathematical problems and engage in improper reasoning.

However, from another perspective, simply utilizing a specialized small model containing only 100 M parameters to provide a set of candidate sub-expressions can significantly enhance the reasoning capabilities of LLMs (from 68.6 to 83.5), even in unfamiliar Chinese contexts. This also illustrates the importance of collaboration between specialized small models and general LLM.

D. Ablation Experiments

The query decoder and state converter are two key designs for reasoning chain generation. Therefore, we ablate several modules in two designs: I. **Multi-query**. Firstly, we ablate the design of multiple relation queries and instead use a single query for decoding two numbers and their math relation. II. **Prefix Prompt**. Then we remove the prefix prompt from the input of the decoder. The decoder cannot be aware of the previous reasoning nodes at each step. III. **Sub-exp**. We replace the parallel extraction of sub-expression with predicting tokens one by one for a sub-expression. IV. **Incre-pred**. Moreover, the incremental prediction is removed from the state converter. Instead, the GRU directly predict the P^{t+1} and H_Q^{t+1} (11). V. **State Update**. Finally, the state converter is fully removed. As shown in Table V, all ablations bring decreases in accuracy. Obviously, sub-expression generation has the most significant effect (-3.2%), followed by multi-query (-1.2%) and incremental prediction (-1.1%). It illustrates that (1) Extracting a sub-expression at each step is more suitable for MWP than a token. (2) Multiple query decoding enhances the interaction between different math relations and improves accuracy. (3) Incremental updating ensures the initial problem is aware by each step, which is necessary for global reasoning.

Moreover, as shown in the Table VI, we also examined whether the ablation studies were statistically significant. We adopt a two-sample one-sided t-test (df=8) for six models (Full+Five Ablations) in Table V. Compared to the full model, the decrease of five ablation experiments is statistically significant (p-value=0.0002 / 0.012 / 6e-7/ 0.0001 / 0.0007). Besides, the differences between ablation models are also significant, e.g., w/o Multi-query vs. w/o Sub-exp: p=1.3e-5. It indicates each module contributes independently to the performance.

E. Analysis for Challenges

We analyze whether our model can handle diverse annotations and complex reasoning challenges.

Diverse Annotation: The equation is always diverse due to the mathematical law. These diverse equation sequences cause inconsistent order (Fig. 1), which is challenging for the generative method. To measure this, we use math laws to randomly deform the labeled equation in the standard dataset (e.g., $a \times b + c \rightarrow c + b \times a$), and the deformed and original equation are used for training together.

We select three baselines (RE-Ext/M-View/M-Tree) for comparison. M-View and RE-Ext are selected from the categories of generative and discriminative methods, which have high accuracy on both datasets. Additionally, M-Tree is a method for addressing annotation diversity. As shown in Table VII, diverse annotation is quite challenging as the performance of all methods is significantly degraded. But our model still achieves the highest accuracy with a slight drop. In detail, the M-Tree method is almost unaffected due to its special labeling manner, but the

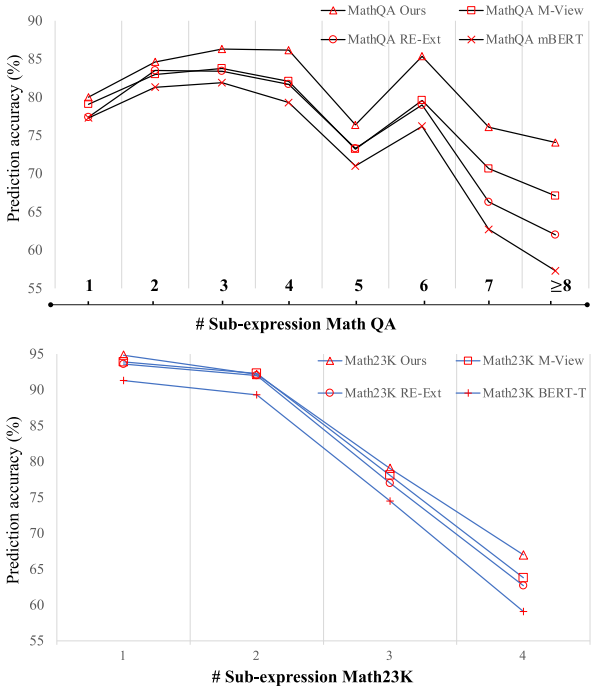

Fig. 3. Performance on the sample with the different number of sub-expression.

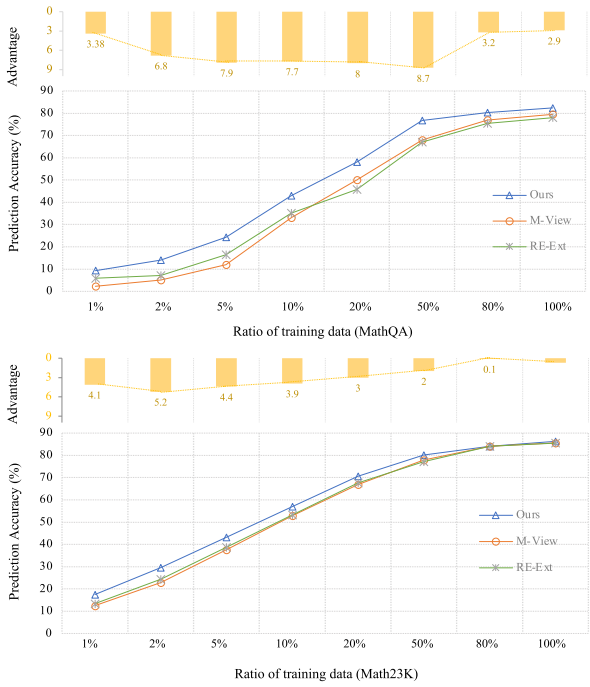

Fig. 4. Low resource analysis. Top: Few shot analysis on MathQA. Bottom: Few shot analysis on Math23 K. Bar Chart: Our advantage over the best baseline. Line Chart: Prediction Accuracy among three methods.

absolute accuracy is lower than ours by 4.5% and 3.5%. Except for M-tree, our method suffer the slightest performance drop (−0.9% and −0.4%) among the three methods, while the other two methods are sensitive with a significant drop, especially for token-level generation (M-view: −2.7% and −0.9%). The results show our sub-expression parallel generation can reduce the dependence on annotation order and achieve a good trade-off between accuracy and robustness against the diverse annotation challenges.

Complex Reasoning: We examine our model in complex reasoning scenarios. Since a sub-expression means a reasoning step, we use the number of sub-expressions to measure the complexity of math reasoning. In Fig. 3, as the number of sub-expressions increases, the reasoning process becomes more complex and the performance decreases rapidly. However, our method consistently maintains high accuracy (≥74% on MathQA) across all cases, especially on the more complex case. Compared with baselines, our advantage increases from +1.7% (#3) to +5.4% (#6). For the problem with the longest reasoning steps (≥#8), our model outperforms the best baseline by nearly +8%, showing that reasoning on sub-expression is more suitable than conventional methods.

F. Analysis for Learning Efficiency

Previous experiments have demonstrated the advantages of our method in terms of absolute accuracy and addressing challenges. Besides our learning process is also efficient in data and parameters.

Data Efficiency in Low Resource: We analyze how our method performs under a low resource setting. The Fig. 4 shows the performance of our model and baselines with partial data for training. It illustrates that (1) Our method can achieve ≥14% accuracy only on 2% training data and ≥40% on 10% data, which is consistently higher than the other methods. (2) Our advantages are more noticeable under lower resource cases on Math23 K (+0.1% improvements on 80% data, +3% on 20%, +5.2% on 2%). A similar phenomenon can be observed on MathQA, i.e., relative advantage improved from +3.2% to +8%. These results indicate our method is more efficient in utilizing limited data for learning, as it is a purely generative framework.

Parameter Efficiency: We explore whether similar performance can be achieved with a smaller model. We reduce the number of transformer layers in the query decoder (Section II-I-B), i.e., the number of decoder layers is set from 6 to 0, where 0 means removing the decoder totally. As shown in Fig. 5, there is no significant loss of model accuracy when the number of decoder layers becomes smaller. We still gain +1.5% improvements than the SoTA baseline even if we use only one layer. However, if the decoder is removed, the performance drops dramatically (−14%), since the model may lack interactions between query, problem and prefix prompt. The result shows we can achieve a new SoTA result only using a lightweight model (Decoder with one layer: 135 M), which is efficient for parameters.

G. Case Study and Visualization

In Fig. 6, we set up five mathematical relations, each corresponding to a learnable query vector. During sub-expression generation, a decoder is used to convert the five queries into five

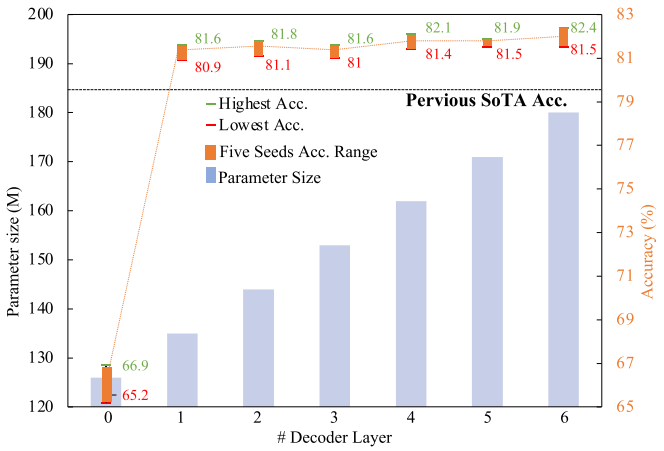

Fig. 5. Experiments with the different number of decoder layers. Bar (Left Axis): Parameter Size. Candlestick Chart (Right Axis): Five-seed Accuracy Range.

candidate sub-expressions, each with two argument slots and a confidence slot to be filled. We analyze this parallel slot-filling process by visualizing the similarity between the slot and the problem. Firstly, we extract every argument slot embedding ((2) in Section III-B) from all proposed sub-expressions by the decoder and also store the problem representations ((11) in Section III-C) at each reasoning step. Then we calculate the similarity between the problem representation and $2 \times N_q$ argument slots $(s_l^+, s_r^+, s_l^-, s_r^-, \cdots)$ for each reasoning step. As shown in Fig. 6, at each reasoning step, the corresponding sub-expression is activated, and its two argument slots also match the most relevant semantic segment and number from the problem text. For example, in the first reasoning step of case 1, the sub-expression for detecting division relation is significantly more relevant than the other four sub-expressions. Then the first argument slot (*Slot_L_Div*) in this sub-expression is matched to the text segment *"a distance of 600 meters"*, while the second argument slot (*Slot_R_Div*) is more focused on the segment *"1000 meters"*. Both fragments are very useful for reasoning at the current step. A similar phenomenon can be observed in the more complex second case. This visualization shows that our parallel sub-expression generation process can attend to different mathematical features at each step for reasoning, thus generating the corresponding sub-expressions step by step.

V. Discussions

A. Why Decode Sub-Expressions in Parallel?

Unlike prior work on sub-expression generation, such as methodologies that predict operators and operands sequentially using a generator (e.g., EPT [13] and ELASTIC [9]) or focus solely on generating a single sub-expression (e.g., RE-Ext [17]), our approach utilizes parallel extraction. Specifically, we employ multiple queries to extract all candidate sub-expressions concurrently at each step. Our query-based extractors can output all possible combinations of operators and operands in parallel. The rationale behind our approach is that token-level sequence generation is not well-suited for diverse mathematical expressions.

- Firstly, preliminary experiments reveal that training on token-level equation sequences may introduce biases related to ordering. Order bias refers to the model only learning to generate equations in a fixed order after training on single annotations. E.g., for $(a+b)*(c-d)$, the model may assume that a must be generated before b or $(c-d)$ must appear after $(a+b)$. These biases potentially impair mathematical reasoning capabilities.
- Secondly, equation augmentation can mitigate order biases, e.g., deforming $(a+b)*(c-d)$ into $(c-d)*(a+b)$. It provides more diverse annotation for learning. However, training on diverse labels may be unstable as it makes loss calculation difficult. [10] employs a controllable generation method to address this issue. In contrast, we modify the decoding and labeling approach within sub-expressions. Specifically, for decoding, we generate a complete sub-expression in parallel at each step, thereby eliminating the decoding order within the sub-expression.
- Lastly, our parallel sub-expression extractors are more flexible as it is possible to predict multiple sub-expressions at one step, e.g., in Step-1, two valid expressions, (v_1+v_2) and (t_1-t_2), are output simultaneously. In Step-2, the results of the two expressions from the previous step are combined, i.e., $(v_1+v_2)\times(t_1-t_2)$. Previous methods did not consider this. As shown in TABLE II, III, and IV, our method outperforms EPT by +8.2%, +7.6%, +8.9%.

B. Can We Solve All Problems Caused by Inconsistent Annotation?

We have focused mainly on the diversity within a sub-expression ($1*2+3$ and $3+2*1$), i.e., our sub-expression generation method can effectively alleviate the challenge of diversity annotation by extracting multiple expressions in parallel. However, we find our method still has trouble in addressing the inconsistent annotation due to the parentheses and distributive law, e.g., $(1+2)\times 3$ and $1\times 3+2\times 3$.

C. How to Collaborate More Closely With LLMs?

Our multi-chain search is also different from previous approaches. [10] propose a controlled equation generation solver by leveraging a set of control codes. However, our method employs a sub-expression level beam search at each step to obtain multiple distinct sub-expression chains. Furthermore, our sub-expression level beam search can seamlessly integrate with LLMs, thereby enabling collaborative reasoning.

Beyond the collaborative methods between LLMs and specialized smaller models mentioned in our experiments, there remains vast potential to delve deeper into more profound collaborations between them. For instance, as with most previous work [17], [34], [35], our work also requires some constant words (e.g., $\{1, \pi, \cdots\}$) for classification when generating equations. As in previous work, we have to predefine some constants

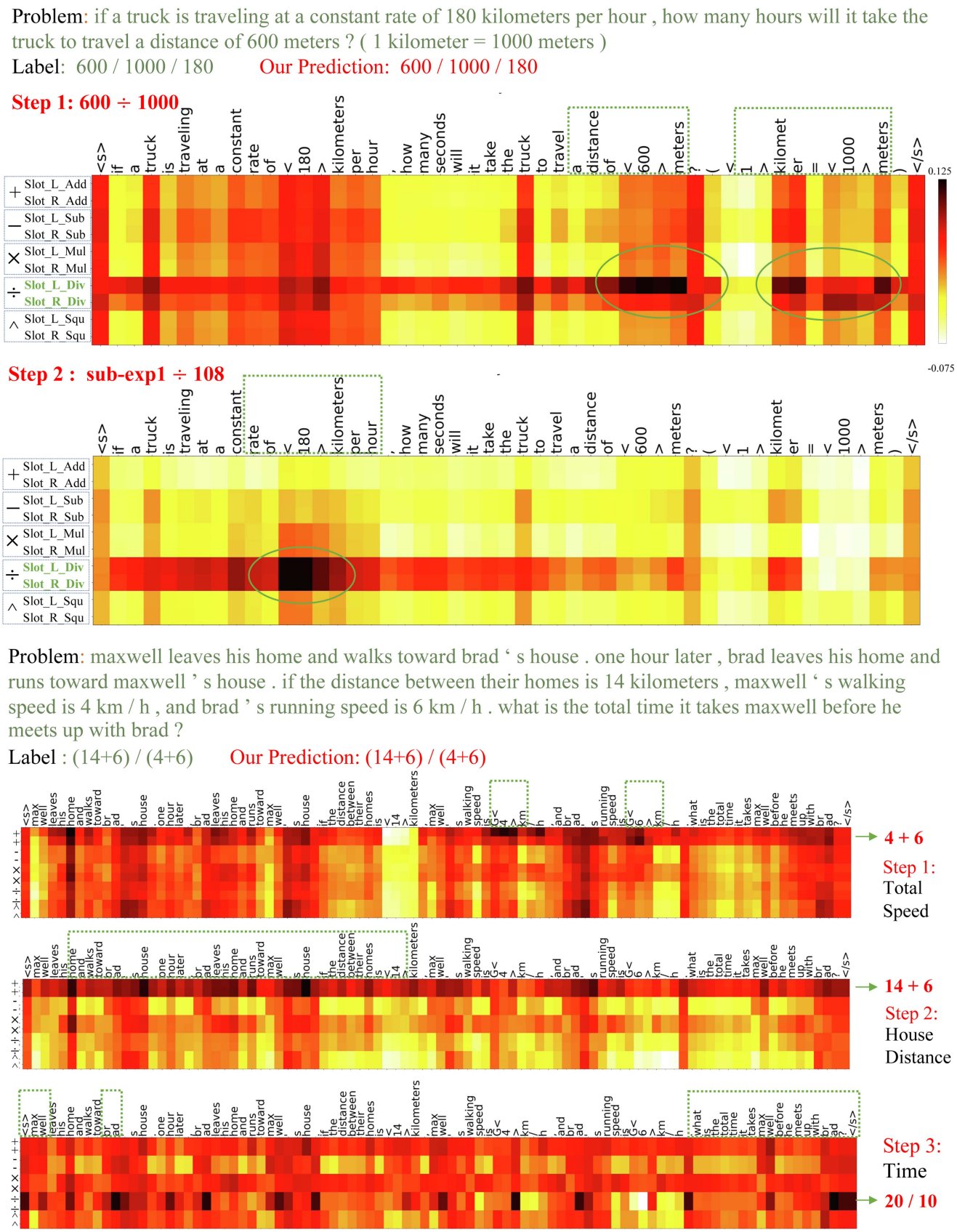

Fig. 6. We visualize the sub-expression generation process. We calculate the cosine similarity between all argument slots of the five sub-expressions and problem representations at each reasoning step. *Slot_L_Add* and *Slot_R_Add* denote the first and the second slots of the sub-expression for detecting additive relation.

for generation in advance, which may limit the application of the model in real scenarios. In the future, we will explore how to inject mathematical knowledge using LLMs, such as distributive laws, into our model and quickly select constant words from the universal vocabulary.

VI. Conclusion

We design a novel sub-expression level generative framework by considering sub-expressions as a minimum generative unit. The sub-expression is decoded in parallel from multiple relation queries as a reasoning step, and the whole reasoning chain is continuously extended by the state converter. Experiments on multilingual datasets demonstrate our method outperforms previous SoTA by a large margin. Besides, our method is more stable and efficient against the challenges of diverse annotation and complex reasoning with limited resources.

References

[1] Z. Xie and S. Sun, "A goal-driven tree-structured neural model for math word problems," in *Proc. Int. Joint Conf. Artif. Intell.*, 2019, pp. 5299–5305. [Online]. Available: https://www.ijcai.org/proceedings/2019/0736.pdf

[2] Q. Liu, W. Guan, S. Li, and D. Kawahara, "Tree-structured decoding for solving math word problems," in *Proc. Conf. Empirical Methods Natural Lang. Process. 9th Int. Joint Conf. Natural Lang. Process.*, 2019, pp. 2370–2379. [Online]. Available: https://aclanthology.org/D19-1241

[3] J. Zhang et al., "Graph-to-tree learning for solving math word problems," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 3928–3937. [Online]. Available: https://aclanthology.org/2020.acl-main.362.pdf

[4] S. Li, L. Wu, S. Feng, F. Xu, F. Xu, and S. Zhong, "Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem," in *Proc. Findings Assoc. Comput. Linguistics, EMNLP*, 2020, pp. 2841–2852. [Online]. Available: https://aclanthology.org/2020.findings-emnlp.255

[5] Y. Zhang, G. Zhou, Z. Xie, and J. X. Huang, "HGEN: Learning hierarchical heterogeneous graph encoding for math word problem solving," *IEEE/ACM Trans. Audio, Speech Lang. Process.*, vol. 30, pp. 816–828, 2022.

[6] Z. Liang et al., "Solving math word problems with teacher supervision," in *Proc. 13th Int. Joint Conf. Artif. Intell.*, 2021, pp. 3522–3528. [Online]. Available: https://www.ijcai.org/proceedings/2021/485

[7] Q. Liu, W. Guan, S. Li, F. Cheng, D. Kawahara, and S. Kurohashi, "RODA: Reverse operation based data augmentation for solving math word problems," *IEEE/ACM Trans. Audio, Speech Lang. Process.*, vol. 30, pp. 1–11, 2022.

[8] W. Zhang et al., "Multi-view reasoning: Consistent contrastive learning for math word problem," in *Proc. Findings Assoc. Comput. Linguistics: EMNLP*, Abu Dhabi, UAE, Dec. 2022, pp. 1103–1116.

[9] J. Zhang and Y. Moshfeghi, "ELASTIC: Numerical reasoning with adaptive symbolic compiler," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 12647–12661.

[10] Y. Shen, Q. Liu, Z. Mao, Z. Wan, F. Cheng, and S. Kurohashi, "Seeking diverse reasoning logic: Controlled equation expression generation for solving math word problems," in *Proc. 2nd Conf. Asia-Pacific Ch. Assoc. Comput. Linguistics 12th Int. Joint Conf. Natural Lang. Process.*, Nov. 2022, vol. 2, pp. 254–260.

[11] Z. Liang, J. Zhang, and X. Zhang, "Analogical math word problems solving with enhanced problem-solution association," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Abu Dhabi, UAE, 2022, pp. 9454–9464.

[12] W. Zhang, Y. Shen, Q. Nong, Z. Tan, Y. Ma, and W. Lu, "An expression tree decoding strategy for mathematical equation generation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2023, pp. 439–456. [Online]. Available: https://aclanthology.org/2023.emnlp-main.29

[13] B. Kim, K. S. Ki, D. Lee, and G. Gweon, "Point to the expression: Solving algebraic word problems using the expression-pointer transformer model," in *Proc. Conf. Empirical Methods Natural Lang. Process*, 2020, pp. 3768–3779. [Online]. Available: https://aclanthology.org/2020.emnlp-main.308

[14] B. Kim, K. S. Ki, S. Rhim, and G. Gweon, "EPT-X: An expression-pointer transformer model that generates eXplanations for numbers," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 4442–4458. [Online]. Available: https://aclanthology.org/2022.acl-long.305

[15] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, vol. 30, pp. 5998–6008

[16] Y. Cao, F. Hong, H. Li, and P. Luo, "A bottom-up DAG structure extraction model for math word problems," in *Proc. 35th AAAI Conf. Artif.*, 2021, pp. 39–46. [Online]. Available: https://yixuancao.github.io/files/AAAI-2021-MWP.pdf

[17] Z. Jie, J. Li, and W. Lu, "Learning to reason deductively: Math word problem solving as complex relation extraction," in *Proc. 60th Annu. Meeting Assoc. Comput. Linguistics*, 2022, pp. 5944–5955. [Online]. Available: https://aclanthology.org/2022.acl-long.410

[18] B. Wang, J. Ju, Y. Fan, X. Dai, S. Huang, and J. Chen, "Structure-unified m-tree coding solver for mathword problem," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Abu Dhabi, UAE, Dec. 2022, pp. 8122–8132.

[19] J. Wei et al., "Chain-of-thought prompting elicits reasoning in large language models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, vol. 35, pp. 24824–24837.

[20] A. Chowdhery et al., "PaLM: Scaling language modeling with pathways," *J. Mach. Learn. Res.*, vol. 24, no. 240, pp. 1–113, 2023.

[21] J. Kim, H. Kim, J. Hahn, and Y.-S. Han, "ATHENA: Mathematical reasoning with thought expansion," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2023, pp. 16315–16327. [Online]. Available: https://aclanthology.org/2023.emnlp-main.1014

[22] Y. Bakman, "Robust understanding of word problems with extraneous information," 2007. [Online]. Available: https://arxiv.org/abs/math/0701393

[23] M. Yuhui, Z. Ying, C. Guangzuo, R. Yun, and H. Ronghuai, "Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems," in *Proc. IEEE 2nd Int. Workshop Educ. Technol. Comput. Sci.*, 2010, vol. 2, pp. 476–479. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/5458590

[24] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay, "Learning to automatically solve algebra word problems," in *Proc. 52nd Annu. Meeting Assoc. Comput. Linguistics*, 2014, pp. 271–281. [Online]. Available: https://aclanthology.org/P14-1026.pdf

[25] S. Roy and D. Roth, "Mapping to declarative knowledge for word problem solving," *Trans. Assoc. Comput. Linguistics*, vol. 6, pp. 159–172, 2018. [Online]. Available: https://doi.org/10.1162/tacl_a_00012

[26] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2019, pp. 4171–4186. [Online]. Available: https://aclanthology.org/N19-1423

[27] Y. Wang, X. Liu, and S. Shi, "Deep neural solver for math word problems," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2017, pp. 845–854. [Online]. Available: https://aclanthology.org/D17-1088/

[28] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, "Program induction by rationale generation: Learning to solve and explain algebraic word problems," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, 2017, pp. 158–167. [Online]. Available: https://aclanthology.org/P17-1015

[29] T.-R. Chiang and Y.-N. Chen, "Semantically-aligned equation generation for solving and reasoning math word problems," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2019, pp. 2656–2668. [Online]. Available: https://aclanthology.org/N19-1272

[30] 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 *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics*, 2019, pp. 6162–6167. [Online]. Available: https://aclanthology.org/P19-1619

[31] Y. Lan et al., "MWPToolkit: An open-source framework for deep learning-based math word problem solvers," *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 11, 2022, pp. 13188–13190.

[32] Q. Wu, Q. Zhang, J. Fu, and X.-J. Huang, "A knowledge-aware sequence-to-tree network for math word problem solving," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2020, pp. 7137–7146.

[33] Q. Wu, Q. Zhang, Z. Wei, and X. Huang, "Math word problem solving with explicit numerical values," in *Proc. 59th Annu. Meeting Assoc. Comput Linguistics 11th Int. Joint Conf. Natural Lang. Process.*, 2021, pp. 5859–5869. [Online]. Available: https://aclanthology.org/2021.acl-long.455

[34] J. Qin, X. Liang, Y. Hong, J. Tang, and L. Lin, "Neural-symbolic solver for math word problems with auxiliary tasks," in *Proc. 59th Annu. Meeting Assoc. Comput. Linguistics 11th Int. Joint Conf. Natural Lang. Process*, 2021, pp. 5870–5881. [Online]. Available: https://aclanthology.org/2021.acl-long.456

[35] W. Yu, Y. Wen, F. Zheng, and N. Xiao, "Improving math word problems with pre-trained knowledge and hierarchical reasoning," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2021, pp. 3384–3394. [Online]. Available: https://aclanthology.org/2021.emnlp-main.272

[36] Z. Yang, J. Qin, J. Chen, L. Lin, and X. Liang, "LogicSolver: Towards interpretable math word problem solving with logical prompt-enhanced learning," in *Proc. Findings Assoc. Comput. Linguistics: EMNLP*, 2022, pp. 1–13.

[37] Z. Li et al., "Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems," in *Proc. Findings. Assoc. Comput. Linguistics*, S. Muresan, P. Nakov, and A. Villavicencio, Eds., Dublin, Ireland, May 2022, pp. 2486–2496.

[38] Y. Shen, Q. Liu, Z. Mao, F. Cheng, and S. Kurohashi, "Textual enhanced contrastive learning for solving math word problems," in *Proc. Findings. Assoc Comput. Linguistics: EMNLP*, 2022, pp. 4297–4307.

[39] Y. Liu et al., "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*. [Online]. Available: https://arxiv.org/abs/1907.11692

[40] K. Cho et al., "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1724–1734.

[41] R. J. Williams and D. Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Comput.*, vol. 1, no. 2, pp. 270–280, 1989.

[42] Z. Liang, J. Zhang, J. Shao, and X. Zhang, "MWP-BERT: A strong baseline for math word problems," 2021, *arXiv:2107.13435*.

[43] M. Tan, L. Wang, L. Jiang, and J. Jiang, "Investigating math word problems using pretrained multilingual language models," in *Proc 1st Workshop on Math. Natural Lang. Process. (MathNLP)*, 2022, pp. 7–16.

[44] J. Shen et al., "Generate & rank: A multi-task framework for math word problems," in *Proc. Findings Assoc. Comput. Linguistics, EMNLP*, 2021, pp. 2269–2279. [Online]. Available: https://aclanthology.org/2021.findings-emnlp.195

[45] A. Amini, S. Gabriel, S. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi, "MathQA: Towards interpretable math word problem solving with operation-based formalisms," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2019, pp. 2357–2367. [Online]. Available: https://aclanthology.org/N19-1245

[46] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, "MAWPS: A math word problem repository," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2016, pp. 1152–1157. [Online]. Available: https://aclanthology.org/N16-1136.pdf

[47] A. Patel, S. Bhattamishra, and N. Goyal, "Are NLP models really able to solve simple math word problems?," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2021, pp. 2080–2094. [Online]. Available: https://aclanthology.org/2021.naacl-main.168