Complex & Intelligent Systems (2024) 10:5805–5830
https://doi.org/10.1007/s40747-024-01454-8

ORIGINAL ARTICLE

Comparative study of typical neural solvers in solving math word problems

Bin He[1] · Xinguo Yu[1] · Litian Huang[1] · Hao Meng[1] · Guanghua Liang[1] · Shengnan Chen[1]

Received: 27 March 2023 / Accepted: 17 April 2024 / Published online: 22 May 2024

Abstract

In recent years, there has been a significant increase in the design of neural network models for solving math word problems (MWPs). These neural solvers have been designed with various architectures and evaluated on diverse datasets, posing challenges in fair and effective performance evaluation. This paper presents a comparative study of representative neural solvers, aiming to elucidate their technical features and performance variations in solving different types of MWPs. Firstly, an in-depth technical analysis is conducted from the initial deep neural solver DNS to the state-of-the-art GPT-4. To enhance the technical analysis, a unified framework is introduced, which comprises highly reusable modules decoupled from existing MWP solvers. Subsequently, a testbed is established to conveniently reproduce existing solvers and develop new solvers by combing these reusable modules, and finely regrouped datasets are provided to facilitate the comparative evaluation of the designed solvers. Then, comprehensive testing is conducted and detailed results for eight representative MWP solvers on five finely regrouped datasets are reported. The comparative analysis yields several key findings: (1) Pre-trained language model-based solvers demonstrate significant accuracy advantages across nearly all datasets, although they suffer from limitations in math equation calculation. (2) Models integrated with tree decoders exhibit strong performance in generating complex math equations. (3) Identifying and appropriately representing implicit knowledge hidden in problem texts is crucial for improving the accuracy of math equation generation. Finally, the paper also discusses the major technical challenges and potential research directions in this field. The insights gained from this analysis offer valuable guidance for future research, model development, and performance optimization in the field of math word problem solving.

Keywords Comparative analysis · Deep learning model · Math expression decoding · Math word problem solving · Problem text encoding

Abbreviations

MWPs	Math Word Problems
PLM	Pre-trained Language Model
TreeDecoder	Tree-structural Decoder
DL	Deep Learning
DT	Depth-first decomposing Tree
BT	Breadth-first decomposing Tree
MAD	Mean accuracy difference
UET	Universal Expression Tree
UDG	Unit Dependency Graph

Symbols

Symbol	Explanation
P	Problem text
V	Word token of the P
$F_{encoding}(.)$	Encoding network
$F_{decoding}(.)$	Decoding network
h_i^p	The hidden vector state i
Q,K and V	the query matrix, key matrix and value matrix separately

✉ Xinguo Yu
xgyu@ccnu.edu.cn

Bin He
hebin@ccnu.edu.cn

Litian Huang
litianhuang@mails.ccnu.edu.cn

Hao Meng
menghao@mails.ccnu.edu.cn

Guanghua Liang
liangguanghua@mails.ccnu.edu.cn

Shengnan Chen
shengnanC@mails.ccnu.edu.cn

[1] Faculty of Artificial Intelligence in Education, Central China Normal University, Wuhan, China

$G = (V, E)$	A graph with vertex V and edge E
W_1, W_2, b	trainable parameters and bias
L	Expression length
H	Expression tree depth
C	Implicit condition
S	Arithmetic situation
E_{acc}	Equation accuracy
ME	Math expressions
A_{acc}	Answer accuracy

Introduction

Math Word Problem (MWP) solving has been a long-standing research problem in the field of artificial intelligence [1]. However, previous methods required hand-crafted features, making them less effective for general problem-solving. In a milestone contribution, Wang et al. [2] designed the first deep learning-based algorithm, DNS, to solve MWPs, eliminating the need for hand-crafted features. Since then, multiple neural solvers with various network cells and architectures have emerged [3–9], with pioneering experiments conducted on diverse datasets with varying sizes and characteristics [1, 10]. However, the experimental results show that even MWP solvers built with similar architectures exhibit varying performance on datasets with different characteristics. Hence, a precise and impartial analysis of the existing MWP solvers has become essential to reveal the potential factors of network cells and architectures that affect the performance of neural solvers in solving different characteristics of MWPs.

Earlier MWP solvers leveraged manually designed rules or semantic parsing to map problem text into math equations, followed by an equation solver to obtain the final answer. These early efforts could only solve a limited number of problems defined in advance. Inspired by deep learning models for natural language processing [11, 12], recent neural solvers use an Encoder-Decoder framework [13] to transform a sequence of problem sentences into another sequence of arithmetic expressions or equations. The Encoder captures the information presented by the problem text, which can be divided into two categories: sequence-based representation learning [5, 14, 15] and graph-based representation learning [6, 16, 17]. Sequence-based representation learning processes the problem text as a sequence of tokens using recurrent neural networks [18, 19] or transformers [11], while graph-based representation learning constructs a graph from the problem text. Graph neural networks (e.g., graph transformer model [20], inductive graph learning model [21]) are then used to learn a representation for the entire graph. Mathematical expressions can be viewed as sequences of symbols or modeled as trees based on their syntactic structure, allowing Decoders to predict output expressions based on the encoding vectors produced by the encoder. By combining different types of encoders and decoders, diverse architectures of MWP solvers have been developed, including Seq2Seq-based solvers, Seq2Tree-based solvers, and Graph2Tree-based solvers.

Several reviews and surveys have been conducted to examine the progress of research in this field. For example, Mukherjee et al. [22] made a first attempt to analyze mathematical problems solving systems and approaches according to different disciplines. Zhang et al. [1] classified and analyzed different representation learning methods according to technical characteristics. Meadows et al. [23] and Lu et al. [24] conducted a literature review on the recent deep learning-based models for solving math word problems. Lan et al. [10] established a unified algorithm test platform and conducted comparative experiments on typical neural solvers. While these reviews provide valuable insights into the field of automatic math word problem solving, little comparative evaluation has been carried out to reveal the performance variations of neural solvers with different architectures in solving various types of MWPs. An initial attempt can be found in [10] which provides a collection of experimental results of the typical neural solvers on several datasets. However, no other attempts to explore the performance variations of neural solvers with different architectures in solving different types of math word problems.

While significant efforts have been made, there remains a lack of comprehensive technical analysis to compare different network structures and their impacts on final performance. This paper presents a comparative study of typical neural solvers to unveil their technical features and performance variations in solving MWPs with diverse characteristics. We initially identify the architectures of typical neural solvers, rigorously analyzing the framework of each category, notably: Seq2Seq [2, 4], Seq2Tree [5, 25], Graph2Tree [6, 17] and PLM-based models [26–30]. We propose a four-dimensional indicator to categorize the considered datasets for precise evaluation of neural solvers' performance in solving various characteristics of MWPs. Typical neural solvers are disassembled into highly reusable components, enabling researchers to reconstruct them and develop new solvers by replacing components with proposed ones, which benefits both model evaluation and extension. To assess the considered solvers, we establish a testbed and conduct comprehensive experiments on five popular datasets using eight representative MWP solvers, followed by a comparative analysis of the results achieved. The contributions of our work can be summarized as follows:

(1) We provide a comprehensive and systematic analysis of deep learning-based MWP solvers, ranging from the initial deep neural solver DNS to the latest GPT-4. This is achieved through an in-depth technical analysis of net-

work structures and neural cell types, enabling a deeper understanding of the technological evolution of MWP solvers for the research community.
(2) To enhance the technical analysis, we introduce a unified framework consisting of reusable encoding and decoding modules decoupled from existing MWP solvers. This framework allows for the straightforward reproduction and extension of typical MWP solvers by combining these reusable modules.
(3) We establish a testbed and provide finely regrouped datasets to facilitate objective and fair evaluations of MWP solvers. Through this testbed, we conduct comprehensive testing and report detailed results for eight representative MWP solvers on five finely regrouped datasets, specifically highlighting the performance variations of solvers with different modules in solving different types of MWPs.
(4) We present three key findings from our experiments and discuss the major technical challenges and potential research directions in this field.

The rest of the paper is organized as follows: Sect. "Related work" describes related work on math word problem solving. Section "Architecture and technical feature analysis of neural solvers" provides a detailed analysis of the framework of typical neural solvers. A characteristic analysis of the considered datasets is presented in Sect. "Characteristics analysis of benchmark datasets", and experiments and a comparative analysis are conducted in Sect. "Experiment". We conclude this paper in Sect. "Conclusion".

Related work

In this section, we will explore various deep learning-based approaches for solving math word problems. We will also provide an introduction to previous surveys in this field.

Deep learning-based approaches for solving MWPs

Solving MWPs has been a longstanding research focus in the field of artificial intelligence since the 1960s, as illustrated in Fig. 1. The evolution of MWP solvers can be categorized into different stages based on the underlying technologies utilized, including rule-based approaches [31], semantic parsing-based approaches [16, 32–34], etc.. More recently, neural networks inspired by deep learning models for natural language processing [11, 12] have been designed to tackle MWPs. For instance, the Deep Neural Solver (DNS) [2] is the first deep learning algorithm capable of translating problem texts to equations without relying on manually-crafted features. This advantage has motivated extensive research on neural solvers using larger datasets, as evidenced by several studies in the literature.

A significant challenge in these studies is efficiently capturing the logical relationships between natural language texts and their corresponding equations [1] which is known as problem text representation and equation representation. Inspired by translation models [19], MWP solver is typically designed as an Encoder-Decoder framework [1] as shown in Table 1. The Encoder is responsible for learning the semantic representation and logic relationships presented explicitly or implicitly of the problem text. Researchers have tried different sequence models, leading to several representative models such as DNS [2], MathEN [4]. The Decoder, usually designed as a sequence or tree structural model, treats the math equation as a symbolic sequence consisting of numbers and operators for decoding. Several tree-structured models, such as Tree-Dec [25], GTS [6], were designed and then widely accepted for math equation decoding to enhance the math equation generation. Recently, encoder-only pre-trained models like BERT [35] and GPT [28, 29], were included in MWP solvers to effectively represent background knowledge. In the subsequent sections, we will provide a comprehensive review from these three perspectives.

Problem text representation

To avoid sophisticated feature engineering, deep learning technologies were applied for problem text representation. In this field, Wang et al. [2] have made significant contributions by designing a customized model called Deep Neural Solver (DNS) to automatically solve MWPs. Within the DNS, the problem text and mathematical expressions are represented as sequential data, making them amenable to processing by sequence models commonly used in Natural Language Processing (NLP). Consequently, the task of solving mathematical problems is modeled as a "translation" problem within a Sequence-to-Sequence (Seq2Seq) framework. Following this pioneering work, a number of Seq2Seq models [4, 5, 13, 36, 37] for MWPs have been developed. These Seq2Seq models treat the problem text as a sequence of word tokens and utilize Recurrent Neural Networks (RNNs) such as Long-Short Term Memory (LSTM) network [3], Gated Recurrent Unit (GRU) [47], and Transformer [11] for encoding the word sequence.

To enhance the representation of the problem text, numerous optimization strategies and auxiliary techniques have been proposed. For instance, Wang et al. [4] utilized different deep neural networks for problem encoding and achieved higher accuracy compared to other Seq2Seq models. Shen et al. [33] employed a multi-head attention mechanism to capture both local and global features of the problem text. Li et al. [37] developed a group attention mechanism to extract diverse features pertaining to quantities and questions in

Table 1 The typical models reported in the literature for solving MWPs

No	Model	Encoder	Decoder	Pre-trained Model
1	DNS (Wang et al., 2017) [2]	GRU	LSTM	–
2	MathEN (Wang et al., 2018) [4]	BiLSTM/ConvS2S/Transformer	LSTM/ConvS2S/Transformer	–
3	T-RNN(Wang et al., 2019) [36]	BiLSTM	LSTM	–
4	S-Aligned (Chiang et al., 2019) [13]	BiLSTM	LSTM	–
5	Group-ATT (Li et al., 2019) [37]	BiLSTM	LSTM	–
6	GTS (Xie and Sun, 2019) [5]	GRU	TreeDecoder	–
7	Tree-Dec (Liu et al., 2019) [25]	BiLSTM	TreeDecoder	–
8	GTP-2 (Radford et al., 2019) [29]	–	GPT	GPT-2
9	SAU-Solver (Qin et al., 2020) [38]	GRU	TreeDecoder	–
10	Graph2Tree (Li et al., 2020) [17]	BiLSTM+BiGraphSAGE	TreeDecoder	-
11	Graph2Tree (Zhang et al., 2020) [6]	BiLSTM+GCN	TreeDecoder	–
12	MulltiE&D (Shen et al., 2020) [33]	BiLSTM+GCN	GRU	–
13	NumS2T (Wu et al., 2021) [39]	BiLSTM	TreeDecoder	–
14	EEH-G2T (Wu et al., 2021) [7]	BiLSTM+GCN	TreeDecoder	–
15	RPKHS (Yu et al., 2021) [40]	BERT+GRU	TreeDecoder	RoBERTa
16	BERT-TD (Li et al., 2021) [41]	BERT	TreeDecoder	BERT
17	Gen&Rank (Shen et al., 2021) [42]	BART	BART	mBART2
18	MWP-BERT (Liang et al., 2022) [26]	BERT	TreeDecoder	BERT+RoBERTa
19	DeductiveMWP (Jie et al., 2022) [9]	BERT	D-Reasoner	BERT+RoBERTa
20	LogicSolver (Yang et al., 2022) [8]	BERT	TreeDecoder	MacBERT
21	TE-CL (Shen et al., 2022) [43]	BERT	TreeDecoder	BERT
22	MVR-CCL (Zhang et al., 2022) [27]	BERT+GRU	TreeDecoder+D-Reasoner	BERT+RoBERTa
23	PaLM (Chowdhery et al., 2022) [44]	–	PaLM	PaLM
24	Minerva (Lewkowycz et al., 2022) [45]	–	PaLM	Minerva
25	LLaMA (Touvron et al., 2023) [46]	–	LLaMA	LLaMA
27	GPT-4 (Zhou et al., 2023) [30]	–	GPT	GPT-4

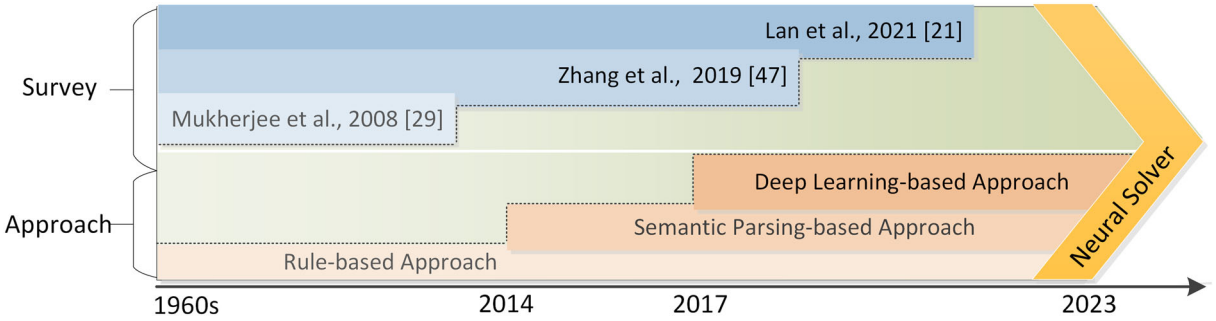

Fig. 1 Approach evolution in solving MWPs

MWPs. These efforts aim to better capture the contextual information in the problem text, thereby improving the efficiency of expression generation.

In addition to capturing the contextual information from the problem text, researchers have explored graph-based models inspired by the success of previous works [20, 21] to capture non-sequential information, such as quantity unit relations, numerical magnitude relations, and syntactic dependency relations. These non-sequential relations are considered helpful in ensuring the logical correctness of expression generations. For instance, quantity unit relationships can help reduce illegal operations between values with different units, and numerical magnitude relationships can help reduce the occurrence of negative results from subtracting a larger number from a smaller number. Based on these assumptions, Zhang et al. propose Graph2Tree [6], which constructs a quantity cell graph and a quantity comparison graph to represent quantity unit relationships and numerical magnitude relationships, respectively. Similarly, Li et al. [17] introduce the constituency tree augmented text graph, which incorporates a constructed graph into a graph neural network [48, 49] for encoding. The output of these graph models, combined with the output of the sequence model, is used for decoding. Additionally, knowledge-aware models [7, 50, 51] have been designed to improve problem representation.

Recently, Pre-trained Language Models (PLMs), and especially transformer-based language models, have shown to contain commonsense and factual knowledge [52, 53]. To enhance the representation of problem texts, PLMs were employed for problem text encoding, aiming to reason through outside knowledge provided by the PLMs. Yu et al. [40] utilized RoBERTa [54] to capture implicit knowledge representations in input problem texts. Li et al. [41] leveraged BERT [35, 55] for both understanding semantic patterns and representing linguistic knowledge. Liang et al. [26] employed BERT and RoBERTa for contextual number representation. These models have yielded significant improvement in terms of answer accuracy. Recently, decode-only PLMs, such GPT [28], PaLM [44, 45] and LLaMA [46], exhibit strong reasoning abilities and their potential in solving MWPs, especially integrated with technologies of prompt [56] and chain-of-thought [57]. For instance, the latest release, GPT4-CSV [30], achieved an almost 20% increase in answer accuracy on the MATH dataset compared to GPT3.5 [28]. However, despite these improvements, issues such as actual errors and reasoning errors [58] by LLMs may lead to wrong answers even with carefully crafted prompt sequences.

Math equation representation

The representation of math equations presents another challenge in the design of MWP solvers. Initially, math equations were commonly modeled as sequences of symbols and operators, known as equation templates [2]. This allowed for direct processing by sequence models such as LSTM, GRU, etc. However, these sequence models suffer non-deterministic transduction [1, 4] as a math word problem can have multiple correct equations. To address this issue, approaches such as MathEN [4] was proposed to normalize the duplicated equations to ensure that each problem text corresponds to a unique math equation. Chiang et al. [13] took it further by utilizing the Universal Expression Tree (UET) to represent math equations. However, these methods encode math equations using sequence models, ignoring the hierarchical structure of logical forms within math equations.

To capture the structural information, researchers have proposed tree-structured models (TreeDecoders) [5, 17, 25] for the iterative construction of equation trees. Liu et al. [25] developed a top-down hierarchical tree-structured decoder (Tree-Dec) inspired by Dong et al. [59]. The Tree-Dec [25] enhances a basic sequence-based LSTM decoder by incorporating tree-based information as input. This information consists of three components: parent feeding, sibling feeding, and previous token feeding, which are then processed by a global attention network. Xie et al. [5] introduced a goal-driven mechanism (GTS) for feeding tree-based information. Li et al. [17] applied a separate attention mechanism to the node representations corresponding to different node types. Additionally, Zhang et al. [27] proposed a multi-view reasoning approach that combines the top-down decomposition

of TreeDecoder with the bottom-up construction of reductive reasoning [9]. Due to its exceptional ability in math equations generation, TreeDecoders has been widely adopted by subsequent MWP solvers [7, 38, 39, 43, 60]. Furthermore, several extensions of TreeDecoders have been explored, such as the generation of diverse and interpretable solutions [7, 38, 39, 60].

The previous survey work

Despite the extensive research conducted in the field of MWP solving, there is a lack of comprehensive reviews. Mukherjee et al. [22] conducted a functional review of various natural language mathematical problem solvers, starting from early systems like STUDENT [61] to lately developed ROBUST [62]. The paper provides a systematic review of representative systems in domains such as math problems, physics problems, chemistry problems, and theorem proving. It highlights that these systems are generally useful for typical cases but have limitations in understanding and representing problems of diverse nature [22]. Additionally, there is a lack of unified benchmark datasets and clear evaluation strategies. However, since the publication date of the paper is early, it does not cover the current mainstream neural network-based methods, which limits its comprehensive assessment of the research field.

With the rise of machine learning-based MWP solving, Zhang et al. [1] conducted a review of these emerging works from the perspective of representation of problem texts and mathematical expressions. The paper categorizes the development of machine-answering techniques into three stages: rule-based matching, statistical learning and semantic parsing, and deep learning. The authors argue that the primary challenge in machine answering is the existence of a significant semantic gap between human-readable words and machine-understandable logic. They focus on reviewing tree-based methods [16, 63–65] and deep learning-based methods [32, 66–69]. The paper also reports the test results of these methods on certain datasets, aiming to provide readers with insights into the technical characteristics and classification of machine answering in the era of machine learning.

In recent literature, Meadows et al. [23] and Lu et al. [24] conducted comprehensive surveys on the emerging deep learning-based models developed for solving math word problems. These studies systematically classify and document the network architectures and training techniques utilized by these models. Furthermore, they provide a detailed analysis of the challenges faced in this field as well as the trends observed in the development of such models. Lan et al. [10] developed MWPToolkit, a unified framework and re-implementation of typical neural solvers [2, 4–6, 13, 19, 33, 36–38]. MWPToolkit provides specified interfaces for running existing models and developing new models. However, there is a lack of technical analysis on the network structures of these neural solvers. Recently, pilot work has been conducted to compare the performance of MWP solvers based on deep learning models. Chen et al. [70] performed a comparative analysis of six representative MWP solvers to reveal their solving performance differences. Building upon this prior work, He et al. [71] further investigated the performance comparison of representation learning models in several considered MWP solvers.

This paper conducts an in-depth and comprehensive comparative analysis to reveal the technical features and performance variations of typical neural solvers when solving MWPs with different characteristics. The goal is to assist researchers in selecting more effective network units and structures for tasks with different features.

Architecture and technical feature analysis of neural solvers

The general architecture of neural solvers

Math word problem solving is a mixed process of reasoning and calculating that can hardly be solved directly by neural networks that are designed for classification or regression tasks. Hence, most of the neural solvers take a two-step solution of expression generation and answer calculation. The former aims to translate the input problem text into a calculable math expression and then be followed by a mathematical solver to calculate the final answer. Therefore, the key challenge of solving a math word problem is to generate the target math expression.

Earlier solvers, such as DNS [2], tackle this challenge by using a seq2seq model in which math expressions are abstracted into expression templates and each template is treated as a sequence with operators and symbols. Later, to improve the capability of new expression generation, math expressions are modeled as decomposable tree structures instead of fixed structures of sequences. A milestone work of tree-structured decomposing is the Graph2Tree model proposed by Xie et al. [5] and this model is widely used in the newly developed neural solvers. Under this Graph2Tree model, the math expression generation is further divided into three sub-steps, including problem modeling, problem encoding and expression decoding as shown in Fig. 2.

Generally, a neural solver can be summarized as an Encoder-Decoder architecture of

$$ME = F_{decoding}(F_{encoding}(P)) \tag{1}$$

where the problem P is consisted by a word token sequence $V = (v_1, v_2, ..., v_n)$ and each w_i denotes the token of word w_i. $F_{encoding}(.)$ and $F_{decoding}(.)$ are networks to obtain the problem text representation and generate math equations

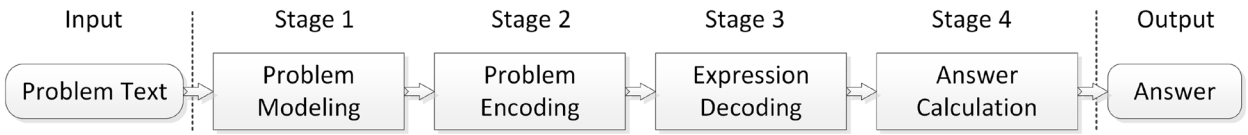

Fig. 2 The general architecture of a neural solver for solving math word problems

accordingly. The goal of building a neural solver is to train an encoding network $F_{encoding}(.)$ for problem feather representation learning, and a decoding network $F_{decoding}(.)$ for predicting math expressions $ME = (e_1, e_2, ..., e_m)$ to achieve the final answer. We give a detailed analysis of the architecture of mainstream encoders and decoders below separately.

Problem modeling. Problem modeling defines the pipeline of neural networks. Specifically, it models the data structure of the input and output of the solvers. For input, the problem texts are usually modeled as word sequences followed by a recursive neural network for feature learning. A huge improved work has been made which converts sequential texts into graphs, hence graph neural networks can be used for feature learning.

The output of the solvers is the target math expression which can be modeled as specially designed sequences composed of operators and number tokens. An expression vocabulary is defined which contains operators (e.g., $+, -, \times, \div$), constant quantities (e.g.,$1, \pi$) and numbers presented by the problem text. Based on the built vocabulary, a math expression can be abstracted as an expression template in which digits are replaced by number tokens of n_i. In recent works, target expressions are represented as expression trees. A basic expression tree contains three nodes of the root, left child and right child. The child node can be a digit or an operator that owns at most two children. By employing this tree-structured decomposing, nearly all types of expressions, even those that did not exist in the training set, can also be constructed.

Problem encoding. Problem encoding is a representation learning module to learn the features from the input problem text. According to the representation learning methods applied, problem encoding can be divided into sequence-based methods and graph-based methods.

Expression decoding. Expression decoding is to train a decoding network to convert features obtained in problem encoding into expression templates. As discussed in ***Problem Modeling***, the expression templates can be number token sequences or trees. Hence, expression decoding can be accordingly divided into sequence-based decoding methods and tree-structured decoding methods.

Answer calculation. A number mapping operation is implemented in the stage of answer calculation after expression templates are obtained by replacing the number tokens n_i back to digits, followed by a mathematical solver to calculate the final answer.

Currently, neural solvers are designed as an *Encoder-Decoder* framework to accomplish the tasks of problem encoding and expression decoding. The early encoder-decoder model refers to Seq2Seq [2], that is, the *Encoder* takes the input problem text as a sequence, and the output expression predicted by the *Decoder* is also a sequence [65]. Later, researchers pointed out that the output expression can be better described as a tree structure, e.g. expression tree [72], equation tree [25], so the Seq2Tree model was proposed. The GTS, a typical Seq2Tree-based model, was proposed by Xie et al. [5], in which the output expressions are transformed as pre-order trees and a goal-driven decomposition method is proposed to generate the expression tree based on the input sequence Furthermore, several works revealed that a math word problem is not only a sequence, but also contains structured information about numeric quantities. To represent the quantity relationships, the graph structure is applied to model the quantities as nodes and relations as edges. By combining with tree-structural decoders, several Graph2Tree-based models are proposed [6, 33] recently.

According to the network components applied in problem encoding (*Encoder*) and expression decoding (*Decoder*), neural network-based MWP solvers can be divided into four major categories: Seq2Seq, Seq2Tree, Graph2Tree and PLM-based model as shown in Table 2.

Seq2Seq is a sequence-to-sequence framework, where both the *Encoder* and *Decoder* are sequence-based networks. The Encoder takes the sequence of word tokens as input and outputs the feature vectors, usually an embedding vector and a hidden state vector. The feature vectors are sent to the Decoder to predict the expression templates. The embedding vector is usually used to predict the current character of operators or number tokens and the hidden state vector records the contextual features of the current character. LSTM [3] and GRU [47] are two commonly used networks in building *Encoders* and *Decoders* [2, 5, 37, 38, 65]. For example, MathEN [65] leverages two LSTM networks as *Encoder* and *Decoder*, while DNS [2] employs an LSTM network and a GRU network as *Encoder* and *Decoder* separately.

Seq2Tree is an improved framework based on the Seq2Seq architecture in which the sequence-based *Decoder* is replaced by a tree-structured network to generate expression trees. As

Table 2 The category of neural solvers

Type	Model	Encoder	Decoder	Reference
Seq2Seq	DNS	GRU	LSTM	[2]
	MathEN	LSTM/GRU	LSTM/GRU	[4]
Seq2Tree	GTS	GRU	TreeDecoder	[5]
	SAU-Solver	GRU	TreeDecoder	[38]
Graph2Tree	Graph2Tree[1]	BiLSTM+GCN	TreeDecoder	[6]
	Graph2Tree[2]	BiLSTM+BiGraphSAGE	TreeDecoder	[17]
PLM-based model	Bert2Tree	BERT	TreeDecoder	[26]
	GPT-4	–	GPT	[30]

discussed above, the tree-structured network is a compound of prediction networks and feature networks, as well as a decision mechanism. For instance, in GTS [5], a prediction network and two feature networks are employed to merge the previous state vectors and to calculate the current state vector. In another work [17], only one feather network is used to accomplish the task of feature merging and calculation.

Graph2Tree combines the advantages of a graph-based encoder and a tree-based decoder in the process of problem encoding and expression decoding. Compared to Seq2Tree, Graph2Tree applies graphs to represent the structural relations among word tokens and digits into a network structure (e.g., graph) to enhance the feature learning during the problem encoding. Various kinds of algorithms have been proposed to construct graphs [6, 7, 17, 39] by modeling the structural relations on both word token level and sentence level.

PLM-based models leverage pre-trained language models to generate intermediate MWP representation and solution. Depending on the type of PLM [58], there are two specific implementations of PLM-based models. The first implementation, represented by encoder-only PLMs like BERT [26, 27], utilizes the PLM as an encoder to obtain the latent representation of the math word problem. This representation is then fed into a decoder, such as a Tree-based decoder, to generate the final mathematical expression. The second implementation, represented by models like GPT [28–30], directly employs Transformer networks for mathematical reasoning, producing the desired results without an explicit separation between encoding and decoding stages. This approach streamlines the process and enhances the efficiency of solving math word problems.

As shown in Table 2, DNS and MahtEN are Seq2Seq models, while GTS is built as a seq2tree structure. The tree-structured decoder designed in GTS is also applied in Graph2Tree[1]. Graph2Tree[1] and Graph2Tree[2] are two graph2tree models but differ in both graph encoding and tree decoding. In the stage of graph encoding, Graph2Tree[1] uses *Quantity Cell Graph* and *Quantity Comparison Graph* to describe the quantity relationships, while Graph2Tree[2] leverages *Syntactic Graph* to present the word dependency and the phrase structure information. In the decoding stage, a pre-order expression tree is generated in Graph2Tree[1], while Graph2Tree[2] employs a hierarchical expression tree to model the output expression.

Problem text encoding

In recent years, a trend in building MWP solvers [1] is to apply deep neural networks to capture the quantity relationships presented by problem texts explicitly and implicitly. The early MWP solvers [2, 65] mainly use sequence-based models, such as LSTM [3], GRU [47], etc., to conduct problem representation learning, in which the problem text is regarded as an unstructured sequence. Recently, graph-based representation learning methods [5, 6, 17] are widely employed to enhance both structured and unstructured information learning, which attracts more and more attention of community researchers. On the other hand, several benchmark datasets with diverse characteristics were released for performance evaluation of the proposed solvers [10]. To reveal the potential effectiveness of presentation learning methods on diverse characteristics of MWPs, a comparative analysis of sequence-based and graph-based representation learning is conducted in this paper.

Sequence-based problem encoding

As a problem is mainly presented by natural language text, the sequence-based recursive neural network (RNN) models [3, 47] are naturally taken to problem representation learning. For example, DNS [2] uses a typical Seq2Seq model for problem representation learning, where words are split into tokens inputted into a GRU module to capture quantity relations. Several follow-up works were proposed by replacing GRU with BiLSTM or BiGRU to enhance the ability of quantity relation learning [7, 14, 15]. To improve the semantic embedding, pre-trained language models, such as GloVe [17], BERT [26], Chinese BERT [55] and GPT [28, 73],

etc., were used to better understand the input problem texts. Besides, to capture more features between problem sentences and the goal, attention modules are employed in several works to extract local and global information. For instance, Li et al. [37] introduced a group attention that contains different attention mechanisms which achieved substantially better accuracy than baseline methods.

In a sequence-based representation learning model, every word of the problem text P is first transformed into the context representation. Given an input problem text $P = \{w_1, w_2, ..., w_n\}$, each word token w_i is vectorized into the word embedding w_i through word embedding techniques such as GloVe [17], BERT[26], etc. To capture the word dependency and learn the representation of each token, the sequence of word embeddings is input into the RNN whose cells can be LSTM [3], GRU [47], etc. Formally, each word embedding w_i of the sequence $E = \{w_1, w_2, ..., w_n\}$ is input into the RNN one by one, and a sequence of hidden states is produced as the output.

For unidirectional encoding, the procedure of problem representation learning can be described as follows:

$$h_i^p = RNN(h_{i-1}^p, w_i) \tag{2}$$

where $RNN(\cdot, \cdot)$ denotes a recursive neural network, h_{i-1}^p denotes the previous hidden state and w_i denotes the current input. Repeat the above calculation from step 1 to n to obtain the final hidden state h_n, which is the result of the sequence-based representation learning. In practice, $RNN(\cdot, \cdot)$ is usually specified as a two-layer LSTM or GRU network.

For bi-direction encoding, BiLSTM or BiGRU is applied to obtain the left vector $\overrightarrow{h_i^p}$ and the right vector $\overleftarrow{h_i^p}$ separately. Finally, the output hidden state h_i^p is calculated as follows:

$$h_i^p = \overrightarrow{h_i^p} + \overleftarrow{h_i^p} \tag{3}$$

To capture different types of features in hidden state h_s^p, attention mechanisms are employed to enhance the related features. For example, Li et at. [37] applied a multi-head attention network following a BiLSTM network. The output of the group attention h_a^p is produced by:

$$h_a^p = GroupAtt(Q, K, V) \tag{4}$$

where Q,K and V denote the query matrix, key matrix and value matrix separately, which are all initialized as h_i^p.

The above process can be replaced by employing a pre-trained language model. As shown in Eq. 5, a pre-trained language model $PLM(.)$ is used to directly map the problem text, denoted as X, to a representation matrix H.

$$H = PLM(X) \tag{5}$$

Graph-based problem encoding

To improve structural information learning, graph-based encoders were applied to represent relationships among numbers, words and sentences, etc. The structural information includes token-level information and sentence-level information. The former is also considered as local information which is constructed from the number comparison relationship (e.g., bigger, smaller), neighborhood relationship between numbers and the associated word tokens, etc. For example (as shown in Fig. 3a), Zhang et al. [6] applied two graphs, including a quantity comparison graph and a quantity cell graph to enrich the information between related quantities. The sentence-level information, in a sense, is the global information that connects local token-level information. A commonly used sentence-level information is the syntactic structure information generated from dependency parsing. As shown in Fig. 3b, to capture the sentence structure information, the dependency parsing and the constituency analysis [17] were applied to construct graphs. Furthermore, Wu et al. [50] proposed a mixed graph, called an edge-labeled graph, to establish the relationship between nodes at both the sentence level and problem level. Once the problem text is represented as a graph, graph networks such as GraphSAGE [21], GCN [74], can be used to learn the node embedding. One of the advantages of using graph representation learning is that external knowledge can be easily imported into the graph to improve the accuracy of problem solving [50].

Different from the sequence-based representation learning methods, the graph-based representation learning methods take important structural information into consideration when encoding. Due to the fact that different researchers construct the graph using different methods, unifying these methods is more complex than unifying the sequence-based representation learning methods. Through the summary and induction of several typical works[6, 7, 16, 17], we divide the procedure of sequence-based representation learning into three steps: node initialization, graph construction and graph encoding.

Graph Construction. The graph construction is a pre-process before graph encoding, which converts the problem P into a graph $G = (V, E)$ aiming at preserving more structural information hidden in P. To this end, elements such as words and quantities are treated as nodes V, and syntactic relationships such as grammatical dependency and phrase structure are modeled as edges E.

To enrich the information during graph construction, several adjacency modeling approaches are proposed to construct graphs according to the relationships of words and numbers in P. For example, in reference [16], a Unit Dependency Graph (UDG) is constructed to represent the relationship between the numbers and the question being

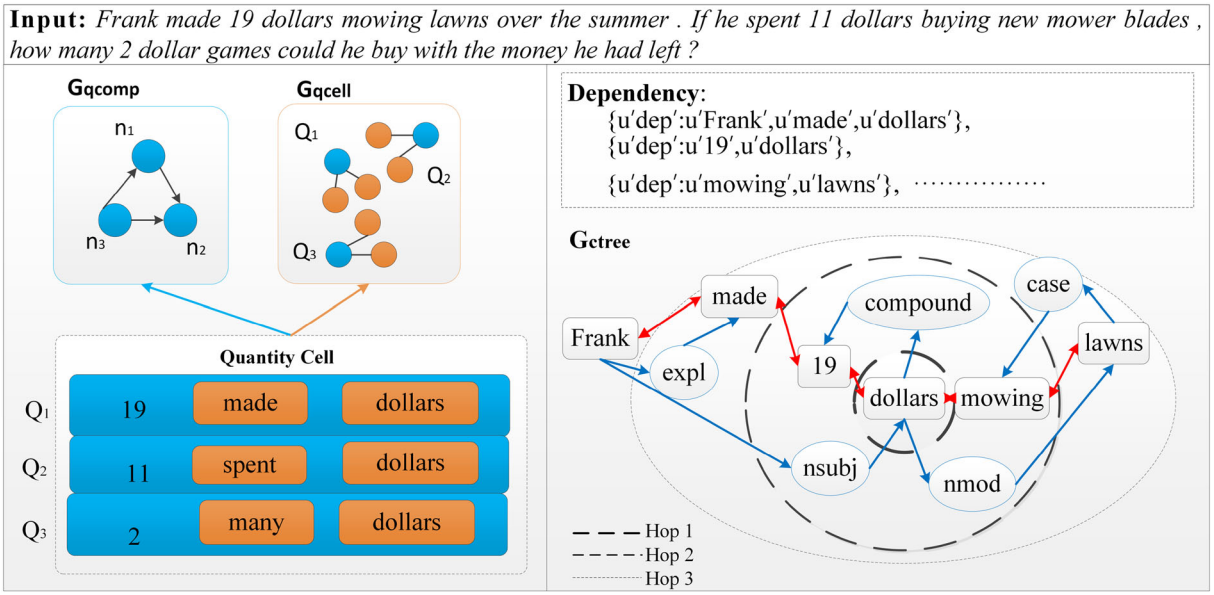

Fig. 3 Comparison of graph-based quantity relation representation. **a** Quantity comparison graph and quantity cell graph designed by Zhang et al. [6]; **b** Constituency tree augmented text graph applied by Li et al. [17]

asked. In work [6], two graphs, including a quantity comparison graph and a quantity cell graph, are built to model the relationships between the descriptive words associated with a quantity. Syntactic constituency information is used to construct the quantity graph in [17]. Through the graph construction process, a set of graph $\mathbb{G} = \{G_1, G_2, ..., G_K\}$ is obtained from problem P for graph encoding.

Graph encoding. After initializing the node and constructing the graph, the graph neural network is applied to obtain the output vector. The procedure can be summarized as follows:

$$h_k^g = GNN(E_k, V_k) \tag{6}$$

where $GNN(\cdot, \cdot)$ denotes a graph neural network, such as GCN [74] or GraphSAGE [21]. The pair (E_k, V_k) represents the k_{th} graph G_k in $\mathbb{G}$, with V_k as the node set and E_k as the edge set. Both V_k and E_k are formed during the node initialization stage. h_k^g denotes the hidden state corresponding to the input graph G_k. When more than one graph $(k > 1)$ is utilized, the output values $h_{k\,k=1}^{g\,K}$ are concatenated and projected to produce the final value H. Finally, the global graph representation h^g can be obtained:

$$h^g = FC(Pooling(H)) \tag{7}$$

where $FC(\cdot)$ is a fully connected network and $Pooling(\cdot)$ denotes pooling function.

Math expression decoding

To achieve the final answer, vectors after problem representation learning are decoded as mathematical expressions followed by a math solver to calculate the answer. Early neural solvers, such as DNS [2], employ a typical Seq2Seq model to predict mathematical expressions. Later, to improve the generation ability of new expressions, tree-based models [5] are proposed to capture the structure information hidden in expressions.

Expression *Decoder* decodes the feature vectors obtained by the problem *Encoder* into expression templates. The decoding process is a step-by-step prediction of number tokens and operators. Therefore, recursive neural networks are naturally chosen for this task. The decoding process can be described as a conditional probability function as follows:

$$p(y_t|y_{<t}, x) = F_{prediction}(h_t) \tag{8}$$

where x denotes vectors of input problems, y_t and h_t is the predicted character and decoder hidden state at step t separately, and $F_{prediction}$ is a non-linear function. The key component of Eq. (8) is the computation of h_t to ensure the output expressions are mathematically correct. Hence, the default activation functions of the general RNNs need to be redesigned. According to the redesigned activation functions, expression decoding can be divided into two main categories: sequence-based decoding and tree-based decoding.

Sequence model based expression decoding

In sequence-based models, expressions are usually abstracted as a sequence of equation templates with number tokens and operators [2, 37, 65]. For example, the expression $x = 5+2*3$ is described as an equation template $x = n_1+n_3+n_2$, n_i is the token of the ith number in problem P. In the stage of math expression generation, a decoder is designed to predict an equation template for each input problem and then expressions are generated by mapping the numbers in the input problem to the number tokens in the predicted equation template [2]. Hence, the math expression generation is transformed into a sequence prediction task and one of the core tasks of math expression generation is to design a decoder to predict the equation templates. Typical sequence models built for NLP tasks can be directly applied for building such decoders [47, 72]. Compared to retrieval models [32, 75], sequence-based models achieved significant improvement in solving problems requiring new equations that not existed in the training set. However, these models are usually sensitive to the length of the expressions as they generate solution expressions sequentially from left to right.

In sequence-based expression decoding, the activation function is defined according to the basic rules of arithmetic operations. For example in infix expressions, if y_{t-1} is a number, then y_t should be a non-number character. Therefore, the redesigned activation function differs according to the infix and suffix expressions used.

In infix sequence models [2], predefined rules are used to decide the type of the t_{th} character according to the $(t-1)_{th}$ character. For example, rule "If y_{t-1} in $\{+, -, \times, \div\}$, then y_t will not in $\{+, -, \times, \div,), =\}$" defines the following character after an operator is predicted. Similar rules are used to determine characters after "(,), =" and numbers are predicted.

In suffix sequence models [36, 37], two numbers will be first accessed by the RNN to determine the operator and generate a new quantity as the parent node. The representation of the parent node o_c can be calculated by a probability function like:

$$P(o_c|h_l, h_r) = \text{softmax}(W_2 \cdot \tanh(W_1([h_l, h_r]] + b))) \quad (9)$$

where h_l, h_r are the quantity representations for the previously predicted nodes, and W_1, W_2 and b are trainable parameters.

Tree-structured model based expression decoding

To describe the structural relation among operators and digits, expression templates are represented as tree structures and tree-structured networks are proposed to learn the structural features of the expression trees. Compared to left-to-right sequential representation in sequence-based methods, relationships among operators and numbers are represented by tree structures, such as expression tree [72] or equation tree [63]. Strictly, the tree-structured network is not a novel network architecture but a compound of networks and a decision mechanism. For example, the previous state when predicting a left child node is the parent node state, but in a right child node prediction, both the parent node state and the left child node state are considered as the previous state [5]. Hence, a decision mechanism is designed to choose different embedding states when predicting a left and right child node. Besides, various neural cells (e.g., a prediction network and a feature network) are usually employed for current character prediction and current hidden state calculation [6, 17].

Therefore, the tree-structured networks are decided by the structure of the expression trees and the tree-based decoding is a decomposing process of an expression tree. According to the decomposing strategy employed, tree-based decoding can be divided into two main categories: depth-first decomposing [5, 6] and breadth-first decomposing [17].

Depth-first decomposing. As shown in Fig. 4b, the depth-first decomposing starts from the root node and implements a pre-order operation during the prediction. As such, if an operator is predicted, then go to predict the left child until a number node is predicted, then go to predict the right child. To make full of available information, the prediction of the right child takes the information of its left sibling node and the parent information into consideration. Roy et al. [72] proposed the first approach that leverages expression trees to represent expressions. Xie et al. [5] proposed a goal-driven tree-structured neural network, which was adopted by a set of latter methods [6, 14, 15], to generate an expression tree.

Breadth-first decomposing. In breadth-first decomposing models, expressions are represented as hierarchically connected coarse equations. A coarse equation is an algebraic expression that contains both numbers and unknown variables. Compared to depth-first decomposing, an essential difference of breadth-first decomposing is that the non-leaf nodes are specified as variables. Therefore, the variable nodes are decomposable nodes that can be replaced by sub-trees. As shown in Fig. 4(c), an example equation is firstly represented as a 1st-level coarse equation $s_1 \div n_3(2) = x$ containing a non-leaf node s_1 and four leaf nodes. Then, the non-leaf node s_1 is decomposed into a sub-tree as the 2nd-level coarse equation of $n_1(19) - n_2(11)$. When all coarse equations are achieved then go to predict the 3rd-level coarse equations if it has, otherwise, the decomposing stops.

To start a tree generation process, the root node vector q_{root} is initialized according to the global problem representation. For each token y in the target word V^{dec}, the representation for a certain token $\mathrm{e}(y \mid P)$, as denoted as h_t in Eq. (8), is defined as follows:

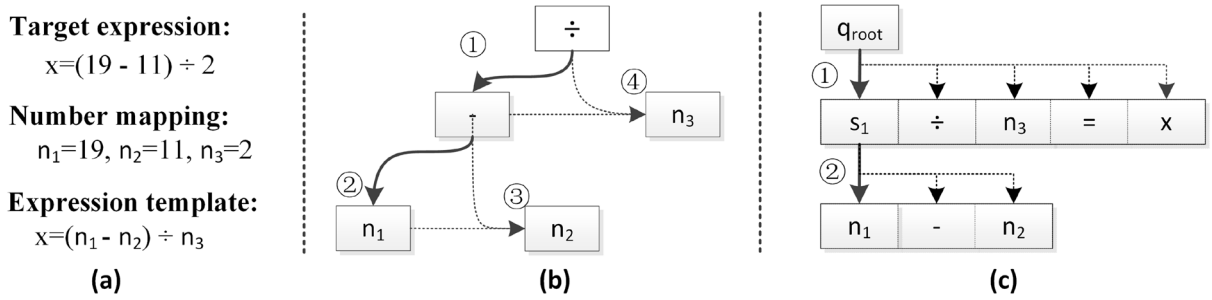

Fig. 4 An example of tree-structured decomposing. **a** Input expression template; **b** Depth-first decomposing; **c** Breadth-first decomposing

$$\mathrm{e}(y \mid P) = \begin{cases} \mathrm{e}_{(y,op)} & \text{if } y \in V_{op} \\ \mathrm{e}_{(y,u)} & \text{if } y \in V_u \\ \mathrm{e}_{(y,con)} & \text{if } y \in V_{con} \\ \bar{h}^p_{loc(y,P)} & \text{if } y \in n_p \end{cases} \tag{10}$$

where $\mathrm{e}_{(y,op)}$, $\mathrm{e}_{(y,u)}$ and $\mathrm{e}_{(y,con)}$ denotes the representation of operators, unknowns and quantities separately that is obtained from 3 independent embedding matrices M_{op}, M_{unk} and M_{con}. $\bar{h}^p_{loc(y,P)}$ is the quantity representation from Eqs. (3) or (4). V^{dec} is the target vocabulary which consists of 4 parts: math operators V_{op}, unknowns V_u, constants V_{con} and the numbers n_p.

In order to adapt to the tree-structured expression generation, activation functions are redesigned according to the types of nodes in the expression tree. The nodes are categorized into two types: leaf nodes and non-leaf nodes. When a non-leaf node is predicted, further decomposing is needed to predict the child nodes. Otherwise, stop the current decomposing and go to predict the right child nodes. The non-leaf node differs in different representations of tree-structured expressions. In regular expression trees [5, 6], the non-leaf nodes are operators while numbers are treated as leaf nodes. While in a heterogeneous expression tree, the non-leaf nodes are non-target variables that are represented by sub-expressions.

Based on the above discussion, the whole procedure of tree-based expression decoding can be summarized as follows [5–7, 14]:

1) Tree initialization: Initialize the root tree node with the global embedding H_g and perform the first level decoding:

$$\hat{q}_{root} = F_{prediction}(H_g) \tag{11}$$

where the global embedding H_g is the original output of the problem *Encoder*.

2) Left sub-node generation: A sub-decoder is applied to derive the left sub-node. The new left child n_l is conditioned on the parent node n_p and the global embedding H_g. The token $\hat{y}_l$ is predicted when generating the new left node:

$$\begin{aligned} n_l &= Embedding(n_p, H_g) \\ \hat{y}_l &= F_{prediction}(n_l, H_g) \end{aligned} \tag{12}$$

If the generated $\hat{y}_l \in V_{op}$ or $\hat{y}_l \in V_u$, repeat step 2). If the generated $\hat{y}_l \in V_{con}$ or $\hat{y}_l \in n_p$, get into step 3).

3) Right-node generation: Different from the left sub-node generation, the right sub-node is conditioned on the left sub-node n_l, the global embedding H_g and a sub-tree embedding t_l. The right sub-node n_r and the corresponding token $\hat{y}_r$ can be obtained as:

$$\begin{aligned} n_r &= Embedding(n_l, H_g, t_l) \\ \hat{y}_r &= F_{prediction}(n_r, H_g) \end{aligned} \tag{13}$$

where the sub-tree embedding t_l is conditioned on the left sub-node token $\hat{y}_l$ and left sub-node n_l. If the $\hat{y}_r \in V_{op}$ or $\hat{y}_r \in V_u$, repeat step 2). If the generated $\hat{y}_r \in V_{con}$ or $\hat{y}_r \in n_p$, stop decomposing and backtrack to find a new empty right sub-node position. If no new empty right nodes can be found, the generation is completed. If the empty right node position still exists, go back to step 2).

In other models [17], step 2) and 3) are combined into a sub-tree generation module in which the token embedding s_t and the corresponding token $\hat{y}_t$ at time t are calculated as follows:

$$\begin{aligned} s_t &= Embedding(y_{t-1}, s_{t-1}, st_{parent}, st_{sibling}) \\ \hat{y}_t &= F_{prediction}(s_t) \end{aligned} \tag{14}$$

where st_{parent} stands for sub-tree node embedding from the parent layer and $st_{sibling}$ is the sentence embedding of the sibling.

Compared to earlier sequence-based decoders which are usually retrieved models, tree-based decoders are generative models that can generate new expressions not existing in the training set. The generation ability lies in the iterative process of tree-structured decomposing as defined in Eq. 10 and the equation accuracy was greatly improved by

using tree-based decoders. Detailed results can be found in Sect. "Experiment".

Characteristics analysis of benchmark datasets

Widely used benchmark datasets

Problem texts and equations are two essential items for neural solver evaluation. The problem text of each example in the dataset is a natural language stated short text that presents a fact and raises a question and the equation is a math expression(s) (e.g., an arithmetic expression, an equation or equations) that can be used to generate the final answer to the question raised by the problem text. A problem text can be stated in any language but most of the widely used datasets are stated in English [32, 76, 77] until Wang et al. [2] released a Chinese dataset Math23K in 2017 which contains 23,161 problems with carefully labeled equations and answers. A brief introduction of the widely accepted benchmark datasets is given as follows and the result of a statistical analysis conducted on the considered datasets is shown in Table 3.

Alg514 is a multiple-equation dataset created by Kushman et al. [32]. It contains 514 algebra word problems from Algebra.com. In the dataset, each template corresponds to at least 6 problems (T6 setting). It only contains 28 templates in total.

Draw1K is a multiple-equation dataset created by Upadhyay et al. [78]. It contains 1000 algebra word problems also crawled and filtered from Algebra.com.

Dolphin18K is a multiple-equation dataset created by Huang et al.[77]. It contains 18,711 math word problems from Yahoo! Answers with 5,738 templates. It has much more and harder problem types than the previous datasets.

MAWPS-s is a single-equation dataset created by Koncel-Kedziorski et al. [76]. It contains 3320 arithmetic problems of different complexity compiled from different websites.

SVAMP is a single-equation dataset created by Patel et al. [79]. It contains 1000 problems with grade levels up to 4. Each problem consists of one-unknown arithmetic word problems which can be solved by expressions requiring no more than two operators.

Math23K is a single-equation dataset created by Wang et al. [2]. It contains 23, 162 Chinese math word problems crawled from the Internet. Each problem is labeled with an arithmetic expression and an answer.

HMWP is a multiple-equation dataset created by Qin et al. [38]. It contains 5491 Chinese math word problems extracted from a Chinese K12 math word problem bank.

Despite the available large-scale datasets, neural solver evaluation is still a lot trickier for the various types and characteristics of math word problems. As almost all neural solvers predict equation templates directly from the input problem text, the complexity and characteristics of the equations and the input problem texts need further study to make the evaluated results more elaborate.

Characteristics analysis

To evaluate the neural solvers, three widely used benchmark datasets include two English datasets *MAWPS-s* and *SWAMP*, and a Chinese dataset *Math23k*. All the selected datasets are single-equation problems as almost all solvers support the single-equation generation task. Conversely, not all solvers support the multi-equation generation task which may lead to poor comparability.

As discussed in Sect. "Characteristics analysis of benchmark datasets", problem texts and expressions differ greatly in terms of scope and difficulty between different datasets. In order to reveal the performance difference of neural solvers on datasets with different characteristics, the selected benchmark datasets are categorized into several sub-sets based on four-index characteristic factors of L, H, C and S defined as follows:

- Expression Length (L): denotes the length complexity of the output expression. L can be used as an indicator of the expression generation capability of a neural solver. According to the number of operators involved in the output expression, L is defined as a three-level indicator containing L_1, L_2 and L_3. L_1 level: $l < T_0$, L_2 level: $T_0 <= l <= T_1$, L_3 level: others. Where l represents the number of operators in the output expression. T_0, T_1 denote the thresholds of l at different levels of length complexity.
- Expression Tree Depth (H): denotes the height complexity of the output expression tree. H is another generation capability indicator, especially for tree-structured neural solvers. According to the depth of the expression tree, H is defined as a two-level indicator containing H_1 and H_2. H_2 level: $h < T_2$, H_3 level: others. Where h refers to the height of the expression tree, T_2 is a threshold.
- Implicit Condition (C): denotes whether implicit expressions needed to solve the problem are embedded in the problem text. C_1 refers to problems with no implicit expression, while C_2 refers to problems with one or more implicit expressions. C can be used as an indicator associated with the relevant information understanding of the solver.
- Arithmetic Situation (S): denotes the situation type that a problem belongs to. The different arithmetic situation indicates different series of arithmetic operations. Based

Table 3 Statistics of widely used benchmark datasets

Dataset	Lang	#Probs	#Temps	#Avg PL	#Avg EL	Problem Type	Task
Alg514	en	514	28	41	9.7	Algebra, linear	Multiple equations generation
Draw1K	en	1000	232	35	6.5	Algebra, linear	Multiple equations generation
Dolphin18K	en	18,460	5871	20	9.2	Algebra, linear+nonlinear	Multiple equations generation
MAWPS-s	en	1987	39	27	4.6	Algebra, linear	Single equation generation
SVAMP	en	1000	26	31	1.24	Arithmetic, linear	Single equation generation
Math23K	cn	23161	2187	58	5.6	Arithmetic, linear	Single equation generation
HMWP	cn	5470	2779	62	10.7	Algebra, linear+nonlinear	Multiple equations generation

Probs, Temps, Avg PL and Avg EL represent the number of problems, number of templates, the average number of tokens in problem text and the average length of equation separately

on Mayer's work, we divide math word problems into five typical types which are $Motion$ (S_m), $Proportion$ (S_p), $Unitary$ (S_u), $InterestRate$ (S_{ir}), and $Summation$ (S_s). S can be used as an indicator associated with the context understanding of the solver.

Each of the selected benchmark datasets is divided into three sub-sets of train-set (80%), valid-set (10%) and test-set (10%). These sub-sets are further characterized according to the above four indices. Tables 4 and 5 show the percentage of problems of different benchmark datasets on the four indices for training and testing separately. Compared to Math23K, expressions in MAWPS-s and SVAMP are much more simple on both factors of the expression length and expression depth. Hence, we set different thresholds for T_i to generate L_* and H_* subsets. Moreover, implicit expression and problem situation analysis are only implemented on Math23K dataset.

Experiment

Experimental setup

Selected typical neural solvers: To ensure the fairness of the performance evaluation, two representative solvers were selected from each framework as shown in Table 2. The selected solvers are listed below:

DNS [2]: The first Seq2Seq model using a deep neural network to solve math word problems. The model combines the RNN model and the similarity-based retrieval model. If the maximum similarity score returned based on the retrieval model is higher than the specified threshold, the retrieval model is then selected. Otherwise, the Seq2Seq model is selected to solve the problem.

MathEN [4]: The ensemble model that combines three Seq2Seq models uses the equation normalization method, which normalizes repeated equation templates into expression trees.

GTS [5]: A tree-structured neural model based on the Seq2Tree framework to generate expression trees in a goal-driven manner.

SAU-Solver [38]: A semantically aligned universal tree structure solver based on the Seq2Tree framework, and it generates a universal expression tree explicitly by deciding which symbol to generate according to the generated symbols' semantics.

Graph2Tree [6, 17]: Graph2Tree[1] and Graph2Tree[2] are both deep learning architectures based on the Graph2tree framework, combining the advantages of a graph-based encoder and a tree-based decoder. However, the two differ in graph encoding and tree decoding.

Bert2Tree [26]: An MWP-specific large language model with 8 pre-training objectives designed to solve the number representation issue in MWP.

GPT-4 [30]: A decoder-only large language model released by OpenAI in March, 2023.

The above selected typical solvers are evaluated in solving characteristic problems on five benchmark datasets and the detailed results can be found in Sect. "Performance on solving characteristic problems".

Component Decoupling According to the discussion in Sect. "Architecture and technical feature analysis of neural solvers", each solver consists of an encoder which can be decomposed into one or more basic RNN or GNN cells. To identify the contribution of these various cells during the problem solving, we decouple the above considered solvers into individual components. The decoupled components can be integrated into different solvers and can be replaced by other similar components. Components decoupled from encoders are listed as follows.

LSTM Cell: A long-short term memory network derived from sequence-based encoders for non-structural problem text encoding.

Table 4 The percentage of L and H problems for training and testing

		Dataset	# Problems	Expression length			Expression depth	
				L_1	L_2	L_3	H_1	H_2
Training set	Single equation	MAWPS-s	1589	61.5	36.8	1.6	86.4	13.8
		SVAMP	800	76.3	23.7	–	95.0	5.0
		Math23K	18,592	23.8	48.0	28.2	76.2	23.8
	Multi-equation	Draw1K	600	30.5	69.5	–	65.5	34.5
		HMWP	4392	31.5	68.5	–	28.4	71.6
Testing Set	Single equation	MAWPS-s	199	58.8	39.2	2.0	86.4	13.6
		SVAMP	100	76.0	24.0	–	95.0	5.0
		Math23K	2317	23.7	46.6	29.7	75.8	24.2
	Multi-equation	Draw1K	200	25.0	75.0	–	64.0	36.0
		HMWP	550	30.9	69.1	–	25.6	74.4

Table 5 The percentage of C and S problems for training and testing on Math23k

	# Problems	Implicit condition		Arithmetic situation				
		C_1	C_2	S_p	S_u	S_{ir}	S_s	S_m
Training set	18592	86.9	13.1	22.8	18.5	4.0	24.3	30.4
Testing set	2317	85.2	14.8	22.8	18.5	4.0	24.3	30.4

GRU Cell: A gated recurrent unit derived from sequence-based encoders for non-structural problem text encoding.

BERT Cell: A pre-trained language model used to directly map the problem text into a representation matrix for generating the solution.

GCN Cell: A graph convolution network derived from graph-based encoders for structural problem text encoding.

biGraphSAGE Cell: A bidirectional graph node embedding module derived from graph-based encoders for structural problem text encoding.

The LSTM cell and GRU cell take text sequence as input and output two text vectors including an embedding vector and a hidden state vector. The GCN cell and biGraphSAGE cell take the adjacency matrix as input and output two graph vectors. Similarly, components decoupled from decoders are listed below.

DT Cell: A depth-first decomposing tree method derived from Graph2Tree[1] for math equation decoding. DT cell takes an embedding vector and a hidden vector as input and output a math equation.

BT Cell: A breadth-first decomposing tree method derived from Graph2Tree[2] for math equation decoding. The BT cell takes three vectors as input, including one embedding vector and two hidden state vectors.

Hence, a super solver is developed to reproduce the selected typical solvers and design new solvers by redefining the combination of the decoupled components. The performance of newly developed solvers are shown and discussed in Sects. "Comparative analysis of math expression decoding models" and "Comparative analysis of problem encoding models" separately.

Evaluation Metrics Math word problems used for neural solver evaluation are usually composed of problem texts, equations and answers. Neural solvers take the problem texts as input and output the expression templates which are further mapped to calculable equations [1]. These generated equations are then compared with the equations labeled in datasets for algorithm performance evaluation. Besides this equation-based evaluation, answer-based evaluation is also used in cases where multiple solutions exist. The answer-based evaluation compares the answers calculated from the generated equations with labeled answers. Several commonly used evaluation metrics are introduced below, including accuracy (E_{acc} and A_{acc}), time cost ($\#Time$) and minimum GPU memory capacity ($\#G{-}Mem$).

– *Accuracy.* Accuracy includes answer accuracy and equation accuracy. Answer accuracy [2, 5, 7] is perhaps the most common evaluation method. It simply involves calculating the percentage of final answers produced by the model that is correct. This is a good measure of the model's overall performance, but it can be misleading if the dataset is unbalanced (e.g., if there are more easy problems than difficult ones). Equation accuracy [6, 17] is another important measure, which refers to the accuracy of the solution that the model generates. This is typically calculated by comparing the output of the model to the correct solution to the problem, and determining whether

they match. Evaluating both the solution accuracy and answer accuracy can give a more complete picture of the model's performance on MWP solving tasks.

The ***Equation Accuracy*** (E_{acc}) which is computed by measuring the exact match of predicted equations and ground-truth equations as follows:

$$E_{acc} = \frac{\#Problems\ of\ matched\ equations}{\#Total\ problems} \tag{15}$$

Similarly, the ***Answer Accuracy*** (A_{acc}) is defined as follows:

$$A_{acc} = \frac{\#Problems\ of\ matched\ answers}{\#Total\ problems} \tag{16}$$

To remove extraneous parenthesis during equation matching, equations are transformed into equation trees as described in [17]. By using E_{acc}, outputs with correct answers but incorrect equations are treated as unsolved cases.

Time Cost ($\#Time$): denotes the time required for model training. Specifically, this article refers to the time needed for the model to complete 80 iterations with a batch size of 64.

Minimum GPU Memory Capacity ($\#G{-}Mem$): represents the minimum GPU memory capacity required for training the model. This metric is crucial for assessing the hardware requirements of model training, particularly for researchers with limited resources.

Hyper-parameters To improve the comparability of the experimental results, the hyper-parameters of the selected solvers and the decoupled cells are consistent with the original models. For example, the default LSTM and GRU cells are initialized as a two-layer network with 512 hidden units to accommodate the pre-trained word vector which usually has a dimension size of 300. In the biGraphSAGE cell, we set the maximum number of node hops K as $K = 3$ and the pooling aggregator is employed. As to the optimizer, we use Adam with an initial learning rate of 0.001, and the learning rate will be halved every 20 epochs. We set the number of epochs to 80, batch size to 64, and dropout rate to 0.5. At last, we use a beam search with beam size 5 in both the sequence-based cells and tree-based cells. To alleviate the impact of the randomness of the neural network models, we conduct each experiment 5 times with different random seeds and report the average results. All these hyper-parameters have been carefully selected to balance computational efficiency with model performance.

Experimental Environment Our models run on a server with Intel i7 CPU. The GPU card is one NVIDIA GeForceRTX 3090. Codes are implemented in Python and PyTorch 1.4.0 is used for matrix operation. We use stanfordcorenlp to perform dependency parsing and token generation for Chinese datasets.

Performance comparison

Overall performance of considered solvers

In this section, we initially present the learning curves of all considered models (excluding GPT-4) on two representative datasets (Math23k for Chinese and MAWPS-s for English).

It is evident that overfitting occurs on the MAWPS-s dataset, as shown in Fig. 5. After 10 iterations, the models exhibit oscillations in terms of accuracy despite having low loss values. This suggests that overfitting has occurred in this case. The limited size of the MAWPS-s dataset, which contains only around 1500 training examples, is likely insufficient for effective training of most deep neural networks. On the other hand, the situation improves on the Math23K dataset. After approximately 30 iterations, both the loss and accuracy stabilize.

We have also examined the training time required for different models. As shown in Table 6, without considering GPT-4 and fine-tuning of BERT, all models have completed training with a batch size of 64 within 4 min ($\#Time$), and have reached convergence in less than 80 iterations. Similarly, we have reported the minimum GPU memory capacity ($\#G{-}Mem$) required for training. This has been highly attractive for individual researchers as it has allowed them to quickly train the desired models locally without incurring high costs. The next step will be to evaluate the solving performance of different models.

We provide an overall result of the considered solvers in terms of both single-equation and multi-equation tasks. We evaluate the E_{acc} and A_{acc} separately. Additionally, we report the average training time (#Time(minutes per epoch)) on Math23K.The detailed results are shown in Table 6.

As shown in Table 6, PLM-based models exhibit superior performance in terms of A_{acc} compared to other models. Specifically, without any additional prompts, GPT-4 achieves the best results on the MAWPS-s, SVAMP, and Draw1k datasets, with accuracies of 94.0%, 86.0%, and 42.1% respectively. On the other hand, BertTree performs well on the two Chinese datasets, Math23k and HMWP, with accuracies of 84.2% and 48.3% respectively. This demonstrates the significant advantage of PLM-based models, especially large-scale language modes such as GPT-4, in solving math word problems.

However, it is important to note that there is still room for improvement in the performance of all models, particularly in solving more complex math problems such as those in the Math23k, Draw1K, and HMWP datasets. There is still a considerable gap between the current performance levels and

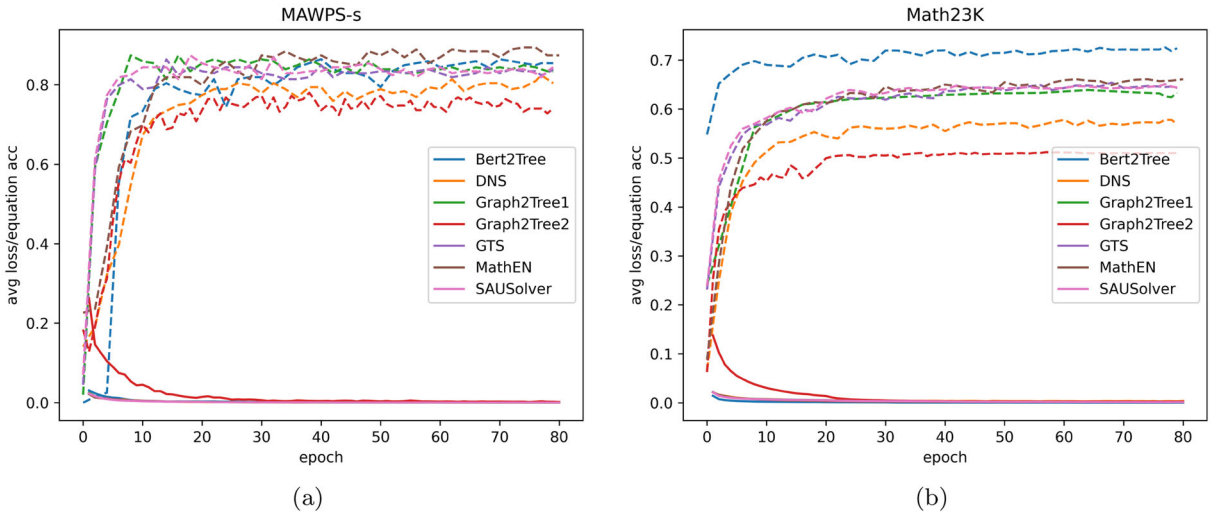

Fig. 5 Learning curves on different datasets. **a** Learning curves on MAWPS-s; **b** Learning curves on Math23K

Table 6 The overall performance of considered solvers on different datasets

Model	Single-Equation Dataset								Multi-Equation Dataset			
	MAWPS-s		SVAMP		Math23K				Draw1K		HMWP	
	E_{acc}	A_{acc}	E_{acc}	A_{acc}	E_{acc}	A_{acc}	$\#Time$	$\#G{-}Mem$	E_{acc}	A_{acc}	E_{acc}	A_{acc}
DNS	78.9	86.3	22.1	24.2	57.1	67.5	< 1 min	4 G	35.8	36.8	24.0	32.7
MathEN	**85.9**	86.4	21.8	25.0	66.7	69.5	< 2 min	4 G	38.2	39.5	32.4	43.7
GTS	83.5	84.1	25.6	29.1	63.4	74.2	< 3 min	4 G	38.6	39.9	33.7	44.6
SAU-Solver	83.4	84.0	27.1	29.7	64.6	75.1	< 4 min	4 G	38.4	39.2	33.1	43.7
Graph2Tree[1]	84.9	85.6	31.6	35.0	64.9	75.3	< 3 min	8 G	**39.8**	41.0	34.4	45.1
Graph2Tree[2]	60.2	61.5	**68.0**	70.2	47.7	58.1	< 4 min	8 G	–	–	–	–
Bert2Tree	84.4	84.9	44.0	46.0	**72.4**	**84.2**	< 4 min	12 G	39.2	39.4	**39.6**	**48.3**
GPT-4	–	**94.0**	–	**86.0**	–	75.4	–	$\gg$ 12 G	–	**42.1**	–	38.5

practical requirements. Additionally, traditional lightweight models also have their merits. For instance, models utilizing Tree-based decoders achieve leading performance in terms of E_{acc}, with 68.0%, 72.4%, 39.8%, and 39.6% on the SVAMP, Math23K, Draw1K, and HMWP datasets respectively. This highlights the potential advantages of Tree-based decoders in representing mathematical expressions. Furthermore, the resource requirements and response efficiency of large language models like GPT are also important considerations.

Among the lightweight models, Graph2Tree models demonstrate the best results on most selected datasets, particularly for multi-equation tasks on Draw1K and HMWP. This underscores the immense potential of the graph-to-tree framework in solving math word problems. However, we observed that Graph2Tree[2] did not perform as well as Graph2Tree[1], underscoring the significance of careful cell selection in both problem encoding and expression decoding steps. Detailed analysis can be found in Sects. "Comparative analysis of math expression decoding models" and "Comparative analysis of problem encoding models". Surprisingly, MathEN achieved the best performance on MAWPS-s and also outperformed other solvers in terms of E_{acc} on the Math23K dataset.

Based on the training time required per epoch on Math23K, we found that more complex models resulted in higher computational times, which is consistent with our general understanding. Among them, SAU-Solver and Graph2Tree[2] had the longest training times, ranging from 3 to 4 min, while the DNS model, which only involves sequence encoding and decoding, had the shortest training time. Graph2Tree[1] and GTS reported similar time costs, indicating that the added CNN unit in Graph2Tree[1] has a minor impact on the computation cost of the graph-to-tree framework.

Performance on solving characteristic problems

In the following comparison, we only considered single-equation tasks for performance evaluation. This is because

multi-equation tasks can be easily converted into a single-equation task by adding a special token $< bridge >$ to convert the equations into a single tree or equation [10], without requiring any model modifications. Therefore, the performance of solvers on single-equation tasks is also useful for evaluating the performance of models on multi-equation tasks.

(1) Performance on solving problems indicated by expression length.

Table 7 presents the comparison results of E_{acc} in solving problems with varying equation lengths. Mean Accuracy Difference (MAD), denoted as d_{i-k}, is used to indicate the accuracy difference in solving problems from level L_i to L_k. Due to the difficulty of obtaining annotated standard arithmetic expressions for GPT models, we utilize the A_{acc} as a reference instead.

As depicted in Table 7, PLM-based models have demonstrated superior performance compared to other models. Bert2Tree, in particular, has exhibited greater stability when compared to GPT-4. Specifically, its accuracy remains relatively consistent in both L_1 and L_2 level problems, with only a slight decrease of 17.2% in the L_3 task. In contrast, GPT-4 experiences a significant decrease of 54.9% from L_1 to L_3.

Graph2Tree models performed the best on SVAMP, achieving an average E_{acc} of 49.0% and 68.0%, respectively. Graph2Tree[1] proved better at solving long equation problems. For example, on Math23k, it achieved state-of-the-art performance of 73.8% and 48.0% on solving L_2 and L_3 level problems, respectively. Similar results were obtained for both SVAMP and MAWPS-s, highlighting the potential of graph-based models in solving problems of varying length complexities.

For Seq2Tree models, GTS and SAU-Solver separately achieved an average improvement of 7.1% and 6.1%, respectively, compared to DNS on MAWPS-s. On Math23k, GTS achieved an average improvement of 7.4% compared to DNS, and SAU-Solver achieved an 8.3% improvement. The considerable improvements by the Seq2Tree models indicate their potential for equation representation learning using tree-structured decoders.

Surprisingly, MathEN achieved the highest problem-solving accuracy on the L_1-level task of MAWPS-s and the L_3-level task of Math23K, and also demonstrated a lower MAD value. On the other hand, DNS exhibited lower problem-solving accuracy than MathEN, and had higher MAD values, indicating that DNS is sensitive to the lengths of the expressions.

Among the four categories of solvers considered, PLM-based models demonstrated the best performance on L_1,L_2 and L_3 level tasks across all datasets. Notably, Graph2Tree exhibited advantages over other lightweight models specifically in handling tasks at L_2 and L_3 levels. Furthermore, it is worth highlighting that among the lightweight models, MathEN and SAU-Solver obtained the best results for L_1 on MAWPS-s and Math23K, respectively. This could be due to the fact that L_1 level tasks typically involve very simple syntax and language structure, which can be efficiently modeled by sequence-based encoders. In contrast, L_2 and L_3 level tasks involve more complex syntax and language structures.

Another interesting finding is that, on MAWPS-s, all models performed better at solving L_2 level problems than the shorter L_1 level problems except GPT-4. Further analysis showed that the average token length of L_1 level problems was 26, which is significantly shorter compared to 31 and 21 on SVAMP and Math23k, respectively. It should be noted that each word is treated as a token for the English dataset MAWPS-s and SVAMP, while for the Chinese dataset Math23k, each token contains one or more words depending on the output of the applied tokenizers.

(2) Performance on solving problems indicated by expression tree height.

Table 8 shows the performances on characteristics of depth complexity. Overall, the accuracy of models decreases as the expression tree depth increases. In particular, GPT-4 achieves accuracies of 97.0% and 74.5% on the H_1 and H_2 subsets of MAWPS-s respectively. However, there is a significant performance drop of 30% from H_1 to H_2. Similar reduction in performance is observed on Math23K. This suggests that GPT-4 is highly sensitive to the depth of the expressions.

For Graph2Tree models, an average accuracy reduction d_{2-1} by 15% and 10% for all models from H_1 to H_2 level problems on MAWPS-s and Mathe23K separately. d_{2-1} is an indicator of model robustness that the lower of d_{2-1} is, the more robustness of the model is. This suggests that capturing the structure information hidden in problem texts is challenging work for both sequence-based and graph-based methods.

Seq2Tree models have a 7% to 10% improvement on Math23k and MAWPS-s separately compared to DNS. SAU-Solver performs better than MathEN on MAWPS-s but worse on Math23k. Graph2Tree models perform better on Math23k than Seq2Tree models. However, Graph2Tree[1] performs equal or better on H_2 level problems compared to all other methods, which indicates the latency of problem learning on complexity structures. Unlike Graph2Tree[1] and others, Graph2Tree[2] is much more insensitive for the task of depth expression prediction. This suggests that the sentence level information might enhance the representation learning of complex expressions.

For Seq2Seq models, MathEN performs better on all datasets compared to DNS, especially on Math23k H_1 level dataset which achieves the best result (69.2%). However, the accuracy reduction of MathEN by 15.6% and 12.9 from H_1 to H_2 level problem on MAWPS-s and Math23k separately

Table 7 Performance comparisons of solvers in solving problems with different expression lengths

Category	Solver	MAWPS-s			SVAMP			Math23K				
		L_1	L_2	d_{2-1}	L_1	L_2	d_{2-1}	L_1	L_2	L_3	d_{2-1}	d_{3-1}
Seq2Seq	DNS	71.8	87.2	+15.4	36.8	8.3	−28.5	71.8	62.0	32.6	−9.8	−39.2
	MathEN	84.6	87.2	+2.6	43.4	12.5	−30.9	77.8	72.0	48.0	−5.8	−29.8
Seq2Tree	GTS	83.8	87.2	+3.4	39.5	–	–	77.1	68.8	42.5	−8.3	−34.6
	SAU-Solver	82.5	86.7	+4.2	30.3	–	–	78.0	69.1	42.4	−8.9	−32.6
Graph2Tree	Graph2Tree[1]	82.9	88.5	+5.6	60.5	12.5	−48.0	72.2	73.8	48.0	1.6	−24.2
	Graph2Tree[2]	53.0	70.7	+17.7	76.3	41.7	−34.6	59.3	55.6	25.8	−3.7	−33.5
PLM-based Model	Bert2Tree	84.6	**89.7**	+5.1	52.7	25.0	−27.7	89.1	**89.4**	**71.9**	+0.3	−17.2
	GPT-4	**96.2**	85.0	−11.2	**96.7**	**52.1**	−44.6	**92.7**	72.2	37.8	−20.5	−54.9

Table 8 Performance comparisons of solvers in solving problems with different expression depths

Category	Solver	MAWPS-s			SVAMP			Math23K		
		H_1	H_2	d_{2-1}	H_1	H_2	d_{2-1}	H_1	H_2	d_{2-1}
Seq2Seq	DNS	77.3	70.4	− 6.9	31.6	–	–	58.1	47.8	− 10.3
	MathEN	86.0	70.4	− 15.6	35.8	–	–	69.2	56.3	− 12.9
Seq2Tree	GTS	86.6	63	− 23.6	31.6	–	–	65.6	54.5	− 11.1
	SAU-Solver	87.2	74.1	− 13.1	24.2	–	–	66.1	54.3	− 11.8
Graph2Tree	Graph2Tree[1]	85.5	70.4	− 15.1	51.6	–	–	68.4	57.4	− 11.0
	Graph2Tree[2]	59.9	55.6	− 4.3	69.5	–	–	49.6	41.1	− 8.5
PLM-based Model	Bert2Tree	88.4	64.0	− 24.4	48.4	–	–	86.2	**77.5**	− 8.7
	GPT-4	**97.0**	**74.5**	− 22.5	**88.9**	–	–	**80.9**	44.0	− 36.9

show that MathEN is much more sensitive to expression depth than DNS.

(3) Performance on solving problems indicated by implicit condition

Table 9 demonstrates the significant advantages of PLM-based models in solving implicit condition problems. In particular, GPT-4 exhibits a 1% performance improvement on C_2 compared to C_1. In terms of lightweight models, MathEN and Graph2Tree[1] obtained an outstanding performance of 67.1% and 66.4% separately. For solving implicit relation problems, MathEN achieved 61.5% which is 2.3% higher than the second-highest result obtained by GTS. Meanwhile, it shows that Seq2Tree models and Graph2Tree[1] method performed similarly (i.e., 59.2%, 58.6% and 58.0% separately) on solving implicit relation problems. For robustness, MathEN has the lowest d_{2-1} performance among the considered models except for the Graph2Tree[2].

(4) Performance on solving problems indicated by the arithmetic situation.

As depicted in Table 10, PLM-based models achieve the best results across all four problem types. However, for $Summation$ type problems (S_s), MathEN achieves an impressive accuracy rate of 72.2%, which is 30% higher than that of GPT-4. Among all lightweight models, MathEN exhibits outstanding performance. For example, MathEN achieved 72.2% accuracy on solving S_s type problems and GTS got 71.3% on solving $Motion$ type problems (S_m). Whereas the Graph2Tree models generally performed poorly on situational problems. Since situational problems contain more complex quantity relations among various math objects. These quantity relations are usually indicated by high-level contexts which are much more challenging to obtain the required quantity relations for most sequence and graph based models. Moreover, performance differs greatly on different types of situation problems even with the same model which indicates that differentiated models are required for solving different types of situational problems.

Conclusively, the experimental results revealed the following: (1) PLM-based models have shown a significant advantage over other models in almost all tasks. However, they also suffer from a rapid decline in performance when the length and depth of expressions increase. (2) Tree-based expression decoders achieved significant improvement compared to sequence-based decoders. This demonstrates the efficiency of generative models in learning the structural information hidden in mathematical expressions, compared to traditional retrieved models. (3) For encoders, graph-based models perform similarly to sequence-based models.

Table 9 Performance comparisons of solvers in solving problems with implicit conditions on Math23K

Implicit condition	Seq2Seq		Seq2Tree		Graph2Tree		PLM-based Model	
	DNS	MathEN	GTS	SAU-Solver	Graph2Tree[1]	Graph2Tree[2]	Bert2Tree	GPT-4
C_1	56.7	67.1	63.6	64.1	66.4	48.2	**84.4**	72.0
C_2	49.6	61.5	59.2	58.6	58.0	44.3	**82.8**	73.0
d_{2-1}	− 7.1	− 5.6	− 4.4	− 5.5	− 8.4	− 3.9	− 1.7	+1.0

Table 10 Performance comparisons of solvers in solving problems with different arithmetic situations on Math23K

Category	Solver	Math23K				
		S_p	S_u	S_{ir}	S_s	S_m
Seq2Seq	DNS	50.6	54.5	47.9	55.5	63.0
	MathEN	57.0	65.7	60.4	**72.2**	68.2
Seq2Tree	GTS	60.5	62.0	62.5	63.0	71.3
	SAU-Solver	61.6	63.8	62.5	64.8	68.8
Graph2Tree	Graph2Tree[1]	52.9	58.2	58.3	63.7	67.6
	Graph2Tree[2]	48.3	45.1	47.9	49.5	54.4
PLM-based Model	Bert2Tree	**70.6**	67.3	80.6	66.7	79.0
	GPT-4	59.4	**94.8**	**85.9**	40.6	**79.4**

There may be two reasons for this observation. First, current sequence and graph-based models may have encountered a technical bottleneck. These models are trained and fine-tuned for general or specific natural language tasks that are not necessarily suitable for learning mathematical relations in sophisticated situations of math word problems containing various common sense knowledge and domain knowledge. (4) Equation normalization and ensemble models (such as MathEN) achieved outstanding performance compared to pure Seq2Seq models. Since a math word problem may have more than one equation, it is necessary to normalize duplicated equations when working with current sequence and graph-based models.

Comparative analysis of math expression decoding models

To investigate the impact of the decoding models integrated with different encoding models, we conduct a confusion test of depth-first tree decoding (DT Cell) [6] and breadth-first tree decoding (BT Cell) [17] in this section. For each decoding module, encoding modules of GRU cell [2], GCN cell [6], biGraphSAGE cell [17] and BERT [26] are connected separately to compose a full pipeline of encoding-decoding network. The test is conducted on Math23k and the corresponding result is shown in Table 11.

As shown in Table 11, the DT cell demonstrates significant advantages in terms of accuracy for both expression and answer when combined with any encoding model. Particularly, when using GRU, GRU+GCN, and BERT as encoders, the DT cell outperforms the BT cell by more than 10%. However, when utilizing GRU+biGraphSAGE as the encoder, the DT cell shows lower performance improvements of 6.7% and 6.5% for both E_{acc} and A_{acc}, compared to other encoder combinations. One possible reason is that the GRU+biGraphSAGE encoder incorporates heterogeneous graph information from the problem text, which has relevance to breadth-first decomposing.

Comparative analysis of problem encoding models

Experimental results obtained in Sect. "Performance on solving characteristic problems" show the effectiveness of tree-structured models in math expression decoding. However, the performance of models in solving different characteristic problems varies as different neural cells are applied during encoding. In this section, a further experiment is conducted to evaluate the performance of different encoding units in solving different characteristic problems.

In the following experiments, we split the encoding modules into composable neural cells from the baseline methods for problem encoding. From sequence-based encoders, LSTM and GRU cells are obtained and GCN and biGraphSAGE cells are obtained from graph-based encoders. The obtained GRU and LSTM cells are designed with 2 layers. In our experiment, both 2 and 4-layer cells are tested to evaluate the effect of cell depth in problem encoding. The above cells are implemented individually or jointly for problem encoding followed by the tree-based decoding module [5] to generate math expressions. To combine the outputs of the GCN module and the biGraphSAGE module, a multi-head attention

Table 11 Comparison test of decoding cells working with different encoding cells on Math23k

Decoding module	Encoding Module							
	GRU		GRU+GCN		GRU+biGraphSAGE		BERT	
	E_{acc}	A_{acc}	E_{acc}	A_{acc}	E_{acc}	A_{acc}	E_{acc}	A_{acc}
DT cell [6]	**66.7**	**58.2**	**69.6**	**62.2**	**64.7**	**56.5**	**72.4**	**84.2**
BT cell [17]	51.4	45.1	59.1	52.7	58.0	50.0	62.1	68.9

network is used which takes the final hidden vectors of the GCN and biGraphSAGE as input and outputs a new combined hidden vector. The E_{acc} and A_{acc} results are presented in Table 12.

In Table 12, G_{qcell} and G_{qcom} denote the quantity cell graph and quantity comparison graph built in [6] separately as the input of the GCN network. G_{wcon} refers to the constituency graph defined in [17] and is processed by the biGraphSAGE network.

Obviously, the BERT-DT combination outperforms other combinations in almost all test items. Here, we focus on discussing the performance of lightweight model combinations.

Firstly, when selecting the number of layers for the sequence encoder, there is no significant difference in performance between 4-layer and 2-layer networks. The 2-layer GRU cell obtained the best result among all sequence-based cells. The 2-layer GRU cell performed better than the 4-layer cells and similar results were obtained by LSTM cells. Therefore, we believe that it may be not an efficient way to try to improve the problem encoding by increasing the depth of sequence-based neural networks.

Secondly, when incorporating graph information, the combination of G_{qcom} and G_{qcell} obtained the best performance. For GCN-based modules, the GCN cell with G_{qcom} information obtained outstanding results on all levels of length complexity tasks and the H_1 level depth complexity task. The GCN cell performs best on H_2 level task when combining the graph information of G_{qcell} and G_{qcom}. A possible reason is that the G_{qcell} is homologous to the basic text sequence while the G_{qcom} contains additional comparison information that plays an important role in guiding the math expression generation. Second, the biGraphSAGE cell with G_{wcon} got lower performance than GCN cells, partly due to the sparsity of the constituency matrix used.

Furthermore, considering the fusion of multiple features, it can be observed from Table 12 that the mixed module by combining GCN (G_{qcell}) and biGraphSAGE (G_{wcon}) achieved better performance than biGraphSAGE (G_{wcon}) but worse than GCN (G_{qcell}) individually applied. The performance is slightly improved after the G_{qcom} is added. However, the overall performance of mixed modules is worse than only GCN modules. This leads us to conclude that choosing an appropriate encoder is a key decision in hybrid information encoding.

Discussion

This paper provides a comprehensive survey and performance analysis of DL-based solvers for Math Word Problems (MWPs). These solvers are categorized into four distinct groups based on their network architecture and neural cell types: Seq2Seq-based models, Seq2Tree-based models, Graph2Tree-based models, and PLM-based models.

During the training phase, it has been observed that most models exhibit overfitting issues on datasets. Figure 5 illustrates the effectiveness of the Math23K training set, which consists of 18k instances, in meeting the training requirements of most deep learning-based models. Conversely, when trained on the MAWPS-s dataset, which contains approximately 1.5 k instances, almost all models show noticeable signs of overfitting. This particular finding serves as a valuable reference point for future endeavors involving dataset construction.

In terms of overall performance, pre-trained language models outperformed other models on both single-equation tasks and multi-equation tasks. As depicted in Table 6, GPT-4 achieves the best results on the MAWPS-s, SVAMP, and Draw-1k datasets, and the BertTree performs well on the two Chinese datasets, Math23k and HMWP. This demonstrates the significant advantage of PLM-based models, especially large-scale language modes such as GPT-4, in solving math word problems. However, there are variations in the performance of different pre-trained language models on Chinese and English datasets. Consistent findings were also reported in previous research studies [26, 27]. For instance, in [26], it is highlighted that the adoption of the Bert2Tree model yields a 2.1% improvement in answer accuracy compared to the Graph2Tree model on the MAWPS-s dataset, while achieving a 7% improvement on the Math23k dataset. This outcome can be attributed to two factors: (1) The Chinese pre-trained model employed in this study, namely Chinese BERT with whole word masking [55], differs from the BERT-base model used for English. Thus, it is reasonable to infer that task-specific training or fine-tuning of pre-trained language models is essential to fully leverage their advantages. (2) Pre-trained language models exhibit greater proficiency in handling MWPs with intricate semantics. As evidenced by Table 3, the average length of question texts in the Math23k and HMWP datasets is 1.5-2 times longer than that of other

Table 12 Confusion test of encoding cells combining with the DT cell on Math23K

Encoding Module	L_1		L_2		L_3		H_1		H_2	
	A_{acc}	E_{acc}	A_{acc}	E_{acc}	A_{acc}	E_{acc}	A_{acc}	E_{acc}	A_{acc}	E_{acc}
GRU (2 layers)	**73.3**	**69.3**	75.9	65.8	49.3	41.2	69.6	61.0	**60.3**	**54.0**
GRU (4 layers)	72.2	67.1	75.3	65.3	44.5	36.0	67.8	58.8	57.8	51.4
LSTM (2 layers)	73.1	68.0	75.6	64.6	**50.9**	**41.3**	70.3	60.1	59.4	53.4
LSTM (4 layers)	72.4	67.5	**76.3**	**66.6**	48.6	39.7	**70.3**	**61.2**	57.0	51.1
GCN (G_{qcell})	73.3	**69.1**	76.1	**66.8**	47.5	38.7	69.1	**61.2**	58.9	52.2
GCN (G_{qcom})	73.1	68.0	76.2	65.8	49.2	40.6	69.6	60.3	**60.5**	**54.3**
GCN (G_{qcell}+G_{qcom})	**75.1**	**69.1**	**76.8**	65.6	**51.5**	**40.9**	**71.8**	**61.2**	59.6	52.4
biGraphSAGE (G_{wcon})	67.8	62.4	71.5	62.9	45.7	36.7	65.7	57.2	54.3	48.0
GCN (G_{qcell})+biGraphSAGE (G_{wcon})	75.6	70.5	77.4	67.1	51.5	40.8	71.9	62.2	61.1	53.6
GCN (G_{qcell}+G_{qcom})+biGraphSAGE (G_{wcon})	**76.0**	**71.6**	**78.6**	**69.3**	**52.7**	**43.4**	**72.9**	**64.1**	**62.0**	**55.8**
BERT	84.7	89.0	77.1	89.4	55.2	72.0	74.0	86.2	67.4	77.5

datasets, suggesting the presence of more complex syntactic and semantic information. Utilizing pre-trained language models allows for improved extraction and utilization of pertinent information necessary for effective problem-solving.

Meanwhile, Tables 7 and 8 show that neural solvers are sensitive to the complexity of equations (e.g., equation length, equation tree height), as well as the length of the original problem text. However, we also found that (1) the MathEN model based on the Seq2Seq framework achieved the best results on some datasets (MAWPS-s), indicating that there is room for further optimization of the Graph2Tree framework. Further work is needed to discover the main factors influencing the performance of Graph2Tree on MAWPS-s and to improve it accordingly. (2) For all solvers on MAWPS-s, the increase in expression length did not result in a decline in solving accuracy, but rather showed varying degrees of improvement, which is completely opposite to what we observed on the other two datasets. Further research is needed to explain this phenomenon.

In terms of decoding performance, models integrated with tree decoders exhibit strong performance in generating math equations. Meanwhile, the DT Cell performed much better than the BT Cell on most datasets, making it widely used. However, we believe that the BT Cell still has its special advantages, as its decoding process is more in line with human thought in arithmetic reasoning, where a task is decomposed into multiple sub-tasks, each corresponding to a certain mathematical operation semantics. Therefore, the output results of this model can be better applied in intelligent education scenario, such as step-by-step intelligent tutoring. This raises new questions for researchers on how to design models with human-like arithmetic reasoning ability and make them run efficiently.

In terms of encoding performance, implicit information representation of problem texts plays a crucial role in enhancing the performance of models. Experimental results have shown that combining structure and non-structure information can effectively enhance solver performance. However, we found that not all structure information is equally effective, and some may be more useful in improving solving performance than others. Therefore, it is necessary to design more effective mechanisms or algorithms to determine which information should be added and how the added information can be fused with current information for maximum utility.

Moreover, the emergence of large language models such as GPT has propelled MWP-solving technology to a new stage. These models can gradually improve the accuracy of MWP solvers and their remarkable reasoning abilities enable them to generate step-by-step solutions based on prompts, which is truly impressive. However, these large language models also face challenges such as large parameter sizes and usage restrictions.

Limitations

The limitations of this study are primarily attributed to the emergence of novel models and their integration with knowledge bases, which present challenges in re-implementing these algorithms. Consequently, performance comparisons with specific papers such as [7, 27] have been considered in this study. Additionally, due to hardware constraints, we did not fine-tune the pre-trained language models, therefore, the performance of various fine-tuned models has not been reported.

Furthermore, for PLM-based models like GPT-4 [30], advanced prompts or interaction strategies were not employed in our experiments, which may result in lower accuracy. Moreover, it is worth noting that PLM-based models have the advantage of generating descriptive solution processes,

but their performance evaluation in this aspect has not been conducted in this study.

Conclusion

In this paper, we have aimed to provide a comprehensive and analytical comparative examination of the state-of-the-art neural solvers for math word problem solving. Our objective was to serve as a reference for researchers in the design of future models by offering insights into the structure of neural solvers, their performance, and the pros and cons of the involved neural cells.

We first identify the architectures of typical neural solvers, rigorously analyzing the framework of each category, particularly the four typical categories: Seq2Seq, Seq2Tree, Graph2Tree and PLM-based models. A four-dimensional indicator is proposed to categorize the considered datasets to precisely evaluate the performance of neural solvers in solving different characteristics of MWPs. Typical neural solvers are decomposed into highly reusable components. To evaluate the considered solvers, we have established a testbed and conducted comprehensive experiments on five popular datasets using eight representative MWP solvers, followed by a comparative analysis on the achieved results.

After conducting an in-depth analysis, we found that: (1) PLM-based models consistently demonstrate significant accuracy advantages across almost all datasets, yet there remains room for improvement to meet practical demands. (2) Models integrated with tree decoders exhibit strong performance in generating math equations. The length of expressions and the depth of expression trees are important factors affecting solver performance when solving problems with different expression features. The longer the expression and the deeper the expression tree, the lower the solver performance. (3) Implicit information representation of problem texts plays a crucial role in enhancing the performance of models. While the use of multi-modal feature representation has shown promising improvements in performance, it is crucial to ensure information complementary among modalities.

Based on our findings, we have the following suggestions for future work. Firstly, there is still room to improve the performance of solvers, including problem representation learning, multi-solution generation, etc.

Secondly, to better support the potential real-world applications in education, the output of solvers should be more comprehensive. Solvers are expected to generate decomposable and interpretable solutions, rather than just simple expressions or answers. The emergence of large language models has provided ideas for addressing this issue, but it remains a challenge to ensure the validity and interpretability of the outputs for teaching and tutoring applications.

Finally, to evaluate the neural solvers more comprehensively, it is necessary to develop more diverse metrics and evaluation methods in future research. These metrics and methods should capture the performance of solvers in problem understanding, automatic addition of implicit knowledge, solution reasoning, interpretability of results, and other relevant aspects.

Acknowledgements This work is supported by the National Natural Science Foundation of China (No. 62007014) and the Humanities and Social Sciences Youth Fund of the Ministry of Education (No. 20YJC880024).

Data Availability The data used in this study is available from the corresponding author upon reasonable request.

Declarations

Conflict of interest The authors declare that they have no Conflict of interest.

References

1. Zhang D, Wang L, Zhang L, Dai BT, Shen HT (2019) The gap of semantic parsing: a survey on automatic math word problem solvers. IEEE Trans Pattern Anal Mach Intell 42(9):2287–2305
2. Wang Y, Liu X, Shi S (2017) Deep neural solver for math word problems. In: Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pp. 845–854. Association for Computational Linguistics, Copenhagen, Denmark
3. Greff K, Srivastava RK, Koutník J, Steunebrink BR, Schmidhuber J (2017) LSTM: a search space odyssey. IEEE Trans Neural Netw Learn Syst 28(10):2222–2232
4. Wang L, Wang Y, Cai D, Zhang D, Liu X (2018) Translating a math word problem to a expression tree. In: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pp. 1064–1069. Association for Computational Linguistics, Brussels, Belgium
5. Xie Z, Sun S (2019) A goal-driven tree-structured neural model for math word problems. In: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, pp. 5299–5305. International Joint Conferences on Artificial Intelligence Organization, Macao, China
6. Zhang J, Wang L, Lee RKW, Bin Y, Wang Y, Shao J, Lim EP (2020) Graph-to-tree learning for solving math word problems. In: Proceedings of the 58th Annual Meeting of the Association for

Computational Linguistics, pp. 3928–3937. Association for Computational Linguistics, Seattle, USA
7. Wu Q, Zhang Q, Wei Z (2021) An edge-enhanced hierarchical graph-to-tree network for math word problem solving. In: Findings of the Association for Computational Linguistics: EMNLP 2021, pp. 1473–1482. Association for Computational Linguistics, Punta Cana, Dominican Republic
8. Yang Z, Qin J, Chen J, Lin L, Liang X (2022) LogicSolver: Towards interpretable math word problem solving with logical prompt-enhanced learning. In: Findings of the Association for Computational Linguistics: EMNLP 2022, pp. 1–13. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates
9. Jie Z, Li J, Lu W (2022) Learning to reason deductively: Math word problem solving as complex relation extraction. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, pp. 5944–5955. Association for Computational Linguistics, Dublin, Ireland
10. Lan Y, Wang L, Zhang Q, Lan Y, Dai BT, Wang Y, Zhang D, Lim EP (2022) Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. Proceedings of the AAAI Conference on Artificial Intelligence 36:13188–13190
11. Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017) Attention is all you need. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, pp. 6000–6010. Curran Associates Inc., Red Hook, NY, USA
12. Lin JCW, Shao Y, Djenouri Y, Yun U (2021) ASRNN: a recurrent neural network with an attention model for sequence labeling. Knowl-Based Syst 212:106548
13. Chiang TR, Chen YN (2019) Semantically-aligned equation generation for solving and reasoning math word problems. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologie, pp. 2656–2668. Association for Computational Linguistics, Minneapolis, Minnesota
14. Hong Y, Li Q, Ciao D, Haung S, Zhu SC (2021) Learning by fixing: Solving math word problems with weak supervision. In: Proceedings of the AAAI Conference on Artificial Intelligence, 35:4959–4967
15. Hong Y, Li Q, Gong R, Ciao D, Huang S, Zhu SC (2021) SMART: a situation model for algebra story problems via attributed grammar. In: Proceedings of the 2021 AAAI Conference on Artificial Intelligence, pp. 13009–13017. Vancouver, Canada
16. Roy S, Roth D (2017) Unit dependency graph and its application to arithmetic word problem solving. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, pp. 3082–3088. San Francisco, USA
17. Li S, Wu L, Feng S, Xu F, Xu F, Zhong S (2020) Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. In: Findings of the Association for Computational Linguistics: EMNLP 2020, pp. 2841–2852. Association for Computational Linguistics, Punta Cana, Dominican Republic
18. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
19. Bahdanau D, Cho KH, Bengio Y (2015) Neural machine translation by jointly learning to align and translate. In: 3rd International Conference on Learning Representations. San Diego, California
20. Cai D, Lam W (2020) Graph transformer for graph-to-sequence learning. Proceedings of the AAAI Conference on Artificial Intelligence 34:7464–7471
21. Hamilton WL, Ying R, Leskovec J (2017) Inductive representation learning on large graphs. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp. 1025–1035. Curran Associates Inc., Red Hook, NY, USA
22. Mukherjee A, Garain U (2008) A review of methods for automatic understanding of natural language mathematical problems. Artificial Intell Rev 29(2):93–122
23. Meadows J, Freitas A (2022) A survey in mathematical language processing. arXiv:2205.15231 [cs]
24. Lu P, Qiu L, Yu W, Welleck S, Chang KW (2023) A survey of deep learning for mathematical reasoning. arXiv:2212.10535 [cs]
25. Liu Q, Guan W, Li S, Kawahara D (2019) Tree-structured decoding for solving math word problems. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 2370–2379. Association for Computational Linguistics, Hong Kong, China
26. Liang Z, Zhang J, Wang L, Qin W, Lan Y, Shao J, Zhang X (2022) MWP-BERT: Numeracy-augmented pre-training for math word problem solving. In: Findings of the Association for Computational Linguistics: NAACL 2022, pp. 997–1009. Association for Computational Linguistics, Seattle, United States
27. Zhang W, Shen Y, Ma Y, Cheng X, Tan Z, Nong Q, Lu W (2022) Multi-view reasoning: Consistent contrastive learning for math word problem. In: Findings of the Association for Computational Linguistics: EMNLP 2022, pp. 1103–1116. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates
28. Brown T, Mann B, Ryder N, Subbiah M, Kaplan JD, Dhariwal P, Neelakantan A, Shyam P, Sastry G, Askell A, Agarwal S, Herbert-Voss A, Krueger G, Henighan T, Child R, Ramesh A, Ziegler D, Wu J, Winter C, Hesse C, Chen M, Sigler E, Litwin M, Gray S, Chess B, Clark J, Berner C, McCandlish S, Radford A, Sutskever I, Amodei D (2020) Language models are few-shot learners. In: Advances in Neural Information Processing Systems, 33:1877–1901. Curran Associates, Inc
29. Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I (2019) Language models are unsupervised multitask learners. Tech. rep., OpenAI. OpenAI blog
30. Zhou A, Wang K, Lu Z, Shi W, Luo S, Qin Z, Lu S, Jia A, Song L, Zhan M, Li H (2023) Solving challenging math word problems using GPT-4 code interpreter with code-based self-verification. https://doi.org/10.48550/arXiv.2308.07921. arXiv:2308.07921 [cs]
31. Fletcher CR (1985) Understanding and solving arithmetic word problems: a computer simulation. Behav Res Methods Instruments Comput 17(5):565–571
32. Kushman N, Artzi Y, Zettlemoyer L, Barzilay R (2014) Learning to automatically solve algebra word problems. In: Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, pp. 271–281. Association for Computational Linguistics, Baltimore, Maryland
33. Shen Y, Jin C (2020) Solving math word problems with multi-encoders and multi-decoders. In: Proceedings of the 28th International Conference on Computational Linguistics, pp. 2924–2934. International Committee on Computational Linguistics, Barcelona, Spain
34. Liang CC, Hsu KY, Huang CT, Li CM, Miao SY, Su KY (2016) A tag-based statistical english math word problem solver with understanding, reasoning and explanation. In: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, pp. 4254–4255. San Diego, USA
35. Devlin J, Chang MW, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), pp. 4171–4186. Association for Computational Linguistics, Minneapolis, Minnesota
36. Wang L, Zhang D, Zhang J, Xu X, Gao L, Dai BT, Shen HT (2019) Template-based math word problem solvers with recursive neural

networks. In: Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, pp. 7144–7151. AAAI Press, Hawaii, USA
37. Li J, Wang L, Zhang J, Wang Y, Dai BT, Zhang D (2019) 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, pp. 6162–6167. Association for Computational Linguistics
38. Qin J, Lin L, Liang X, Zhang R, Lin L (2020) Semantically-aligned universal tree-structured solver for math word problems. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 3780–3789. Association for Computational Linguistics, Punta Cana, Dominican Republic
39. Wu Q, Zhang Q, Wei Z, Huang X (2021) Math word problem solving with explicit numerical values. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pp. 5859–5869. Association for Computational Linguistics, Bangkok, Thailand
40. Yu W, Wen Y, Zheng F, Xiao N (2021) Improving math word problems with pre-trained knowledge and hierarchical reasoning. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pp. 3384–3394. Association for Computational Linguistics, Online and Punta Cana, Dominican Republic
41. Li Z, Zhang W, Yan C, Zhou Q, Li C, Liu H, Cao Y (2022) Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. In: Findings of the Association for Computational Linguistics: ACL 2022, pp. 2486–2496. Association for Computational Linguistics, Dublin, Ireland
42. Shen J, Yin Y, Li L, Shang L, Jiang X, Zhang M, Liu Q (2021) Generate & rank: A multi-task framework for math word problems. In: Findings of the Association for Computational Linguistics: EMNLP 2021, pp. 2269–2279. Association for Computational Linguistics, Punta Cana, Dominican Republic
43. Shen Y, Liu Q, Mao Z, Cheng F, Kurohashi S (2022) Textual enhanced contrastive learning for solving math word problems. In: Findings of the Association for Computational Linguistics: EMNLP 2022, pp. 4297–4307. Association for Computational Linguistics, Abu Dhabi, United Arab Emirates
44. Chowdhery A, Narang S, Devlin J, Bosma M, Mishra G, Roberts A, Barham P, Chung HW, Sutton C, Gehrmann S, Schuh P, Shi K, Tsvyashchenko S, Maynez J, Rao A, Barnes P, Tay Y, Shazeer N, Prabhakaran V, Reif E, Du N, Hutchinson B, Pope R, Bradbury J, Austin J, Isard M, Gur-Ari G, Yin P, Duke T, Levskaya A, Ghemawat S, Dev S, Michalewski H, Garcia X, Misra V, Robinson K, Fedus L, Zhou D, Ippolito D, Luan D, Lim H, Zoph B, Spiridonov A, Sepassi R, Dohan D, Agrawal S, Omernick M, Dai AM, Pillai TS, Pellat M, Lewkowycz A, Moreira E, Child R, Polozov O, Lee K, Zhou Z, Wang X, Saeta B, Diaz M, Firat O, Catasta M, Wei J, Meier-Hellstern K, Eck D, Dean J, Petrov S, Fiedel N (2022) PaLM: Scaling language modeling with pathways . arXiv:2204.02311 [cs]
45. Lewkowycz A, Andreassen A, Dohan D, Dyer E, Michalewski H, Ramasesh V, Slone A, Anil C, Schlag I, Gutman-Solo T, Wu Y, Neyshabur B, Gur-Ari G, Misra V (2022) Solving Quantitative Reasoning Problems with Language Models. In: Advances in Neural Information Processing Systems, vol. 35, pp. 3843–3857. Curran Associates, Inc
46. Touvron H, Lavril T, Izacard G, Martinet X, Lachaux MA, Lacroix T, Rozière B, Goyal N, Hambro E, Azhar F, Rodriguez A, Joulin A, Grave E, Lample G (2023). LLaMA: Open and efficient foundation language models https://doi.org/10.48550/arXiv.2302.13971. arXiv:2302.13971 [cs]
47. Chung J, Gulcehre C, Cho K, Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. In: NIPS 2014 Workshop on Deep Learning, December 2014. Montreal, Canada
48. Ghazvini A, Abdullah SNHS, Kamru Hasan M, Bin Kasim DZA (2020) Crime spatiotemporal prediction with fused objective function in time delay neural network. IEEE Access 8:115167–115183
49. Djenouri Y, Srivastava G, Lin JCW (2021) Fast and accurate convolution neural network for detecting manufacturing data. IEEE Trans Ind Inform 17(4):2947–2955
50. Wu Q, Zhang Q, Fu J, Huang X (2020) A knowledge-aware sequence-to-tree network for math word problem solving. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 7137–7146. Association for Computational Linguistics, Punta Cana, Dominican Republic
51. Gupta A, Kumar S, Kumar P S (2023) Solving age-word problems using domain ontology and bert. In: Proceedings of the 6th Joint International Conference on Data Science & Management of Data, pp. 95–103. ACM, New York, NY, USA
52. Petroni F, Rocktäschel T, Riedel S, Lewis P, Bakhtin A, Wu Y, Miller A (2019) Language models as knowledge bases? In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pp. 2463–2473. Association for Computational Linguistics, Hong Kong, China
53. Jiang Z, Xu FF, Araki J, Neubig G (2020) How can we know what language models know? Trans Assoc Comput Linguistics 8:423–438 (**Place: Cambridge, MA Publisher: MIT Press**)
54. Liu Z, Lin W, Shi Y, Zhao J (2021) A robustly optimized bert pre-training approach with post-training. In: Proceedings of the 20th Chinese National Conference on Computational Linguistics, pp. 1218–1227. Chinese Information Processing Society of China, Huhhot, China
55. Cui Y, Che W, Liu T, Qin B, Yang Z, Wang S, Hu G (2019) Pre-training with whole word masking for chinese bert. IEEE/ACM Trans Audio Speech Language Process 29:3504–3514
56. Chen J, Pan X, Yu D, Song K, Wang X, Yu D, Chen J (2023) Skills-in-context prompting: Unlocking compositionality in large language models. https://doi.org/10.48550/arXiv.2308.00304. arXiv:2308.00304 [cs]
57. Wei J, Wang X, Schuurmans D, Bosma M, ichter b, Xia F, Chi E, Le QV, Zhou D (2022) Chain-of-thought prompting elicits reasoning in large language models. In: Advances in Neural Information Processing Systems, vol. 35, pp. 24824–24837. Curran Associates, Inc
58. Huang X, Ruan W, Huang W, Jin G, Dong Y, Wu C, Bensalem S, Mu R, Qi Y, Zhao X, Cai K, Zhang Y, Wu S, Xu P, Wu D, Freitas A, Mustafa MA (2023) A survey of safety and trustworthiness of large language models through the lens of verification and validation. http://arxiv.org/abs/2305.11391. arXiv:2305.11391 [cs]
59. Dong L, Lapata M (2016) Language to logical form with neural attention. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pp. 33–43. Association for Computational Linguistics, Berlin, Germany
60. Zhang J, Lee RKW, Lim EP, Qin W, Wang L, Shao J, Sun Q (2020) Teacher-student networks with multiple decoders for solving math word problem. In: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, pp. 4011–4017. International Joint Conferences on Artificial Intelligence Organization, Yokohama, Japan
61. Bobrow DG (1964) Natural language input for a computer problem solving system. Tech. rep., Massachusetts Institute of Technology, USA

62. Bakman Y (2007) Robust understanding of word problems with extraneous information. arXiv General Mathematics. https://api.semanticscholar.org/CorpusID:117981901
63. Koncel-Kedziorski R, Hajishirzi H, Sabharwal A, Etzioni O, Ang SD (2015) Parsing algebraic word problems into equations. Trans Assoc Comput Linguistics 3:585–597 (**Place: Cambridge, MA**)
64. Roy S, Upadhyay S, Roth D (2016) Equation parsing: Mapping sentences to grounded equations. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 1088–1097. Association for Computational Linguistics, Austin, Texas
65. Wang L, Zhang D, Gao L, Song J, Guo L, Shen HT (2018) MathDQN: Solving arithmetic word problems via deep reinforcement learning. In: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, pp. 5545–5552. AAAI Press, New Orleans, USA
66. Hosseini MJ, Hajishirzi H, Etzioni O, Kushman N (2014) Learning to solve arithmetic word problems with verb categorization. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 523–533. Association for Computational Linguistics, Doha, Qatar
67. Shi S, Wang Y, Lin CY, Liu X, Rui Y (2015) Automatically solving number word problems by semantic parsing and reasoning. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1132–1142. Lisbon, Portugal
68. Liang CC, Hsu KY, Huang CT, Li CM, Miao SY, Su KY (2016) A tag-based English math word problem solver with understanding, reasoning and explanation. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations, pp. 67–71. Association for Computational Linguistics, San Diego, California
69. Upadhyay S, Chang MW, Chang KW, Yih Wt (2016) Learning from explicit and implicit supervision jointly for algebra word problems. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pp. 297–306. Association for Computational Linguistics, Austin, Texas
70. Chen S, Zhou M, He B, Wang P, Wang Z (2022) A comparative analysis of math word problem solving on characterized datasets. In: In Proceedings of the 2022 International Conference on Intelligent Education and Intelligent Research. IEEE, Wuhan, China
71. He B, Chen S, Miao Z, Liang G, Pan K, Huang L (2022) Comparative analysis of problem representation learning in math word problem solving. In: In Proceedings of the 2022 International Conference on Intelligent Education and Intelligent Research. IEEE, Wuhan, China
72. Roy S, Roth D (2015) Solving general arithmetic word problems. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 1743–1752. Association for Computational Linguistics, Lisbon, Portugal
73. Yenduri G, M R, G CS, Y S, Srivastava G, Maddikunta PKR, G DR, Jhaveri RH, B P, Wang W, Vasilakos AV, Gadekallu TR (2023) Generative pre-trained transformer: A comprehensive review on enabling technologies, potential applications, emerging challenges, and future directions. arXiv:2305.10435
74. Zhang H, Lu G, Zhan M, Zhang B (2021) Semi-supervised classification of graph convolutional networks with laplacian rank constraints. Neural Process Lett 54(4):2645–2656
75. Zhou L, Dai S, Chen L (2015) Learn to solve algebra word problems using quadratic programming. In: Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pp. 817–822. Association for Computational Linguistics, Lisbon, Portugal
76. Koncel-Kedziorski R, Roy S, Amini A, Kushman N, Hajishirzi H (2016) MAWPS: A math word problem repository. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 1152–1157. Association for Computational Linguistics, San Diego, California
77. Huang D, Shi S, Lin CY, Yin J, Ma WY (2016) How well do computers solve math word problems? Large-scale dataset construction and evaluation. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, pp. 887–896. Association for Computational Linguistics, Berlin, Germany
78. Upadhyay S, Chang MW (2017) Annotating derivations: A new evaluation strategy and dataset for algebra word problems. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, pp. 494–504. Association for Computational Linguistics, Valencia, Spain
79. Patel A, Bhattamishra S, Goyal N (2021) Are NLP models really able to solve simple math word problems? In: Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pp. 2080–2094. Association for Computational Linguistics, Bangkok, Thailand