LogicSolver: Towards Interpretable Math Word Problem Solving with Logical Prompt-enhanced Learning

Zhicheng Yang\textsuperscript{1,2\!*}, Jinghui Qin\textsuperscript{3\!*}, Jiaqi Chen\textsuperscript{2,4}, Liang Lin\textsuperscript{2} and Xiaodan Liang\textsuperscript{1,2\!*}\dagger
\textsuperscript{1}Shenzhen Campus of Sun Yat-sen University, \textsuperscript{2}Sun Yat-sen University, \textsuperscript{3}Guangdong University of Technology, \textsuperscript{4}Dark Matter AI Inc.
\{yangzhch6,qinjingh\}@mail2.sysu.edu.cn
linliang@ieee.org, \{jadgechen,xdliang328\}@gmail.com

Abstract

Recently, deep learning models have made great progress in MWP solving on answer accuracy. However, they are uninterpretable since they mainly rely on shallow heuristics to achieve high performance without understanding and reasoning the grounded math logic. To address this issue and make a step towards interpretable MWP solving, we first construct a high-quality MWP dataset named InterMWP which consists of 11,495 MWPs and annotates interpretable logical formulas based on algebraic knowledge as the grounded linguistic logic of each solution equation. Different from existing MWP datasets, our InterMWP benchmark asks for a solver to not only output the solution expressions but also predict the corresponding logical formulas. We further propose a novel approach with logical prompt and interpretation generation, called LogicSolver. For each MWP, our LogicSolver first retrieves some highly-correlated algebraic knowledge and then passes them to the backbone model as prompts to improve the semantic representations of MWPs. With these improved semantic representations, our LogicSolver generates corresponding solution expressions and interpretable knowledge formulas in accord with the generated solution expressions, simultaneously. Experimental results show that our LogicSolver has stronger logical formula-based interpretability than baselines while achieving higher answer accuracy with the help of logical prompts, simultaneously. The source code and dataset is available at https://github.com/yangzhch6/InterMWP.

1 Introduction

Automatically math word problem (MWP) solving is a challenging task in natural language processing since it aims to transform a concise narrative rich in mathematical relationships into a solution equation.

\*Zhicheng Yang and Jinghui Qin are contributed equally to this work.
\daggerXiaodan Liang is the corresponding author.
called InterMWP consisting of 11,495 annotated samples and 210 different logic formulas based on algebraic knowledge. In our InterMWP dataset, each solution equation is annotated with interpretable logic formulas in a tree structure as the grounded logic of each solution equation. As shown in Figure 1(b), each inner node is annotated with an interpretable algebraic knowledge formula which represents the grounded logic for the subtree with the current node as the root ancestor. With these logic annotations, our InterMWP asks for a solver to not only output the solution equation but also output the logic formulas simultaneously when the current predicted node is an inner-node (operator) during expression reasoning. Therefore, an MWP solver developed on InterMWP can output a solution equation while generating a reasonable formula-based interpretation. We use answer accuracy from prior works (Wang et al., 2017; Xie and Sun, 2019; Zhang et al., 2020b), together with formula accuracy and logic accuracy we proposed in Section 5.1 to evaluate the model’s solving ability and interpretability.

To leverage mathematical logic knowledge and empower an MWP solver with interpretability, we further present a novel framework named LogicSolver which extracts mathematical logic knowledge as logical prompts to improve the semantic representations of MWPs and enhance the ability of explanation generation. In our LogicSolver, we design a logic formula retriever to first extract logic prompts consisting of logic formulas highly-correlated with current MWP. Then, the logic prompts will be concatenated with the problem text as the input and drive the MWP model to produce the solution equation. Finally, to obtain the logic formulas-based explanation, we propose a logic generator to predict logic formulas for each inner-node of the solution expression tree. Experimental results show that our LogicSolver has stronger logical formula-based interpretability than baselines while achieving higher answer accuracy with the help of logical prompts, simultaneously.

In this work, our contributions can be summarized in the following three folds:

• We construct a high-quality interpretable MWP dataset InterMWP for interpretable MWP solving. In our InterMWP, there are 11,495 MWPs and each solution equation is annotated with interpretable logical formulas.

• We propose a powerful framework named LogicSolver to incorporate mathematical logic knowledge through logical prompt-enhanced learning for enhancing problem understanding while empowering models with interpretability. To the best of our knowledge, this is the first work to study prompt-enhanced learning in MWPs.

• We achieved 2.1%, 2.9%, and 9.5% improvement on answer accuracy, formula accuracy, and logic accuracy respectively. Experimental results on InterMWP show that our LogicSolver has strong logical formula-based interpretability which achieves higher answer accuracy simultaneously.

2 Related Work

2.1 Math Word Problem Solving

In recent years, deep learning-based models (Wang et al., 2017; Huang et al., 2018; Wang et al., 2018b, 2019; Xie and Sun, 2019; Chiang and Chen, 2019; Zhang et al., 2020a,b; Qin et al., 2020, 2021) have shown impressive performance in solving MWPs by automatically learning to directly translate a problem text into an expression without any hand-crafted feature design. Wang et al. (2017) make the first attempt to apply a vanilla sequence to the sequence (seq2seq) model. Huang et al. (2018) improved their work by introducing a copy and attention mechanism. Xie and Sun (2019) propose a tree-structure decoder to decode expressions in prefix order. Furthermore, Zhang et al. (2020b) improved problem representation by fusing a graph encoder. Hong et al. (2021) propose a situation model for algebra story problems. Qin et al. (2021) propose auxiliary tasks to improve problem representation and the ability to predict common-sense constants. Wu et al. (2021) achieved better performance by incorporating numerical values into a sequence-to-tree network and applying a numerical properties prediction mechanism. Yang et al. (2022) propose an unbiased dataset and a dynamic target selection (DTS) strategy to eliminate the solving bias. However, all these models lack grounded math logic reasoning and interpretability so they can not give a reasonable explanation corresponding to the generated expression. To overcome these issues, we build a novel high-quality interpretable MWP dataset and propose a linguistic logic-enhanced framework for generating expression trees and their corresponding formula-based interpretations.
2.2 Prompt-enhanced Learning

Prompting PLMs for few-shot learning has become a popular learning paradigm. PET (Schick and Schütze, 2020a,b) transfer text classification problems to cloze-style problems while using manually defined prompts to provide additional task guidance. Gao et al. (2021) propose a pipeline for automating prompt generation to facilitate prompt discovery. Jiang et al. (2020) extract prompt from the training corpus. Besides that, Chen et al. (2021) injects latent knowledge contained in relation labels into the prompt for relation extraction. Hu et al. (2021) also incorporate external knowledge into a verbalizer to improve and stabilize prompt-tuning for text classification. Although Chen et al. (2021); Hu et al. (2021) incorporate knowledge into PLMs, they mainly focus on the shallow representation. Unlike these works, we train a model to select prompt automatically from a manually designed prompt set which summarizes the mathematical knowledge needed to solve the math word problems.

2.3 Interpretability of MWP Solvers

Although the prior statistical models with hand-crafted features can be thought of as interpretable due to the clear alignments between inputs and outputs, recently proposed deep learning approaches present new challenges to model interpretability of MWP solvers (Huang et al., 2016). Liang et al. (2018) used pattern matching to increase the robustness and interpretability of MWP solvers. Amini et al. (2019) propose operation-based formalisms to improve the interpretability. Cobbe et al. (2021) propose an MWP dataset called GSM8K which annotates the explanation for each step. But they do not summarize the mathematical knowledge in explanation. Besides, Roy and Roth (2018) also propose declarative rules which govern the translation of natural language to math expressions and presents a framework that learns to select the relevant declarative knowledge for each operation of the expression. Different from these works, we propose to predict linguistic math logic involving real-world knowledge along with expression construction so that an MWP solver can explain the grounded reason about the expression generation with linguistic logic formulas.

Figure 2: Some example comparisons between former MWP benchmarks and InterMWP benchmarks.

3 InterMWP

3.1 Data Collection

Most existing datasets for math word problem solving mainly consist of 4 attributes: problem id, problem text, solution equation, and final answer, such as Math23K (Wang et al., 2017), MaWPS (Koncel-Kedziorski et al., 2016), HMWP (Qin et al., 2020), and CM17K (Qin et al., 2021). Since there is no annotated explanation for solving equations, an MWP solver is incapable to produce an explanation grounded in the generated equation. To make a step towards interpretable MWP solving, we construct a high-quality interpretable MWP dataset called InterMWP to empower an MWP solver with the ability of interpretation to reason out solution equations and produce corresponding explanations for the generated equations. Excepting from the attributes mentioned above, we add the extra interpretable formula-based tree-structure annotation into the dataset so that we can force an MWP solver to not only output solution equation but also give out grounded logic formulas on the operators, thus endowing the MWP solver with certain interpretability.

To collect InterMWP, we sampled 8260 examples randomly from Math23K and crawled another 3,235 examples from a web bank\(^1\) to increase data diversity. In total, there are 11,495 examples collected in InterMWP. For each example, we first transferred the sequence solution equations to tree equations following the method in Xie and Sun (2019). The annotation procedure can be referred to Appendix B in our supplemental materials. During annotating, we mainly focus on the logic formulas which involve real-world knowledge such as \(\text{cost} = \text{quantity} \times \text{price}, \text{speed} = \text{distance} / \text{time}, \) etc. For the basic and simple mathematical logic knowledge described in Roy and Roth (2018), we use common-sense step as the logical formula. The logic formulas involving real-world knowledge are grouped into four

\(^1\)https://damolx.com/
3.2 The other superiorities of our InterMWP

Except for the explainable logic formulas, the other superiorities of our InterMWP can be mainly summarized in the following two points:

a) Formula variables disambiguation: As the prior MWP datasets such as Math23K (Wang et al., 2017), Alg514 (Kushman et al., 2014) and MAWPS (Koncel-Kedziorski et al., 2016) only provide a numeric expression for each problem, the reference to the variables in the formula may be ambiguous. A data example of such formula ambiguous is problem A shown in Figure 2, the original method in Wang et al. (2017) cannot map the two numbers ‘2’ in the equation to different positions in the problem. We overcome this shortcoming by mapping between numbers in the problem and numbers in the solution equation manually during the procedure of annotating logic formulas.

b) The complete solution set for each MWP in the test set: The former metrics to evaluate the accuracy of an MWP solver mainly rely on the answer accuracy, but an MWP solver may output a right answer by generating a wrong formula. As shown in Figure 2, for problem A, an MWP solver can obtain the correct answer by generating an error constant number ‘2’. Besides, for problem B, the generated equation does not match the original equation although they are essentially the same. To overcome this shortcoming, we generate equivalent solution equations as many as possible for each MWP in the test set so that we can measure the ability of an MWP solver better.

4 LogicSolver

4.1 Overview

As shown in Figure 3, there are three main collaborative components in our proposed LogicSolver to solve an MWP and give out the corresponding logic formula-based explanation simultaneously. For each MWP, we first deploy a logic formula retriever to select the top-\(K\) highly-correlated logic formulas as logic prompts for prompt-enhanced solving. Then, the logic prompts will be concatenated with the problem text as the input and drive the MWP model to produce a solution equation. Finally, to obtain the logic formulas-based explanation, we deploy a logic generator to predict logic formulas for each inner-node (operator) of the solution expression tree.
4.2 Logic Formulas Retrieval

It should be helpful for MWP solving if we can inject the semantics of the logic formulas grounded in MWPs into an MWP solver since the logic formulas grounded in MWPs denote the grounded math relationships. Therefore, to inject logic formulas into an MWP solver to improve the ability of semantic representation and reasoning, we train a retriever to match the logic knowledge in our InterMWP. Our retriever takes a problem text and the 210 logic formulas we summarized as the input and outputs the matching score for each logic formula.

BERT (Devlin et al., 2019) is an efficient pre-trained language model for text encoding, so we employ a Chinese BERT pre-trained with whole word masking (Cui et al., 2020) as our encoder, denoted as $\text{BERT}_R$, for learning the semantic representations of text. To encode the problem text $P$ and logic formula set $F = [F_1, F_2, \cdots, F_T]$ where $T$ is the size of the logic set, we pass them to the retriever $\text{BERT}_R$ and average the feature outputs from the last hidden layer to obtain the corresponding semantic embeddings as follows:

$$
\begin{align*}
\hat{p} &= \text{mean}(\text{BERT}_R(P)) \\
\hat{f}_i &= \text{mean}(\text{BERT}_R(F_i))
\end{align*}
$$

Then, a scoring module $\text{Score}_R$ is deployed to rate each logic formula $F_i$ as follows:

$$
\begin{align*}
\text{Score}_R(p, f_i) &= v_s^T \text{tanh}(W_s[p, f_i]) \\
s_i &= \text{Score}_R(p, f_i)
\end{align*}
$$

where $v_s$ and $W_s$ are trainable parameters.

Consider the dataset $\mathcal{D}$, for each data sample $(P, L_f) \in \mathcal{D}$, where $L_f = [l_{f_0}, l_{f_1}, \cdots, l_{f_T}]$ is a 0-1 vector, and $l_{f_i}$ indicate whether the logic formula $F_i$ is used in solving the problem $P$, and $T$ is the size of the logic set, we minimize the following loss function for training the retriever:

$$
\mathcal{L}_r = \log(1 + \sum_{l_{f_i} = 1} e^{s_i}) + \log(1 + \sum_{l_{f_i} = 0} e^{-s_i})
$$

For the positive logic formulas, we expect a higher score, and for the negative logic formulas, the opposite is true.

4.3 Logical Prompt-enhanced MWP Solving

To solve an MWP, we follow the encoder-decoder structure. For the encoder, we choose the Chinese BERT pre-trained with whole word masking (Cui et al., 2020), denoted as $\text{BERT}_E$, for learning the MWP representation. For decoder, we employ the goal-driven tree-structure decoder, denoted as $\text{GTS}$, following the previous work (Xie and Sun, 2019) to generate solution expression in prefix order, as shown in Figure 3.

To conduct logical prompt-enhanced learning with highly-correlated logic formulas, for a problem $P$, we first select the top-$K$ logic formulas based on their matching scores $\{s_0, s_1, \cdots, s_T\}$ as prompts. Then we concatenate the selected $K$ logic formulas with $P$ and pass it into the encoder:

$$
c, q_{root} = \text{BERT}_E([P, F_{\text{top}}])
$$

where $c$ is the encoder’s last hidden layer of all tokens, and $q_{root}$ which will be used as the root node’s goal vector is the [CLS] token embedding of the last hidden layer.

For each node in the expression tree, the goal-driven tree-structure decoder $\text{GTS}$ takes the goal vector $q$ and the token-level embedding $c$ as the input. The decoder $\text{GTS}$ first uses $q$ to predict token $\hat{y}$ of the current node. If the predicted token is a mathematical operator, the goal will be decomposed into two sub-goals which will be passed to the corresponding sub-trees. Otherwise, the goal will be simply realized by the predicted numeric value or constant quantity. The final output of $\text{GTS}$ is the node tokens $\hat{Y} = \{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$ of the solution expression in prefix order and the corresponding goal vector of each node $Q = \{q_1, q_2, \cdots, q_n\}$, $n$ is the solution expression length.

$$
\hat{Y}, Q = \text{GTS}(c, q_{root})
$$

Given the dataset $\mathcal{D} = \{(Y_0, P_0), (Y_1, P_1), \cdots, (Y_N, P_N)\}$, we minimize the following loss function during training the encoder-decoder model:

$$
\mathcal{L}_d(\hat{Y}_i|P_i) = -\sum_{t=1}^{E_i} \log(\text{prob}(y_t|q_t, P_i))
$$

where $\hat{Y}_i$ is the predicted expression for $P_i$, $E_i$ is the number of tokens of the solution expression of problem $P_i$, and $\text{prob}(y_t|q_t, P_i)$ is computed by distribution computation function in $\text{GTS}$.

4.4 Explanation Generation

To empower the MWP solver’s interpretability, we propose a logic generator to take the operator’s hidden embedding $q$, problem text $P$, and the logic set $F$ as the input and predict which linguistic logic formula can explain the decision on the current
operator. We deploy a BERT model denoted as BERT_L to encode each logic formula and problem text.

\[
p^L = \text{BERT}_L(P) \\
f^L_i = \text{mean}(\text{BERT}_L(F_i)), F_i \in F
\]  
(7)

where \(p^L\) and \(f^L_i\) denote the problem’s token embedding for \(P\) and the logic formula embedding for \(F_i\) in the logic generator respectively.

To choose a reasonable explanation for the decision on expression generation, we leverage the attention-based scoring mechanism to select an appropriate logic formula as the explanation. Given an operator’s embedding \(q\), a context vector \(c^L\) is obtained by attending \(q\) with problem token embedding \(p^L\) with the help of the attention mechanism (Bahdanau et al., 2015):

\[
c^L = \text{Attention}(q, p^L)
\]  
(8)

Then, a scoring module denoted as \(s_L\) is deployed to output the unnormalized log probability:

\[
s_L(F_i|q, c^L) = v^T_L \tanh(W_L[q, c^L, f_i])
\]  
(9)

where \(v_L\) and \(W_L\) are trainable parameters.

Finally, the normalized probability \(\text{prob}(F_i|q, c^L, F)\) over logic formula set \(F\) is computed with the softmax function:

\[
\text{prob}(F_i|q, c^L, F) = \frac{\exp(s_L(F_i|q, c^L, F))}{\sum_j \exp(s_L(F_j|q, c^L, F))}
\]  
(10)

Our logic generator selects the logic formula \(y^F\) with the highest probability as the explanation for the decision on operator generation:

\[
y^F = \arg \max \text{prob}(F_i|q, c^L, F)
\]  
(11)

The training objective of the logic generator is as follows:

\[
\mathcal{L}_L(\hat{Y}^F_i|P_i) = -\sum_{t=1}^{E^F_i} \log(p_{r^t}(y^F_i|q, c^L_i, P_i))
\]  
(12)

where \(\hat{Y}^F_i\) is the predicted expression for \(P_i\), \(E^F_i\) is the number of operators of the solution expression of the problem \(P_i\).

5 Experiments

5.1 Experimental Setup

Datasets. We conduct experiments on our InterMWP and Math23K (Wang et al., 2017) in the train-valid-test setting. For Math23K, we train the logic retriever on our InterMWP and then use the pre-trained logic retriever to extract logic prompts for the MWPs in the Math23K.

Baselines. We compare our LogicSolver with the following state-of-the-art models: MathEN (Wang et al., 2018a): a seq2seq model with equation normalization for reducing target space. GROUPATT (Li et al., 2019): a math word problem solver borrowing the idea of multi-head attention from Transformer (Vaswani et al., 2017). GTS (Xie and Sun, 2019): a tree-structured neural network in a goal-driven manner to generate expression trees. Graph2Tree (Zhang et al., 2020b): an enhanced GTS with quantity graph. GTS(BERT): a strong baseline we constructed by replacing RNN encoder with BERTEncoder(Devlin et al., 2019) in GTS.

Evaluation Metric. We use three metrics to measure the problem solving ability and interpretability of the models:

- Following prior works (Wang et al., 2017; Xie and Sun, 2019; Zhang et al., 2020b), we use answer accuracy as one of the evaluation metrics: if the calculated value of the predicted expression tree equals the true answer, it is thought as correct.

- However, answer accuracy will overestimate the ability of reasonable expression generation of an MWP solver, so we also introduce formula accuracy to evaluate whether the generated expression is one of a set of reasonable expressions that we annotate an MWP by listing all possible and reasonable solution equations manually in the test set.

- Moreover, to measure the effectiveness of the output linguistic logic, we introduce logic accuracy: Given the dataset \(\mathcal{D} = \{(Y_0, Y^F_0, P_0), (Y_1, Y^F_1, P_1), \ldots, (Y_N, Y^F_N, P_N)\}\) where \(Y_i\) denotes solution expression, \(Y^F_i\) denotes the target linguistic logic formulas, and \(P_i\) is the problem text. For an MWP, if the predicted solution expression \(\hat{Y}_i\) is correct and the whole predicted linguistic logic \(\hat{Y}^F_i\) is equivalent to the target linguistic logic, we consider this logic formula-based explanation is correct. The formula for computing logic accuracy is as following below:

\[
\text{logic acc} = \frac{1}{N} \sum_{i=1}^{N} (\hat{Y}_i = Y_i)(\hat{Y}^F_i = Y^F_i)
\]  
(13)
Implementation Details. We use Pytorch\(^2\) to implement our model on Linux with an NVIDIA RTX3090 GPU card. We add the [NUM] token to BERT's vocab and convert all numbers in problem text to the [NUM] token. For the training of the logic generator in LogicSolver, we only select the MWPs which can be fitted in the train set of our InterMWP as the training data. We use goal vectors in GTS-decoder models (Xie and Sun, 2019; Zhang et al., 2020b) as the embedding for solution expression tokens in the logic generator, and select RNN's hidden states in RNN-decoder models (Wang et al., 2017; Li et al., 2019) as solution expression tokens embedding. More details can be referred to Appendix C.

5.2 Main Result

| Model                     | answer acc | formula acc | logic acc |
|---------------------------|------------|-------------|-----------|
| Math-EN(Wang et al., 2018a) | 63.9       | 60.2        | 43.2      |
| Group-Attn(Li et al., 2019) | 64.2       | 60.8        | 44.5      |
| GTS(Xie and Sun, 2019)    | 70.5       | 66.1        | 57.2      |
| Graph2Tree(Zhang et al., 2020b) | 71.0       | 66.7        | 57.9      |
| NS-Solver(Qin et al., 2021) | 71.2       | 66.8        | 57.6      |
| GTS(BERT)                 | 80.3       | 76.8        | 66.5      |
| LogicSolver(Ours)         | 82.4       | 79.7        | 76.0      |

Table 1: The answer acc, formula acc, and logic acc on our InterMWP.

The Performance on InterMWP. The results on our InterMWP are shown in Table 1. With the logic prompt-enhanced, the answer accuracy can be improved from 80.3% [GTS(BERT)] to 82.4% [LogicSolver(Ours)]. Similarly, the formula accuracy and the logic accuracy also are improved from 76.8% [GTS(BERT)] to 79.7% [LogicSolver(Ours)] and 66.5%[GTS(BERT)] to 76.0% [LogicSolver(Ours)] respectively. This shows the effectiveness of our proposed prompt-enhanced learning for MWP solving. We also evaluate the performance on the samples contain top-10 logic formulas, the results are shown in Table 8 of Appendix D.

| Indicators | top 1 | top 2 | top 3 | top 4 |
|------------|------|------|------|------|
| Recall     | 0.642| 0.948| 0.985| 0.991|
| Precision  | 0.557| 0.411| 0.284| 0.215|
| F-10       | 0.641| 0.935| 0.962| 0.956|

Table 3: Performance of retriever on InterMWP.

5.3 The Performance of the Logic Retriever

We use Recall, Precision, and F-10 to quantify the performance of the logic retriever on our InterMWP under selecting top 1-4 logic formulas, as shown in Table 3. Corrected logic prompts are very important for improving the MWP solver. To retrieve as many correct logic prompts as possible and decrease the effect of error logic prompts, the recall rate is more important than the precision for logic retrieve. Therefore, we use F-10 score, rather than F-1 score, to measure the balanced performance of the logic retriever. From Table 3, we can observe that selecting top-3 logic formulas as prompts can achieve a better trade-off between Recall and Precision.

5.4 Logical Prompts Design

We study the effects of different prompt designs and the effects placement position of logic prompts in the input by conducting three experiments:

1. Random Selection: the logic prompts are chosen randomly.
2. Retrieve+Ahead: the logic retriever is deployed for logic prompts retrieving and the logic prompts are put in front of the MWP.
3. Retrieve+Behind: the logic retriever is deployed for logic prompts retrieving and the logic prompts are put behind the MWP.

The results are shown in Table 4. From the results, we can know that the best result is obtained under the Retrieve+Behind setting by retrieving top-3 logic prompts.

5.5 Performance of Interpretability

We use the proposed Logic Generator to achieve interpretability and use logic accuracy to evaluate logic formulas for Math23K, and then conduct logical prompt-enhanced learning for GTS(Bert). The results are shown in Table 2. The performance of answer accuracy increases from 82.8% to 83.4%. This improvement shows the strong generalization of our proposed logical prompt-enhanced learning on other MWP datasets even with the logic prompts based on the InterMWP.

\(^2\)http://pytorch.org
A ribbon is cut every 1.4 decimetres to make 1 bow. A total of 27 bows are made. There is 1.2 decimetres left. How many decimetres is this ribbon originally?

Trees were planted on one side of a 30 meter long road. A total of 4 trees were planted from beginning to end (planting at both ends). What is the distance between two adjacent trees in meters?

The Changsha-Guangzhou railway is 728km long, and a truck runs 71km per hour from Guangzhou to Changsha. A train of passenger cars drove from Changsha to Guangzhou at the same time, and the two cars met in 4 hours. What was the speed of this train?
word problem dataset called InterMWP which consists of 11,495 MWP data and annotates interpretable logical formulas based on algebraic knowledge as the grounded linguistic logic of each solution equation. Different from existing MWP datasets, our InterMWP benchmark asks for a solver to not only output the solution expressions but also predict the corresponding logical formulas. We further propose LogicSolver which is enhanced by logical prompts and is able to generate corresponding solution expressions and interpretable knowledge formulas in accord with the generated solution expressions, simultaneously. Experimental results show that our LogicSolver has stronger logical formula-based interpretability than baselines while achieving higher answer accuracy with the help of logical prompts, simultaneously.

7 Limitations

In this work, we make a step towards interpretable MWP solving by constructing a new MWP dataset called InterMWP and proposing a novel LogicSolver enhanced by logical prompts to infer out solution expressions and logical formula-based interpretability, simultaneously. However, there are still some limitations in our work. First, although our InterMWP is annotated with logical interpretability, the number of logical formulas are limited and needed to be extended for covering more cases when applying in real-world application. Second, even though our solver has better reasoning ability than current state-of-the-art methods on MWP solving and interpretation, it still needs more effort to design a more effective symbolic generation mechanism to enable a solver to handle more complex cases, such as more difficult problems with larger equations.

Acknowledgements

This work was supported in part by National Key R&D Program of China under Grant No. 2020AAA0109700, National Natural Science Foundation of China (NSFC) under Grant No.61976233 and Grant No.62206314, Guangdong Province Basic and Applied Basic Research (Regional Joint Fund-Key) Grant No.2019B1515120039, Guangdong Outstanding Youth Fund (Grant No. 2021B1515020061), GuangDong Basic and Applied Basic Research Foundation under Grant No.2022A1515011835, China Postdoctoral Science Foundation under Grant No.2021M703687, Shenzhen Fundamental Research Program (Project No. JCYJ20190807154211365), CAAI-Huawei MindSpore Open Fund, and The Open Project of Anhui Provincial Key Laboratory of Multimodal Cognitive Computation, Anhui University, No.MMC202107. We thank MindSpore for the partial support of this work, which is a new deep learning computing framework.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorksi, Yejin Choi, and Hannaneh Hajishirzi. 2019. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. ArXiv, abs/1905.13319.

Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In 3rd International Conference on Learning Representations, ICLR 2015 ; Conference date: 07-05-2015 Through 09-05-2015.

Xiang Chen, Ningyu Zhang, Xin Xie, Shumin Deng, Yunzhi Yao, Chuanqi Tan, Fei Huang, Luo Si, and Huajun Chen. 2021. Knowprompt: Knowledge-aware prompt-tuning with synergistic optimization for relation extraction. arXiv preprint arXiv:2104.07650.

Ting-Rui Chiang and Yun-Nung Chen. 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 Technologies, Volume 1 (Long and Short Papers), pages 2656–2668. Association for Computational Linguistics.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. ArXiv, abs/2110.14168.

Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting pre-trained models for Chinese natural language processing. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings, pages 657–668, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 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
Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojian Liu. 2018a. Translating a math word problem to a expression tree. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, pages 1064–1069. Association for Computational Linguistics.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Math-dqn: Solving arithmetic word problems via deep reinforcement learning. In Thirty-Second AAAI Conference on Artificial Intelligence, pages 5545–5552.

Lei Wang, Dongxiang Zhang, Zhang Jipeng, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In Thirty-Third AAAI Conference on Artificial Intelligence, pages 7144–7151.

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 845–854. Association for Computational Linguistics.

Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuanjing Huang. 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 (Volume 1: Long Papers), pages 5859–5869. Online. Association for Computational Linguistics.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, pages 5299–5305. International Joint Conferences on Artificial Intelligence Organization.

ZhiCheng Yang, Jinghui Qin, Jiaqi Chen, and Xiaodan Liang. 2022. Unbiased math word problems benchmark for mitigating solving bias. In Findings of the Association for Computational Linguistics: NAACL 2022, pages 1401–1408, Seattle, United States. Association for Computational Linguistics.

Jipeng Zhang, Roy Ka-Wei Lee, Ee-Peng Lim, Wei Qin, Lei Wang, Jie Shao, and Qianru Sun. 2020a. Teacher-student networks with multiple decoders for solving math word problem. In Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20, pages 4011–4017. International Joint Conferences on Artificial Intelligence Organization. Main track.

Jipeng Zhang, Lei Wang, Roy Ka-Wei Lee, Yi Bin, Yan Wang, Jie Shao, and Ee-Peng Lim. 2020b. Graph-to-tree learning for solving math word problems. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, pages 3928–3937.

### A Dataset Statistics

The InterMWP dataset consists of 11,495 problems and is divided into three parts randomly: 9495 training data, 1000 validation data, and 1000 test data. Table 5 shows the statistics and some samples of four logic categories in InterMWP dataset. The contents in parentheses indicate the number of occurrences of the logical formulas of the category in the InterWMP dataset.

| Geometric Logics (998) | Rectangular area = length × width | parallelogram area = bottom × height | square of the radius = radius × radius | circle area = \( \pi \times \text{square of the radius} \) |
|-----------------------|----------------------------------|-------------------------------|---------------------------------|---------------------|
| Physical Logics (4016) | speed = distance ÷ time | distance = speed × time | time = distance ÷ speed | workload = time × work speed |
| Financial Logics (1570) | expenses = price × quantity | insurance cost = insurance amount × insurance rate | income after taxes = income before taxes – tazes | concentration = solve weight ÷ solution weight |
| Commonsense Logics (3852) | average = total ÷ number of units | total = average × number of units | number of units = total ÷ average | segment number = interval points excluding both ends + 1 |

Table 5: Example logic formulas of different skills.

The basic statistics of our InterMWP dataset are shown in Table 6. Figure 6 illustrates the distribution information about word-level question length, char-level question length, and expression tree length. For those problems with multi solutions, we take the shortest solution expression to count. From Figure 6, we can observe that the lengths of most of the questions are adequate, which are not too long to understand for an MWP Solver. Besides, most expression tree contains less than 3 operators, which suggests that the questions should not very difficult to reason. However, the long tail in the distribution requires the MWP solvers to understand the complex mathematical relationships in the textual content.

| Total | Train | Val | Test |
|-------|-------|-----|------|
| Questions | 11,495 | 9,483 | 1,000 | 1,000 |
| Sentences | 16,308 | 13,456 | 1,408 | 1,444 |
| Words | 316,620 | 261,700 | 27,048 | 27,872 |

Table 6: Basic statistics of our InterMWP dataset.

There are 210 algebraic knowledge formulas en-
tailed in InterMWP. We list the most and least frequent knowledge formulas with a frequency greater than 5 in Table 7. It is shown that the distribution of formulas is not balanced but it is consistent with the real-world scene.

| formulas                                      | %    |
|-----------------------------------------------|------|
| Common-sense step                             | 56.35|
| average per unit = total number / number per unit | 4.75 |
| total number = average number per unit × number of units | 4.75 |
| number per unit = total number / average number per unit | 2.83 |
| increased price rate = 1 + price increment ratio | 0.06 |
| increased price = original price / increased price rate | 0.04 |

Table 7: Formulas statistics of our InterMWP dataset.

redundancies. Finally, two annotators were asked to check the correctness of the annotated data from other annotators by conducting statistical sampling. When labeling the full solution to the test set, we use three operations to ensure the coverage of the full solution as far as possible: 1) The left and right sides of the symmetric operators (+, *) are recursively exchanged to generate new expressions; 2) Using sympy to obtain the simplified expressions and then operation 1 will be carried out; 3) New expressions are manually marked and then operation 1 and 2 are conducted. If the correctness of an annotator’s data is less than 96% accurate, the data will be discarded.

C Implementation Details

In our LogicSolver, the size of word embeddings and all hidden states for other layers are all set as 768, following the configuration of BERT-base (Devlin et al., 2019). In each epoch, all training data is shuffled randomly and then cut into mini-batches. BERT models in LogicSolver are initialized by pre-trained BERT-wwm (Cui et al., 2020) for Chinese. Our LogicSolver is optimized by ADAM optimizer (Kingma and Ba, 2015) with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1e^{-8}$. The mini-batch size is set as 32, 32, and 32 for the retriever, encoder-decoder, and logic generator respectively. The initial fine-tuning learning rate is set as $1e^{-5}$ and $1e^{-4}$ for pre-trained BERT models and tree-decoder and then decreases to half every 25 epochs. To prevent overfitting, we set the dropout rate as 0.5 and weight decay as $1e^{-5}$. The training epochs are set as 20, 100, and 100 for retriever, encoder-decoder, and logic generator respectively. During solution expression generation, we use the beam search algorithm to generate expression trees and predict logic formulas.

D Experiments on logic formulas

We also evaluate the formula accuracy and logic accuracy on the samples contain top-10 logic formulas in the test split of InterMWP dataset. The
results are shown in Table 8. The performance gap of our LogicSolver relative to GTS(Bert) is significant on most logic formulas.

| Logics | formula acc | logic acc |
|--------|-------------|-----------|
| total number = average number per unit \times number of units | 77.9/78.6 | 53.6/68.6 |
| total number = number of units \times average number per unit | 77.3/78.7 | 53.9/68.8 |
| average number per unit = total number \div number of units | 76.2/78.6 | 54.8/76.2 |
| expenses = quantity \times price | 74.5/76.5 | 47.1/66.7 |
| number of unit = total number \div average number per unit | 76.8/78.3 | 62.3/71.0 |
| expenses = price \times quantity | 74.0/76.0 | 48.0/68.0 |
| Working speed = workload \div time | 71.4/76.2 | 42.9/71.4 |
| distance = speed \times time | 74.1/71.6 | 66.4/65.2 |
| speed = distance \div time | 76.7/74.4 | 67.4/67.4 |
| Rectangle area = length \times width | 56.4/56.4 | 41.0/46.2 |

Table 8: Formula accuracy and logic accuracy on the samples contain top-10 logic formulas with the most occurrences in the test split. (To the left of the semicolon '/' is the result of GTS(Bert), and to the right is the result of LogicSolver.)