Math Word Problem Solving by Generating Linguistic Variants of Problem Statements

Systems and Software Lab (SSL)
Department of Computer Science and Engineering
Islamic University of Technology, Dhaka, Bangladesh
{rifatraiyan, nafisfaiyaz, jawadkabir, hasan, hasank}@iut-dhaka.edu

Abstract

The art of mathematical reasoning stands as a fundamental pillar of intellectual progress and is a central catalyst in cultivating human ingenuity. Researchers have recently published a plethora of works centered around the task of solving Math Word Problems (MWP) — a crucial stride towards general AI. These existing models are susceptible to dependency on shallow heuristics and spurious correlations to derive the solution expressions. In order to ameliorate this issue, in this paper, we propose a framework for MWP solvers based on the generation of linguistic variants of the problem text. The approach involves solving each of the variant problems and electing the predicted expression with the majority of the votes. We use DeBERTa (Decoding-enhanced BERT with disentangled attention) as the encoder to leverage its rich textual representations and enhanced mask decoder to construct the solution expressions. Furthermore, we introduce a challenging dataset, P\_ARA\_MAWPS, consisting of paraphrased, adversarial, and inverse variants of selectively sampled MWPs from the benchmark M\_AWPS dataset. We extensively experiment on this dataset along with other benchmark datasets using some baseline MWP solver models. We show that training on linguistic variants of problem statements and voting on candidate predictions improve the mathematical reasoning and robustness of the model. We make our code and data publicly available.

1 Introduction

Math word problem solving is a long-standing research problem in Artificial General Intelligence (AGI) and a lot of studies about this topic, from both industry and academia, have been published recently. A typical Math Word Problem (MWP) takes the form of a written narrative that articulates a problem scenario and poses a question regarding one or more unknown quantities. A language model capable of solving such problems has to translate the human-readable problem statement to a valid mathematical expression that can be evaluated to obtain the numeric answer. An example of a classic MWP is portrayed in Table 1, where the reader is asked to infer the revenue of a boutique shop. Such problems are generally found in math textbooks of 1st to 8th grade students and are easily solvable by humans with decent mathematical aptitude.

A lot of challenges manifest while designing an automated system for solving these problems (Zhang et al., 2019; Sundaram et al., 2022). The primary challenge is to understand the quantities in the problem and capture their complex mathematical interconnections from a linear textual sequence written in natural language. There exists a diverse range of MWPs with differing difficulty levels, i.e., varying numbers of unknown values, and depth of the relationships between quantities, which require good mathematical reasoning ability to solve. Furthermore, the absence of crucial information and the presence of irrelevant information in the problem statements proves to be quite a challenge for the solver models (Patel et al., 2021). Other challenges include learning to tackle the chronological and temporal ambiguities of the events happening in the problem statements and dealing with MWPs that significantly differ from the training set in terms of semantic and syntactic structure.

To address the problem outlined in Table 1, a competent MWP solver model would need to possess the ability to associate the quantity, i.e., 69 handbags, with its price attribute of $13, and un-

Table 1: An example of a Math Word Problem.

| Problem: 69 handbags are sold for $13 each. There are a total of 420 handbags in a boutique and the remaining handbags are sold for $7 each. How much did the boutique earn after selling all the handbags? |
| Expression: $x = 69 \times 13 + (420 - 69) \times 7$ |
| Solution: 3354 |
derstand the relative arithmetic order by deriving 351 remaining handbags, i.e., 420 − 69, before associating the price attribute of $7. A lot of psychological studies have been done on how human beings learn to solve mathematical problems and improve their aptitude (Piaget, 2013; Peterson et al., 2003; Kingsdorf and Krawec, 2016). The frontier of research involving MWP solving is considered a momentous step towards the apogee of AGI (Bubeck et al., 2023) and so researchers have dedicated their efforts to replicating these complex cognitive patterns exhibited by human beings within the frameworks of AI models. The existing methods that are considered strong baselines for MWP solving can be demonstrably shown to use shallow heuristics to solve many of the MWPs in the benchmark datasets (Patel et al., 2021) creating a faux impression of their mathematical reasoning capability. To account for this limitation, in this paper —

- We propose a framework for solving simple math word problems by generating paraphrased linguistic variants of the input problem statement using OpenAI’s latest Generative Pre-trained Transformer (GPT-3) (Brown et al., 2020) models, namely text-davinci-003 and gpt-3.5-turbo. The problem statement variants along with the original problem text then undergo the appropriate pre-processing steps and are fed to an MWP solver model with a DeBERTa-based encoder and Enhanced Mask decoder.
- We also generate a large, augmented version of the MAWPS (Koncel-Kedziorski et al., 2016) dataset, namely PARAMAWPS (Paraphrased MAth Word Problem Solving Repository), as a challenging dataset by the introduction of paraphrased structural variations of almost all categories of problems, but emphasizing more on the categories that the strong baseline models find difficult to solve.

DeBERTa (Decoding-enhanced BERT with disentangled attention) (He et al., 2020) is currently one of the most popular language models due to its effectiveness in achieving state-of-the-art results on a variety of natural language processing tasks, including language translation, text classification, and question answering. In our work, we find that the DeBERTa model achieves value accuracies of 63.5% and 91.0% on the SVAMP dataset (Patel et al., 2021) and the MAWPS dataset (Koncel-Kedziorski et al., 2016) respectively. It falls behind the current SOTA accuracy of ROBERTA-DEDUCREASONER (Jie et al., 2022) by a slight margin of 1 ± 0.20% on the MAWPS dataset, but exceeds its accuracy of 47.3 ± 0.20% on the SVAMP dataset. Our code and data are publicly available at — https://github.com/Starscream-11813/Variational-Mathematical-Reasoning

2 Problem Formulation

A Math Word Problem $S$ is a sequence of word tokens and numeric values, where the $V_S = \{v_1, \ldots, v_m\}$ denotes the word tokens in $S$ and the set $n_S = \{n_1, \ldots, n_l\}$ denotes the set of numeric quantities in $S$. The set of word tokens $V_S$ consists of entities such as names of people, objects, units, and rates while the set of quantities $n_S$ consists of the numerical amount relevant to those entities.

The goal of an MWP solver model is to map $S$ to a valid mathematical expression $E$, consisting of the quantities in $(n_S \cup C)$, where $C$ is a set of constants, and the fundamental mathematical operators $O = \{\ +, \ - , \ \times , \ \div \}$, which can be evaluated to obtain the correct answer.

3 Literature Review

3.1 Math Word Problem Solving

3.1.1 Preliminary Works

The dawn of research on MWP solving was in the mid-1960s (Feigenbaum et al., 1963; Bobrow, 1964). Rule-based methods (Fletcher, 1985; Berman, 2007; Yuhui et al., 2010) are chronologically some of the earliest approaches to solving MWPs. They use a set of manually hard-coded rules about the language they are analyzing to find out regularities in the data. Statistical methods (Kushman et al., 2014; Hosseini et al., 2014; Roy et al., 2015; Zhou et al., 2015; Mitra and Baral, 2016; Liang et al., 2016a,b) use generic ML classifiers to extract the entities, quantities, and operators from the problem statement and infer the numeric answer with simple logic. Tree-based methods (Koncel-Kedziorski et al., 2015; Roy and Roth, 2016; Roy et al., 2016; Roy and Roth, 2017) utilize the inherent binary tree-like structure of expressions/equations. Other primitive categories of approaches that have now been rendered somewhat obsolete are Parsing-based methods (Shi et al.,
3.1.2 Deep Learning-based Methods

Currently, the landscape of Deep learning models for the MWP solving task is primarily comprised of five distinct paradigms, \textit{SEQ2SEQ}-based, \textit{SEQ2TREE}-based, \textit{GRAPH2TREE}-based, complex relation extraction-based, and \textit{Large Language Model (LLM)} prompt-based approaches, each of which has demonstrated remarkable levels of performance and efficacy. Wang et al. (2017) were the pioneers of introducing deep learning to solve MWPs with their proposed \textit{SEQ2SEQ} model. To improve the \textit{SEQ2SEQ} model, researchers resorted to alternative strategies, such as reinforcement learning techniques (Wang et al., 2018b; Huang et al., 2018), using dense problem representation (Mishra et al., 2018), and incorporating group attention mechanisms (Li et al., 2019). Xie and Sun (2019) were the progenitors of the novel Goal-driven Tree-Structured (GTS) model, designed to generate expression trees using the tree-based decoder in order to imitate the goal-driven problem-solving approach of humans. The use of this tree decoder along with pre-trained language models, such as BERT (Devlin et al., 2018), BART (Lewis et al., 2019), RoBERTa (Liu et al., 2019b), as the encoder in some of the \textit{SEQ2TREE} approaches (Liu et al., 2019a; Shen and Jin, 2020; Wu et al., 2020; Lin et al., 2021; Shen et al., 2021; Liang et al., 2021; Liang et al.; Li et al., 2021; Xiong et al., 2022) brought about substantial performance improvements over the previous \textit{SEQ2SEQ} methods. Cao et al. (2021) devised a directed acyclic graph (\textit{SEQ2DAG}) model of the equations for the purpose of extracting the expression. Zhang et al. (2020a) incorporated the idea of Knowledge Distillation (KD) (Hinton et al., 2015) in their proposed model where the teacher network is pre-trained to guide the learning behaviors of the student networks. Yu et al. (2021) introduced 2 types of encoders in their model. Hong et al. (2021) modified the work of Xie and Sun (2019) by incorporating a symbolic reasoning based \textit{Learning-by-fixing} (LBF) framework. Huang et al. (2021) attempted to emulate human-like analogical learning in their proposed memory-augmented model. \textit{GRAPH2TREE}-based approaches (Zhang et al., 2020b; Li et al., 2020) fused the merits of Graph-based Transformer (Yun et al., 2019; Cai and Lam, 2020) encoders with multiple Graph Convolutional Network (multi-GCN) modules (Kipf and Welling, 2016), and tree-based decoders to solve MWPs. Chatterjee et al. (2021) introduced a weakly supervised approach for MWP solving. Li et al. (2021) introduced a contrastive learning approach with pattern divergence to solve MWPs. Jie et al. (2022) formulated the MWP solving task as a complex relation extraction problem and leveraged explainable deductive reasoning techniques to iteratively construct the target equations.

With the advent of LLMs, many innovative prompt-based methods (Shao et al., 2022; Li et al., 2022; Wang et al., 2022; Pi et al., 2022; Chen et al., 2022; Liang et al., 2023) of solving MWPs that capitalize on the models’ exceptional few-shot learning capability came into the limelight and demonstrated good performance across numerous benchmark datasets. Cobbe et al. (2021) used verifiers with their GPT-3 (Brown et al., 2020) model. Although LLMs excel at natural language understanding and have serendipitous emergent reasoning abilities (Yang et al., 2023), they are still lackluster in complex reasoning tasks (Huang and Chang, 2022). Numerous studies on complex reasoning tasks have empirically demonstrated that the approach of fine-tuning smaller models is more effective (Ho et al., 2022) than adopting LLM prompting techniques like Chain of Thought (CoT) prompting (Wei et al., 2022).

3.2 Paraphrasing

Paraphrase generation has garnered significant attention from various NLP approaches, encompassing rule-based methods (McKeown, 1980; Meteer and Shaked, 1988), data-driven techniques (Madnani and Dorr, 2010), linguistic translation methods (Bannard and Callison-Burch, 2005; Barzilay and McKeown, 2001; Prakash et al., 2016) that leverage bilingual corpora for iterative refinement (Madnani and Dorr, 2010; Prakash et al., 2016; Mallinson et al., 2017). Witteveen and Andrews (2019) demonstrated the superiority of LLMs like GPT-3 over the preceding methods in the paraphrasing task.

Accordingly, our work attempts to leverage the strengths of GPT-3 to generate a more linguisti-
cally diverse pool of problem statements to fine-tune a relatively smaller DeBERTa solver model on the downstream task of MWP solving which falls under the rubric of complex reasoning tasks.

4 Methodology

Figure-1 in Appendix-A shows an overview of our proposed architecture. Given a problem statement $S$, we prompt the paraphraser model to generate $k$ linguistic variants of $S$ which are, $S_1, S_2, \ldots, S_k$. These $k$ variant problems along with the seed problem $S$ consists of quantities that are tagged appropriately using quantity tags. Each of the $k + 1$ text sequences is then tokenized and the content embeddings $H$ and positional embeddings $P$ of the tokens are fed to the DeBERTa model. The dis-entangled self-attention mechanism of DeBERTa’s encoder utilizes $H$ and $P$ to generate the output $H_{output}$, which is a contextual representation of the content of each problem statement. $H_{output}$, along with the relative positional embeddings $P$ and absolute positional embeddings $I$ of each of the problem statements are used by the Transformer layers of Enhanced Mask Decoder (EMD) of DeBERTa to generate the $k + 1$ predicted equations $E_1, E_2, \ldots, E_{k+1}$. These equations are then simplified and the equation that is predicted the most number of times is elected as the final prediction of the model. This majority voting module is used only during the validation/testing phase and for inference. During the training phase, the $k + 1$ problem statements are deemed as stand-alone training samples and the Negative Log-Likelihood loss (NLLLoss) is calculated using the predicted equations and the ground-truth equation. Consequently, if the training set of the dataset used to train the model consists of $n$ samples, it is as if the model is trained with $(k + 1) \times n = kn + n$ samples. The knowledge points gathered after being trained on an extra $kn$ samples contributes to the robustness of the model.

4.1 Paraphrasing Model

The task of correctly reformulating a Math Word Problem statement requires a good level of language understanding which is not present in its entirety in rule-based and data-driven methods of paraphrasing rendering them unsuitable in this case. These methods frequently yield incorrect, incoherent, and grammatically inaccurate linguistic variations; sometimes even leaving out crucial numerical information. Accordingly, we choose text-davinci-003 and gpt-3.5-turbo, two GPT-3 models from OpenAI, as the paraphrasing models. GPT-3 (Generative Pre-trained Transformer 3) (Brown et al., 2020) is a large language model with 175 billion parameters, that is capable of performing a wide range of natural language processing tasks, including paraphrasing a given sentence. Upon being prompted, it restates a given problem statement in different words while still maintaining the original meaning. To select the most appropriate paraphrase, GPT-3 uses a scoring mechanism that evaluates the semantic similarity between the original sentence and each of the generated paraphrases. The model assigns a higher score to paraphrases that are more similar in meaning to the input sentence, based on its understanding of the context and the relationships between the words. It also allows users to customize the level of complexity and the style of writing in the paraphrased version. We generate $k$ variants of the original problem text by prompting the model.

4.1.1 Prompts and System Task Description

The prompts that we use for accomplishing our linguistic variant generation task are,

- **system role Task Description** —
  You are a Math Word Problem rephraser that generates variations of math word problem statements.

- **user role Prompts** —
  - Generate $k_1$ paraphrased variations of the problem by changing the sentence structure.
  - Generate $k_2$ paraphrased variations of the problem by changing the named entities and objects.
  - Generate $k_3$ paraphrased variations of the problem with irrelevant numerical information.

Here, the total number of linguistic variants of a problem, $k = k_1 + k_2 + k_3$ and $5 \leq k \leq 15$.

A detailed discussion on the types of problem variations is delineated in Section-5.

4.2 Quantity Tagging

All the quantities (written either numerically or in words) in every single variant of the problem along with the original problem itself, are tagged with unique quantity tags using RegEx and a Python script which is provided in our GitHub repository (see Section-1). This quantity tagging step ensures that the same quantity is present in both the input
as well as in the output. The quantity-tagged tokens have their own content and positional embeddings. For example, if the problem statement is,

“Melanie picked 4 plums, Dan picked 9 plums, and Sally picked 3 plums from the plum tree. How many plums were picked in total?”

then the quantity-tagged version of the problem statement is,

“Melanie picked [Q1] plums, Dan picked [Q2] plums, and Sally picked [Q3] plums from the plum tree. How many plums were picked in total?”

We use this quantity tagging for the ground truth equation’s quantities as well.

4.3 Encoder

We use the pre-trained language model DeBERTa (Decoding enhanced BERT with disentangled attention). DeBERTa is a newly developed neural language model by He et al. (2020) that is based on the Transformer architecture. It boasts a significant advancement over previous state-of-the-art (SOTA) pre-trained language models (PLMs) due to the incorporation of two novel techniques. The first technique is a disentangled attention mechanism and the second technique is an enhanced mask decoder. Together, these techniques make DeBERTa a highly effective PLM that outperforms its predecessors on a wide range of NLP downstream tasks.

4.3.1 Disentangled Attention

Contrary to BERT, which utilizes a vector representation for each word in the input layer by summing its content and position embeddings, in DeBERTa, every word is represented by two separate vectors that encode its content and position individually. The attention scores between words are computed using separate matrices that are disentangled based on the content and relative position of each word. This design choice is based on the observation that the attention weight between a pair of tokens is influenced by both their content and in tandem their relative positions. This especially holds paramount importance for the task of MWP solving as the relative positions of certain keywords in the problem statements dictate the solution.

To represent a token \( x_i \) located at a specific position \( i \) within a given sequence, it employs two distinct vectors, \( H_i \) and \( P_{ij} \), which are respectively the content and relative positional representation vectors of \( x_i \) with respect to a token \( x_j \) at position \( j \). The inter-token attention weights between \( x_i \) and \( x_j \) can be broken down into four constituent components,

\[
A_{ij} = (H_i, P_{ij}) \times (H_j, P_{ji})^\top
= H_i H_j^\top + H_i P_{ji}^\top + P_{ij} H_j^\top + P_{ij} P_{ji}^\top
\]

(1)

where, the four disentangled matrix attention scores represent their contents and positions as content-to-content (C2C), content-to-position (C2P), position-to-content (P2C), and position-to-position (P2P). The P2P portion of (1) is somewhat rendered obsolete since DeBERTa uses relative positional embedding which is why no useful information can be extracted from it.

The self-attention mechanism described by Vaswani et al. (2017) has 3 parameters, \( Q \) (Query), \( K \) (Key), and \( V \) (Value). The non-contextual embedding that is being contextualized at any point requests for information from its surrounding tokens within the context window and that is represented by the query token, and the tokens that the model pays attention to are the key tokens.

\[
Q_e = HW_{eQ}, K_e = HW_{eK}, V_e = HW_{eV}
\]

\[
Q_r = PW_{rQ}, K_r = PW_{rK}
\]

(2)

where, \( W_{eQ} \in \mathbb{R}^{d \times d} \), \( W_{eK} \in \mathbb{R}^{d \times d} \), \( W_{eV} \in \mathbb{R}^{d \times d} \) are the projection weight matrices for the projected content vectors \( Q_e, K_e, V_e \) respectively. Similarly, \( W_{rQ} \in \mathbb{R}^{d \times d} \) and \( W_{rK} \in \mathbb{R}^{d \times d} \) play the role of projection matrices for the projected relative position vectors \( Q_r \) and \( K_r \). The metric to calculate the relative distance between tokens \( x_i \) and \( x_j \) is,

\[
\delta(i, j) = \begin{cases} 
0, & \text{if } i - j \leq k \\
2k - 1, & \text{if } i - j \geq k \\
i - j + k, & \text{otherwise}
\end{cases}
\]

(3)

which implies, \( \delta(i, j) \in [0, 2k) \). Each element \( \bar{A}_{ij} \) of the attention matrix \( \bar{A} \) denotes the attention score from token \( x_i \) to the token \( x_j \) and is computed using the vectors defined in (2) in the following manner,

\[
\bar{A}_{ij} = \sum_{C2C} Q_i^C K_j^C + \sum_{C2P} Q_i^C K_{ij}^P + \sum_{P2C} P_{ij}^C Q_{ij}^C
\]

(4)

The attention score is yielded using the dot product of the query and key in the formula to let
the model have an idea of how similar the key is to
the query. The output of the self-attention mecha-
nism, which is denoted by $H_{output} \in \mathbb{R}^{N \times d}$ is,

$$H_{output} = \text{softmax} \left( \frac{A}{\sqrt{3d}} \right) V_c$$

(5)

The result of the dot-product is normalized by di-
viding with $\sqrt{3d}$ to avoid very hard softmax with
small gradients, which is especially required for
training stability in the case of large-scale PLMs
(Vaswani et al., 2017; He et al., 2020).

4.4 Decoder

He et al. (2020) postulates that the premature inte-
gration of absolute positions, which is employed by
BERT (Devlin et al., 2018) in its decoding
phase, could potentially impede the model’s ability
to acquire adequate knowledge of relative rela-
tionships. With this as the justification, DeBERTa,
being a model that was pre-trained using MLM
(Masked Language Modeling), uses the absolute
positions of the tokens in the penultimate layer,
right before the softmax layer during the masked
token prediction in its decoding phase. This en-
ables all the Transformer layers in the decoder to
work with the relative positional information with-
out the susceptibility of hampering the learning
process of the model. Since the absolute positions
of the tokens in a sentence highly influence the
nuanced understanding of the sentence’s semantic
and syntactic structure, and extracting information
from only the relative positions isn’t sufficient, the
absolute positions are incorporated in the tail-end
of the pipeline in the case of DeBERTa. This is
why DeBERTa’s decoding module is dubbed an
Enhanced Mask Decoder (EMD) and it demonstra-
ively outperforms the decoder counterparts of its
predecessor PLMs (He et al., 2020).

4.5 Majority Voting

Since there can be multiple valid equations for
a single MWP, each of the $k + 1$ predictions
from the decoder, $E_1, E_2, \ldots, E_{k+1}$, is simplified
to a reduced normal form using the python pack-
age sympy\(^1\). These $k + 1$ simplified predictions,
$E'_1, E'_2, \ldots, E'_{k+1}$, are then counted and the prediction
that is the most frequent or that is yielded the
most number of times is elected as the final an-
swer of the whole solver model. It is to be noted
that this voting mechanism is used only during the

\[
E^* = \operatorname{argmax}_{E'} \text{Votes}(E'_i); \quad i = 1, 2, \ldots, k + 1
\]

(6)

5 Experiment

5.1 Data Acquisition

We introduce a new large-scale dataset, namely
PARAMAWPS (Paraphrased MATH Word
Problem Solving Repository), consisting of
16,278 single equation MWPs. It is gener-
ated as a by-product of using one of the most
commonly-used English MWP datasets, MAWPS
(Koncel-Kedziorski et al., 2016) which consists
of a total of 2,373 problems, and the paraphraser
model. We save the generated paraphrased vari-
ants of selectively sampled problems of MAWPS
and also manually include inverse versions of the
problems to create our dataset. The dataset con-
tains all the problems from the original MAWPS
dataset as well as paraphrased versions of some of
the more challenging problems within MAWPS,
hence the name, PARAMAWPS. The samples are
manually checked for correctness by 3 under-
graduate students. By generating variations of
some of the more difficult problems, we intend to
increase familiarity of challenging concepts found
within those problems to any model trained over
this data, as well as more thoroughly challenge
existing models trained on datasets that do not
provide said complexity at an equal or higher
density. We generate $k$ problems from each seed
problem in the dataset, adding up to a total of
$k + 1$ problems, where $5 \leq k \leq 16$. Each of the
$k$ generated problems will be a variation on the
original that will feature several changes to the
problem text. We generate 4 types of variations of
each seed problem (see Table-7 in Appendix-A).

- **Changed phrase order** — Variations with
the order of the phrases being changed facil-
itate a break from the standard problem state-
ment template where quantities are generally
given before the question formulation. Hav-
ing a changed ordering of phrases makes apri-
ori question formulations more common.

- **Changed object and entity names** — Ob-
ject and entity names are altered with inter-
changeable alternatives (names, synonyms) in
problem variations to prevent fixation on ele-
ments of the problem mostly agnostic to

\[^1\text{https://www.sympy.org/en/index.html}\]
the process of solving the problem. It also serves to prevent an increase in density for similar terms that originate from the seed problem yielding good problem samples for language models (Lee et al., 2021).

- **Added unrelated information** — Some variations contain an extra phrase or quantity, or similar additions that are in excess of the information required to solve a problem and do not affect the original problem formulation in any meaningful way. These adversarial variations serve to obfuscate and familiarize the models with only the necessary information, enhancing deductive abilities (Kumar et al., 2021).

- **Inverted question** — Some variations will take a previously known quantity and turn it into an unknown quantity while revealing the previous unknown quantity of the problem. This, in many cases, alters the question drastically, changing the needed calculations and equations, while keeping a roughly similar question body to the seed problem. Liu et al. (2021) used such problem samples in their work.

5.1.1 Seed Problems

Many of the seed problems used to generate variations from MAWPS pose sufficient difficulty to even SOTA MWP solvers and often contain numeric information embedded within the statement itself. An example is the following problem,

"Mary, Sam, Keith, and Alyssa each have 6 marbles. How many marbles do they have in all?"

This problem yields the equation 
\[ x = 4 \times 6 \]

that yields the equation 
\[ x = (1.0 - (1.0 + (10.0 \times 0.01)) \times (1.0 - (10.0 \times 0.01))) \times 100.0 \].

Another problem with similar inferences of a more complex nature is,

"Lauren wants to mix 5 liters of 7% milk with skim-milk (0% fat) to produce a mixture of 2.9787% milk. How much skim-milk should Lauren add?"

yielding the equation 
\[ x = (7.0 \times 0.01) \times 5.0 / (2.9787 \times 0.01) - 5.0 \], containing similar conversions of percentages, as well as additional knowledge of types of mixtures. Here, 7% milk is mixed with pure milk, or 100% milk. Yet the only indication that the milk is of 100% purity is nowhere to be seen in a direct capacity in the problem, but rather in a roundabout way - by referring to the amount of fat (0%) rather than the purity of the milk. Models have to infer a vast amount of real-world contextual knowledge to be able to solve such problems. Problems with second-degree unknown quantities are also present as seed problems. For example, the problem

"The Hudson River flows at a rate of 3 miles per hour. A patrol boat travels 60 miles upriver and returns in a total time of 9 hours. What is the speed of the boat in still water?"

that yields the equation
\[ (60.0 / (x - 3.0)) + (60.0 / (3.0 + x)) = 9.0 \], which is a quadratic equation. The problem itself deals with calculations of speed, which requires knowledge of how speed is calculated given certain quantities, as well as the effect of certain elements in the problem scenario on speed.

We resort to this data generation approach due to the lack of large-scale, diverse, single-equation English MWP datasets. Other commonly-used benchmark datasets, MATH23K (Wang et al., 2017) and APE210K (Liang et al., 2021) consist of math problems written in Chinese Mandarin. We also aim to diversify the samples in MAWPS to enable better training for MWP solvers (Schick and Schütze, 2021; Kumar et al., 2022). SVAMP, created by Patel et al. (2021) consists of challenging versions of problems and is considered a challenge set for testing the robustness of MWP solvers. We use the original version of MAWPS and SVAMP along with our dataset PARAMAWPS.
for conducting our experiments. A comparative summary of the statistics of the datasets used is shown in Table-2 and their operator count distributions are portrayed in Figure-2.

Table 2: Comparison of the datasets used.

<table>
<thead>
<tr>
<th>Properties</th>
<th>SVAMP</th>
<th>MAWPS</th>
<th>PARAMAWPS</th>
</tr>
</thead>
<tbody>
<tr>
<td># of problems</td>
<td>1.000</td>
<td>2.373</td>
<td>16.278</td>
</tr>
<tr>
<td># of unique templates</td>
<td>27</td>
<td>159</td>
<td>215</td>
</tr>
<tr>
<td>Avg. # of operators</td>
<td>1.236</td>
<td>1.606</td>
<td>1.68</td>
</tr>
<tr>
<td>Avg. # of quantities per prob.</td>
<td>2.81</td>
<td>2.57</td>
<td>2.54</td>
</tr>
<tr>
<td>Avg. # of quantities per eq.</td>
<td>2.23</td>
<td>2.59</td>
<td>2.67</td>
</tr>
<tr>
<td># of problems with constants</td>
<td>0</td>
<td>185</td>
<td>3313</td>
</tr>
</tbody>
</table>

Table 3: Value accuracy of the DeBERTa model and various baseline models.

<table>
<thead>
<tr>
<th>Methods</th>
<th>MAWPS (%)</th>
<th>SVAMP (%)</th>
<th>PARAMAWPS (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS</td>
<td>39.5</td>
<td>22.1</td>
<td>71.2</td>
</tr>
<tr>
<td>Math-EN</td>
<td>69.2</td>
<td>21.8</td>
<td>71.6</td>
</tr>
<tr>
<td>GROUP-ATT</td>
<td>76.1</td>
<td>19.2</td>
<td>70.8</td>
</tr>
<tr>
<td>RNNEncDec</td>
<td>79.4</td>
<td>25.4</td>
<td>73.6</td>
</tr>
<tr>
<td>RNNVAE</td>
<td>79.8</td>
<td>25.9</td>
<td>72.8</td>
</tr>
<tr>
<td>Transformer</td>
<td>85.6</td>
<td>20.7</td>
<td>64.6</td>
</tr>
<tr>
<td>BERT</td>
<td>86.9</td>
<td>24.8</td>
<td>72.1</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>88.4</td>
<td>30.3</td>
<td>72.5</td>
</tr>
<tr>
<td>DeBERTa</td>
<td>90.7</td>
<td>63.5</td>
<td>74.1</td>
</tr>
<tr>
<td>DeBERTa w/ VM</td>
<td>91.0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>DeBERTa w/ VM x VM</td>
<td>-</td>
<td>-</td>
<td>79.1</td>
</tr>
</tbody>
</table>

Table 4: Value accuracy with different numbers of linguistic variants of the problem samples. † denotes 5-fold cross validation.

To gain insights into the individual contributions of the Paraphrasing Model and Voting Mechanism in conjunction with the DeBERTa model, we perform ablation studies. Table-4 shows the effect of increasing the number of generated problem variants to infer the solution expressions of the problem samples in the MAWPS dataset’s test set. Although there is a slight decrease in the accuracy for \( k = 5 \), we see a minuscule increase in accuracy for...
Table 6: Value accuracy of the LLMs in a zero-shot setup testing. † denotes evaluation on the whole dataset.

<table>
<thead>
<tr>
<th>Models</th>
<th>MAWPS† (%)</th>
<th>PARA-MAWPS† (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPT-J (6B)</td>
<td>9.9</td>
<td>3.9</td>
</tr>
<tr>
<td>text-babbage-001 (6.7B)</td>
<td>2.76</td>
<td>3.21</td>
</tr>
<tr>
<td>text-curie-001 (13B)</td>
<td>4.09</td>
<td>4.20</td>
</tr>
<tr>
<td>gpt-3.5-turbo (175B)</td>
<td>80.3</td>
<td>73.0</td>
</tr>
</tbody>
</table>

One of the most capable models in the GPT-3.5 series of models is text-davinci-003, with 175 billion parameters and the ability to follow instructions consistently and produce lengthy outputs. However, the most capable and up-to-date model according to OpenAI is gpt-3.5-turbo, with 175 billion parameters, which is primarily optimized for chat completions but can be tweaked to follow more specific instructions similar to text-davinci-003. While all models used are instructed to output in a specific format — ‘Answer: [ANS]’ with just the numerical value in the place of ‘[ANS]’, the ability to do so consistently deteriorated with the models with relatively fewer parameters. Out of the base GPT-3 models, the 13 billion parameters text-curie-001 can output in the given format relatively consistently, text-babbage-001 with 6.7 billion parameters can occasionally produce the output in the correct format, but tries to generate full sentences more often than not, whereas the 350 million parameters text-ada-001 can barely generate a single output in the correct format, choosing to generate full sentences almost all of the time. Models tend to try to 'work through' the problem in text form rather than just generating the output, although with gpt-3.5-turbo this can be mostly mitigated by using very specific instructions for the prompt. The results in Table 6 support the current weakness of LLMs in mathematical reasoning tasks and the suitability of fine-tuning smaller models. It indicates the improvement in performance for a well-reasoning, but comparatively small model when it has the option to democratically choose from a substantial number of solution guesses.

6 Conclusion and Future Work

In this paper, we propose the idea of an MWP solving framework that utilizes the paraphrased linguistic variations of problem texts to train a DeBERTa model that generates candidate solution expressions and finalizes the predicted math expression by employing majority voting on a set of simplified candidate expressions. Our findings demonstrate that incorporating linguistic variants of problem statements during training and utilizing a voting mechanism for candidate predictions enhance the model’s mathematical reasoning and overall robustness. We also introduce a large-scale, diverse, and challenging single-equation MWP dataset, PARA-MAWPS, consisting of paraphrased, inverse, and adversarial variants of selectively sampled datapoints from MAWPS, as a formidable evaluation test-bed and a proper benchmark for training MWP solver models. We wish to experiment further with harder problem text variations (e.g. grammatical errors) and conduct a thorough error analysis of the models for identifying their lapses in mathematical reasoning and discovering more scopes of improvement. We also aim to expand our research to encompass the intricate realms of multi-equation, multi-step deduction, and domain-knowledge problems. We hope our approach and findings will pave the way to more scholarly works on the vistas of AGI and in tandem be deemed a noteworthy and meaningful contribution to this domain of research.

7 Limitations

There are still some avenues of improvement in our work. The temporal overhead due to the problem variant generation by the paraphraser model may make our proposed architecture unsuitable for real-world applications even though it takes merely 10 to 12 seconds to generate $k = 5$ variants for a single sample. Another limitation of our work is the absence of a proper tie-breaking
strategy in our Majority Voting module. Furthermore, we need to introduce a system of weighted votes (e.g. semantic similarity scores as weights) so that the votes of wrongly predicted equations don’t trump that of correctly generated predictions. We also plan to incorporate and experiment with the Tree-based decoder (Xie and Sun, 2019) in our proposed pipeline.

References


Daniel G Bobrow. 1964. Natural language input for a computer problem solving system.


Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. 2021. Learning by fixing: Solving math word problems with weak supervision. In *AAAI Conference on Artificial Intelligence*.


Zhenwen Liang, Jipeng Zhang, Lei Wang, Wei Qin, Jie Shao, and Xianglei Zhang. Mwp-bert: A numeracy-augmented pre-trained encoder for math word problems.


in Natural Language Processing (EMNLP), pages 7137–7146.


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. IJCAI.


A Appendix

A.1 Dataset Split

We use an 80:10:10 train-validation-test split for our PARAMAWPS dataset. For MAWPS, we use 5-fold cross-validation using the splits provided by its authors Koncel-Kedziorski et al. (2016). The SVAMP dataset is a challenge set and all 1,000 of its samples constitute the test set while the model itself is trained on a combination of the MAWPS and ASDiv-A (Miao et al., 2021) dataset.

A.2 Performance Evaluation and Metric

We use Negative log-likelihood loss (NLLLoss) for training all the models. For the baseline models, MWPToolkit uses two metrics of accuracy, Equation Accuracy and Value Accuracy. Equation accuracy measures the correctness of the generated equation. Value accuracy measures the correctness of the value yielded from evaluating the generated equation. This metric takes into consideration the fact that models may generate equations that have a different template than the respective ground truth equations but nevertheless yield the correct answers to the problem statements.

A.3 Hyperparameters

In the DeBERTa model, we use embedding dimension $d = 768$, $FFN_{size} = 1024$, number of decoder layers $N = 4$, number of attention heads $h = 16$, dropout ratio $P_{drop} = 0.5$, learning rate $lr = 10^{-5}$, batch size $b = 8$, and $Epochs = 200$. The hyperparameters for the other baseline models are as set on the respective MWPToolkit implementations.

A.4 Optimizer

We use Adam (Kingma and Ba, 2014) with a StepLR learning rate scheduler as our optimizer. The learning rate $lr$ is set according to Vaswani et al. (2017), $lr = d^{-0.5} \cdot \min(n^{-0.5}, n \cdot w^{-1.5})$ where, $d$ is the embedding dimension, $n$ is the step number and $w$ is the number of warm-up steps. Here, warm-up steps $w$ simply insinuate that the learning rate rises linearly for the initial $w$ training steps. We set $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$ and $w = 1500$ for the models’ Adam optimizer. For the StepLR scheduler, we set $\gamma = 0.5$ and $step\_size = 5$. 

375
A.5 Hardware and Schedule

We have used the NVIDIA RTX 3090 GPU equipped with 25GB of VRAM and Intel Core i9 Processor for conducting our experiments. The DeBERTa model took around 18 hours to fully train on the PARAMAWPS dataset with 5-fold cross-validation and 200 epochs per fold, which was the highest expense of time among the lot. The other baseline models took approximately 7 to 9 hours on the PARAMAWPS dataset and around 5 hours on MAWPS and SvAMP. The greater the number of parameters that a model possesses the more time it takes to fully complete the 5-fold training process. As DeBERTa has an astounding 134 million parameters (He et al., 2020), it takes the longest time to train.
Figure 1: Overview of our proposed architecture.
Variation
Type Original Variation
Changed phrase order There were originally 20817 houses in Lincoln County. During a housing boom, developers built 97741. How many houses are there now in Lincoln County?

How many houses are there in Lincoln County now, after developers built an additional 97741 during a housing boom, when there were originally 20817 houses?

Changed object and entity names While playing a trivia game, Mike answered 3 questions correct in the first half and 5 questions correct in the second half. If each question was worth 3 points, what was his final score?

While playing a game of Hangman, Emily guessed 3 letters correctly in the first half and 5 letters correctly in the second half. If each letter was worth 3 points, what was her final score?

Added unrelated information A carpenter bought a piece of wood that was 8.9 centimeters long. Then he sawed 2.3 centimeters off the end. How long is the piece of wood now?

A carpenter bought a piece of wood that was 8.9 centimeters long. Then he sawed 2.3 centimeters off the end and sanded the wood for 20 minutes. How long is the piece of wood now?

Inverted question Mary bought 3 pizzas for $8 each. What was the total amount she paid for the 3 pizzas?

If Mary paid $24 for 3 pizzas, how much did she pay for each pizza?

Table 7: Types of Variations with examples. The problems in the Original column are samples taken from the MAWPS dataset, whereas, the ones in the Variation column are from the PARAMAWPS dataset.

<table>
<thead>
<tr>
<th>Variation Type</th>
<th>Original</th>
<th>Variation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Changed phrase order</td>
<td>There were originally 20817 houses in Lincoln County. During a housing boom, developers built 97741. How many houses are there now in Lincoln County?</td>
<td>How many houses are there in Lincoln County now, after developers built an additional 97741 during a housing boom, when there were originally 20817 houses?</td>
</tr>
<tr>
<td>Changed object and entity names</td>
<td>While playing a trivia game, Mike answered 3 questions correct in the first half and 5 questions correct in the second half. If each question was worth 3 points, what was his final score?</td>
<td>While playing a game of Hangman, Emily guessed 3 letters correctly in the first half and 5 letters correctly in the second half. If each letter was worth 3 points, what was her final score?</td>
</tr>
<tr>
<td>Added unrelated information</td>
<td>A carpenter bought a piece of wood that was 8.9 centimeters long. Then he sawed 2.3 centimeters off the end. How long is the piece of wood now?</td>
<td>A carpenter bought a piece of wood that was 8.9 centimeters long. Then he sawed 2.3 centimeters off the end and sanded the wood for 20 minutes. How long is the piece of wood now?</td>
</tr>
<tr>
<td>Inverted question</td>
<td>Mary bought 3 pizzas for $8 each. What was the total amount she paid for the 3 pizzas?</td>
<td>If Mary paid $24 for 3 pizzas, how much did she pay for each pizza?</td>
</tr>
</tbody>
</table>

Figure 2: Operator count distributions of PARAMAWPS, MAWPS, and SVAMP. We keep the distribution of PARAMAWPS somewhat similar to that of MAWPS to maintain a proper balance between easy and difficult problems.