Math Word Problem Solving by Generating Linguistic Variants of Problem Statements

Syed Rifat Raiyan, Md. Nafis Faiyaz, Shah Md. Jawad Kabir,
Mohsinul Kabir, Hasan Mahmud, Md. Kamrul Hasan
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, PARAMAWPS, consisting of paraphrased, adversarial, and inverse variants of selectively sampled MWPs from the benchmark MAWPS 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

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

Table 1: An example of a Math Word Problem.

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-

362

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 preprocessing 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 (**Para**phrased **MA**th **W**ord **P**roblem **S**olving 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-DEDUCTREASONER (Jie et al., 2022) by a slight margin of $1 \pm 0.20\%$ on the MAWPS dataset, but exceeds its accuracy of $47.3 \pm 0.20\%$ 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; Bakman, 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.,

2015; Zou and Lu, 2019), *Similarity-based methods* (Huang et al., 2016), and *Template-based methods* (Kushman et al., 2014; Zhou et al., 2015; Roy et al., 2016; Upadhyay et al., 2016; Huang et al., 2017).

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, SEQ2SEQ-based, SEQ2TREE-based, GRAPH2TREE-based, *complex relation extraction-based*, and *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 SEQ2SEQ model. To improve the 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), adopting template-based methodologies (Wang et al., 2019), 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 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 SEQ2SEQ methods. Cao et al. (2021) devised a directed acyclic graph (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 *Learning-by-fixing* (LBF) framework. Huang et al. (2021) attempted to emulate human-like analogical learning in their proposed

memory-augmented model. 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 disentangled 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 nu-

merical 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 (**D**ecoding **e**nhanced **BERT** with disentangled **a**ttention). 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 $P_{i|j}$, 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,

$$
\begin{aligned}
A_{ij} &= \langle H_i, P_{i|j} \rangle \times \langle H_j, P_{j|i} \rangle^\top \\
&= \underbrace{H_i H_j^\top}_{C2C} + \underbrace{H_i P_{j|i}^\top}_{C2P} + \underbrace{P_{i|j} H_j^\top}_{P2C} + \underbrace{P_{i|j} P_{j|i}^\top}_{\substack{P2P \\ (omitted)}}
\end{aligned} \tag{1}
$$

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.

$$
\begin{aligned}
Q_c &= HW_{c_Q}, K_c = HW_{c_K}, V_c = HW_{c_V} \\
Q_r &= PW_{r_Q}, K_r = PW_{r_K}
\end{aligned} \tag{2}
$$

where, $W_{c_Q} \in \mathbb{R}^{d \times d}$, $W_{c_K} \in \mathbb{R}^{d \times d}$, $W_{c_V} \in \mathbb{R}^{d \times d}$ are the projection weight matrices for the projected content vectors Q_c, K_c, V_c respectively. Similarly, $W_{r_Q} \in \mathbb{R}^{d \times d}$ and $W_{r_K} \in \mathbb{R}^{d \times d}$ 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,

$$
\delta(i,j) = \begin{cases} 0, & \text{if } i - j \leq k \\ 2k - 1, & \text{if } i - j \geq k \\ i - j + k, & \text{otherwise} \end{cases} \tag{3}
$$

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,

$$
\bar{A}_{ij} = \underbrace{Q_i^c K_j^{c\top}}_{C2C} + \underbrace{Q_i^c K_{\delta(i,j)}^{r\top}}_{C2P} + \underbrace{K_j^c Q_{\delta(j,i)}^{r\top}}_{P2C} \tag{4}
$$

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 mechanism, which is denoted by $H_{output} \in \mathbb{R}^{N \times d}$ is,

$$H_{output} = \mathbf{softmax}\left(\frac{\bar{A}}{\sqrt{3d}}\right) V_c \qquad (5)$$

The result of the dot-product is normalized by dividing 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 integration 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 positions. 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 enables all the Transformer layers in the decoder to work with the relative positional information without 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 demonstrably 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 package 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 answer of the whole solver model. It is to be noted that this voting mechanism is used only during the

---
[1] https://www.sympy.org/en/index.html

testing/validation phases or during inference.

$$E^* \leftarrow \underset{E'}{\operatorname{argmax}} \mathbf{Votes}(E'_i); \quad i = 1, 2, \ldots, k+1 \qquad (6)$$

5 Experiment

5.1 Data Acquisition

We introduce a new large-scale dataset, namely PARAMAWPS (**Para**phrased **MA**th **W**ord **P**roblem **S**olving Repository), consisting of 16,278 single equation MWPs. It is generated 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 variants of selectively sampled problems of MAWPS and also manually include inverse versions of the problems to create our dataset. The dataset contains 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 undergraduate 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 facilitate a break from the standard problem statement template where quantities are generally given before the question formulation. Having a changed ordering of phrases makes apriori question formulations more common.

- **Changed object and entity names** — Object and entity names are altered with interchangeable alternatives (names, synonyms) in problem variations to prevent fixation on elements of the problem mostly agnostic to

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 "$x = 4 \times 6$", despite the quantity 4 not being mentioned anywhere in the statement. This quantity had to be inferred from the other parts of the statement itself, namely, the 4 entities referred to in the statement; Mary, Sam, Keith, and Alyssa. Another such problem is,

> "*When the price of diesel rose by 10%, a user reduced his diesel consumption by the same amount. How much would his diesel bill change in terms of percentage?*"

which yields the complex equation of "$x = (1.0 - ((1.0 + (10.0 \times 0.01)) \times (1.0 - (10.0 \times 0.01)))) \times 100.0$". This problem, although seemingly simple

on the surface in terms of quantities described, has several calculations dictated through the problem statement, some of which require additional real-world anecdotal knowledge, such as the conversion of percentages. 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.

Properties	SVAMP	MAWPS	PARAMAWPS
# of problems	1,000	2,373	16,278
# of unique templates	27	159	215
Avg. # of operators	1.236	1.606	1.68
Avg. # of quantities per prob.	2.81	2.57	2.54
Avg. # of quantities per equ.	2.23	2.59	2.67
# of problems with constants	0	185	3313

Table 2: Comparison of the datasets used.

5.2 Model Implementation Details and Training

5.2.1 Baseline Models

We implement the DeBERTa model using Microsoft's *deberta-base* that is publicly available in Hugging Face[2]. The other baseline MWP solver models are implementations already available in the open-source MWPToolkit[3] developed by Lan et al. (2022). We use an extensive set of baseline models, Transformer (Vaswani et al., 2017), DNS (Wang et al., 2017), MathEN (Wang et al., 2018a), GroupATT (Li et al., 2019), RNNEncDec (Sutskever et al., 2014), RNNVAE (Su et al., 2018), BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019b), and compare them with the performance of the DeBERTa model. See Appendix-A for more training process details.

5.3 Result Analysis

Methods	MAWPS[†] (%)	SVAMP (%)	PARA-MAWPS[†] (%)
DNS	59.5	22.1	71.2
Math-EN	69.2	21.8	71.6
GROUP-ATT	76.1	19.2	70.8
RNNEncDec	79.4	25.4	73.6
RNNVAE	79.8	25.9	72.8
Transformer	85.6	20.7	64.6
BERT	86.9	24.8	72.1
RoBERTa	88.4	30.3	72.5
DeBERTa	90.7	**63.5**	74.1
DeBERTa$_{PM + VM}$	**91.0**	-	-
DeBERTa$_{VM}$	-	-	**79.1**

Table 3: Value accuracy of the DeBERTa model and various baseline models. † denotes 5-fold cross validation. PM stands for Paraphrasing Model and VM stands for Voting Mechanism.

Table-3 shows the performance comparison of the DeBERTa model and the baseline models mentioned in Section-5.2.1. The DeBERTa model coupled with the Paraphrasing model and the Voting

Mechanism outperforms all the baseline models in the MAWPS (Koncel-Kedziorski et al., 2016) dataset with an accuracy of 91.0%. The Paraphrasing Model and the Voting Mechanism contributed to a 0.3% increase in accuracy. The vanilla DeBERTa model also outperforms the baseline models in our PARAMAWPS dataset by boasting an accuracy of 74.1%. With the voting mechanism at the tail-end of the pipeline, we are able to yield an improvement of the accuracy by 5.04% making the accuracy 79.1%. We test the robustness of the vanilla DeBERTa model on the SVAMP (Patel et al., 2021) challenge dataset and get an accuracy of 63.5% which is quite higher than that of the other baseline models. The model still lags a mere $1 \pm 0.20\%$ behind the current SOTA model on MAWPS, which is the ROBERTA-DEDUCTREASONER model by Jie et al. (2022) ($92.0 \pm 0.20\%$) but supersedes its accuracy of $47.3 \pm 0.20\%$ on the SVAMP dataset.

The superiority of the model's accuracy in PARAMAWPS over SVAMP, despite the demonstrably greater difficulty of the MWP samples in PARAMAWPS, indicates that training a language model on a more diverse set of linguistically varied problem statements leads to a better quality mathematical reasoning ability after the training phase.

5.4 Ablation Study

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

# of variants	MAWPS[†] (%)
w/ $k = 0$	90.7
w/ $k = 5$	90.4
w/ $k = 10$	90.8
w/ $k = 15$	91.0

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

Voting Mechanism	PARAMAWPS[†] (%)
w/o VM	72.9, 74.1, 76.5, 72.1, 74.6
w/ VM	78.5, 77.8, 82.4, 77.2, 79.5

Table 5: Effect of Majority Voting on Value accuracy across all 5 folds. † denotes 5-fold cross validation.

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

$k = 10$ and $k = 15$. In Table-5 we see the impact of the Voting Mechanism which contributed to a 5.4% increase on average in the accuracy of the DeBERTa model on the PARAMAWPS dataset.

5.5 MWP Task Performance Analysis of Large Language Models

To test out the assertion made in other studies (Huang and Chang, 2022; Ho et al., 2022) about the incompetence of LLMs in complex reasoning tasks compared to fine-tuned smaller models, we use the GPT-J model and some of the presently used GPT-3 models by OpenAI to perform the task of MWP solving. We use the original version of MAWPS (Koncel-Kedziorski et al., 2016) along with our dataset PARAMAWPS for testing the mathematical reasoning of these models.

Models	MAWPS[†] (%)	PARA-MAWPS[†] (%)
GPT-J (6B)	9.9	5.9
text-babbage-001 (6.7B)	2.76	3.21
text-curie-001 (13B)	4.09	4.20
gpt-3.5-turbo (175B)	80.3	73.0

Table 6: Value accuracy of the LLMs in a zero-shot setup testing. † denotes evaluation on the whole dataset.

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 and Table-3 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 De-BERTa 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, PARAMAWPS, 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

Yefim Bakman. 2007. Robust understanding of word problems with extraneous information. *arXiv preprint math/0701393*.

Colin Bannard and Chris Callison-Burch. 2005. Paraphrasing with bilingual parallel corpora. In *Proceedings of the 43rd annual meeting of the Association for Computational Linguistics (ACL05)*, pages 597–604.

Regina Barzilay and Kathleen McKeown. 2001. Extracting paraphrases from a parallel corpus. In *Proceedings of the 39th annual meeting of the Association for Computational Linguistics*, pages 50–57.

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

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, et al. 2023. Sparks of artificial general intelligence: Early experiments with gpt-4. *arXiv preprint arXiv:2303.12712*.

Deng Cai and Wai Lam. 2020. Graph transformer for graph-to-sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 7464–7471.

Yixuan Cao, Feng Hong, Hongwei Li, and Ping Luo. 2021. A bottom-up dag structure extraction model for math word problems. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 39–46.

Oishik Chatterjee, Aashish Waikar, Vishwajeet Kumar, Ganesh Ramakrishnan, and Kavi Arya. 2021. A weakly supervised model for solving math word problems. *arXiv preprint arXiv:2104.06722*.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. 2022. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Edward A Feigenbaum, Julian Feldman, et al. 1963. *Computers and thought*. New York McGraw-Hill.

Charles R Fletcher. 1985. Understanding and solving arithmetic word problems: A computer simulation. *Behavior Research Methods, Instruments, & Computers*, 17(5):565–571.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Geoffrey Hinton, Oriol Vinyals, Jeff Dean, et al. 2015. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2(7).

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2022. Large language models are reasoning teachers. *arXiv preprint arXiv:2212.10071*.

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*.

Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. 2014. Learning to solve arithmetic word problems with verb categorization. In *EMNLP*, volume 523533. Citeseer.

Danqing Huang, Jing Liu, Chin-Yew Lin, and Jian Yin. 2018. Neural math word problem solver with reinforcement learning. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 213–223.

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 805–814.

Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying Ma. 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 (Volume 1: Long Papers)*, pages 887–896.

Jie Huang and Kevin Chen-Chuan Chang. 2022. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403*.

Shifeng Huang, Jiawei Wang, Jiao Xu, Da Cao, and Ming Yang. 2021. Recall and learn: A memory-augmented solver for math word problems. *arXiv preprint arXiv:2109.13112*.

Zhanming Jie, Jierui Li, and Wei Lu. 2022. Learning to reason deductively: Math word problem solving as complex relation extraction. *arXiv preprint arXiv:2203.10316*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Sheri Kingsdorf and Jennifer Krawec. 2016. A broad look at the literature on math word problem-solving interventions for third graders. *Cogent Education*, 3(1):1135770.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Rik Koncel-Kedziorski, Hannaneh Hajishirzi, Ashish Sabharwal, Oren Etzioni, and Siena Dumas Ang. 2015. Parsing algebraic word problems into equations. *Transactions of the Association for Computational Linguistics*, 3:585–597.

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 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*, pages 1152–1157.

Vivek Kumar, Rishabh Maheshwary, and Vikram Pudi. 2021. Adversarial examples for evaluating math word problem solvers. *arXiv preprint arXiv:2109.05925*.

Vivek Kumar, Rishabh Maheshwary, and Vikram Pudi. 2022. Practice makes a solver perfect: Data augmentation for math word problem solvers. *arXiv preprint arXiv:2205.00177*.

Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. 2014. Learning to automatically solve algebra word problems. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 271–281.

Yihuai Lan, Lei Wang, Qiyuan Zhang, Yunshi Lan, Bing Tian Dai, Yan Wang, Dongxiang Zhang, and Ee-Peng Lim. 2022. Mwptoolkit: An open-source framework for deep learning-based math word problem solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 13188–13190.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2021. Deduplicating training data makes language models better. *arXiv preprint arXiv:2107.06499*.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang Zhang. 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*, pages 6162–6167.

Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng Zhong. 2020. Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem. *arXiv preprint arXiv:2004.13781*.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022. On the advance of making language models better reasoners. *arXiv preprint arXiv:2206.02336*.

Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. 2021. Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems. *arXiv preprint arXiv:2110.08464*.

Chao-Chun Liang, Kuang-Yi Hsu, Chien-Tsung Huang, Chung-Min Li, Shen-Yu Miao, and Keh-Yih Su. 2016a. 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*, pages 67–71.

Chao-Chun Liang, Shih-Hong Tsai, Ting-Yun Chang, Yi-Chung Lin, and Keh-Yih Su. 2016b. A meaning-based English math word problem solver with understanding, reasoning and explanation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: System Demonstrations*, pages 151–155, Osaka, Japan. The COLING 2016 Organizing Committee.

Zhenwen Liang, Wenhao Yu, Tanmay Rajpurohit, Peter Clark, Xiangliang Zhang, and Ashwin Kaylan. 2023. Let gpt be a math tutor: Teaching math word problem solvers with customized exercise generation. *arXiv preprint arXiv:2305.14386*.

Zhenwen Liang, Jipeng Zhang, Jie Shao, and Xiangliang Zhang. 2021. Mwp-bert: A strong baseline for math word problems. *arXiv preprint arXiv:2107.13435*.

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

Xin Lin, Zhenya Huang, Hongke Zhao, Enhong Chen, Qi Liu, Hao Wang, and Shijin Wang. 2021. Hms: A hierarchical solver with dependency-enhanced understanding for math word problem. In *Thirty-Fifth AAAI Conference on Artificial 2021*, pages 4232–4240.

Qianying Liu, Wenyu Guan, Sujian Li, Fei Cheng, Daisuke Kawahara, and Sadao Kurohashi. 2021. Roda: reverse operation based data augmentation for solving math word problems. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:1–11.

Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke Kawahara. 2019a. 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)*, pages 2370–2379.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019b. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Nitin Madnani and Bonnie J Dorr. 2010. Generating phrasal and sentential paraphrases: A survey of data-driven methods. *Computational Linguistics*, 36(3):341–387.

Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. 2017. Paraphrasing revisited with neural machine translation. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, pages 881–893, Valencia, Spain. Association for Computational Linguistics.

Kathleen R McKeown. 1980. Paraphrasing using given and new information in a question-answer system. *Technical Reports (CIS)*, page 723.

Marie Meteer and Varda Shaked. 1988. Strategies for effective paraphrasing. In *Coling Budapest 1988 Volume 2: International Conference on Computational Linguistics*.

Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. 2021. A diverse corpus for evaluating and developing english math word problem solvers. *arXiv preprint arXiv:2106.15772*.

Pruthwik Mishra, Litton J Kurisinkel, Dipti Misra Sharma, and Vasudeva Varma. 2018. Equgener: A reasoning network for word problem solving by generating arithmetic equations. In *Proceedings of the 32nd Pacific Asia Conference on Language, Information and Computation*.

Arindam Mitra and Chitta Baral. 2016. Learning to use formulas to solve simple arithmetic problems. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2144–2153.

Arkil Patel, Satwik Bhattamishra, and Navin Goyal. 2021. Are nlp models really able to solve simple math word problems? *arXiv preprint arXiv:2103.07191*.

Jordan Peterson, Robert Pihl, Daniel Higgins, Jean Séguin, and Richard Tremblay. 2003. Neuropsychological performance, iq, personality, and grades in a longitudinal grade-school male sample. *Individual Differences Research*, 1:159–172.

Xinyu Pi, Qian Liu, Bei Chen, Morteza Ziyadi, Zeqi Lin, Yan Gao, Qiang Fu, Jian-Guang Lou, and Weizhu Chen. 2022. Reasoning like program executors. *arXiv preprint arXiv:2201.11473*.

Jean Piaget. 2013. *Child's Conception of Number: Selected Works vol 2*. Routledge.

Aaditya Prakash, Sadid A Hasan, Kathy Lee, Vivek Datla, Ashequl Qadir, Joey Liu, and Oladimeji Farri. 2016. Neural paraphrase generation with stacked residual lstm networks. *arXiv preprint arXiv:1610.03098*.

Subhro Roy and Dan Roth. 2016. Solving general arithmetic word problems. *arXiv preprint arXiv:1608.01413*.

Subhro Roy and Dan Roth. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.

Subhro Roy, Shyam Upadhyay, and Dan Roth. 2016. Equation parsing: Mapping sentences to grounded equations. *arXiv preprint arXiv:1609.08824*.

Subhro Roy, Tim Vieira, and Dan Roth. 2015. Reasoning about quantities in natural language. *Transactions of the Association for Computational Linguistics*, 3:1–13.

Timo Schick and Hinrich Schütze. 2021. Generating datasets with pretrained language models. *arXiv preprint arXiv:2104.07540*.

Zhihong Shao, Fei Huang, and Minlie Huang. 2022. Chaining simultaneous thoughts for numerical reasoning. *arXiv preprint arXiv:2211.16482*.

Jianhao Shen, Yichun Yin, Lin Li, Lifeng Shang, Xin Jiang, Ming Zhang, and Qun Liu. 2021. Generate & rank: A multi-task framework for math word problems. *arXiv preprint arXiv:2109.03034*.

Yibin Shen and Cheqing Jin. 2020. Solving math word problems with multi-encoders and multi-decoders. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 2924–2934.

Shuming Shi, Yuehui Wang, Chin-Yew Lin, Xiaojiang Liu, and Yong Rui. 2015. Automatically solving number word problems by semantic parsing and reasoning. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1132–1142.

Jinsong Su, Shan Wu, Deyi Xiong, Yaojie Lu, Xianpei Han, and Biao Zhang. 2018. Variational recurrent neural machine translation. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

Sowmya S Sundaram, Sairam Gurajada, Marco Fisichella, Savitha Sam Abraham, et al. 2022. Why are nlp models fumbling at elementary math? a survey of deep learning based word problem solvers. *arXiv preprint arXiv:2205.15683*.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27.

Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 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*, pages 297–306.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Lei Wang, Yan Wang, Deng Cai, Dongxiang Zhang, and Xiaojiang Liu. 2018a. Translating a math word problem to an expression tree. *arXiv preprint arXiv:1811.05632*.

Lei Wang, Dongxiang Zhang, Lianli Gao, Jingkuan Song, Long Guo, and Heng Tao Shen. 2018b. Mathdqn: Solving arithmetic word problems via deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Lei Wang, Dongxiang Zhang, Jipeng Zhang, Xing Xu, Lianli Gao, Bing Tian Dai, and Heng Tao Shen. 2019. Template-based math word problem solvers with recursive neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7144–7151.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Sam Witteveen and Martin Andrews. 2019. Paraphrasing with large language models. *arXiv preprint arXiv:1911.09661*.

Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuan-Jing Huang. 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), pages 7137–7146.

Zhipeng Xie and Shichao Sun. 2019. A goal-driven tree-structured neural model for math word problems. In *IJCAI*, pages 5299–5305.

Jing Xiong, Zhongwei Wan, Xiping Hu, Min Yang, and Chengming Li. 2022. Self-consistent reasoning for solving math word problems. *arXiv preprint arXiv:2210.15373*.

Jingfeng Yang, Hongye Jin, Ruixiang Tang, Xiaotian Han, Qizhang Feng, Haoming Jiang, Bing Yin, and Xia Hu. 2023. Harnessing the power of llms in practice: A survey on chatgpt and beyond. *arXiv preprint arXiv:2304.13712*.

Weijiang Yu, Yingpeng Wen, Fudan Zheng, and Nong Xiao. 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*, pages 3384–3394.

Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang Ronghuai. 2010. Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems. In *2010 Second International Workshop on Education Technology and Computer Science*, volume 2, pages 476–479. IEEE.

Seongjun Yun, Minbyul Jeong, Raehyun Kim, Jaewoo Kang, and Hyunwoo J Kim. 2019. Graph transformer networks. *Advances in neural information processing systems*, 32.

Dongxiang Zhang, Lei Wang, Luming Zhang, Bing Tian Dai, and Heng Tao Shen. 2019. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE transactions on pattern analysis and machine intelligence*, 42(9):2287–2305.

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.

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. Association for Computational Linguistics.

Lipu Zhou, Shuaixiang Dai, and Liwei Chen. 2015. Learn to solve algebra word problems using quadratic programming. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 817–822.

Yanyan Zou and Wei Lu. 2019. Text2math: End-to-end parsing text into math expressions. *arXiv preprint arXiv:1910.06571*.

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 $step\_size = 5$.

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.
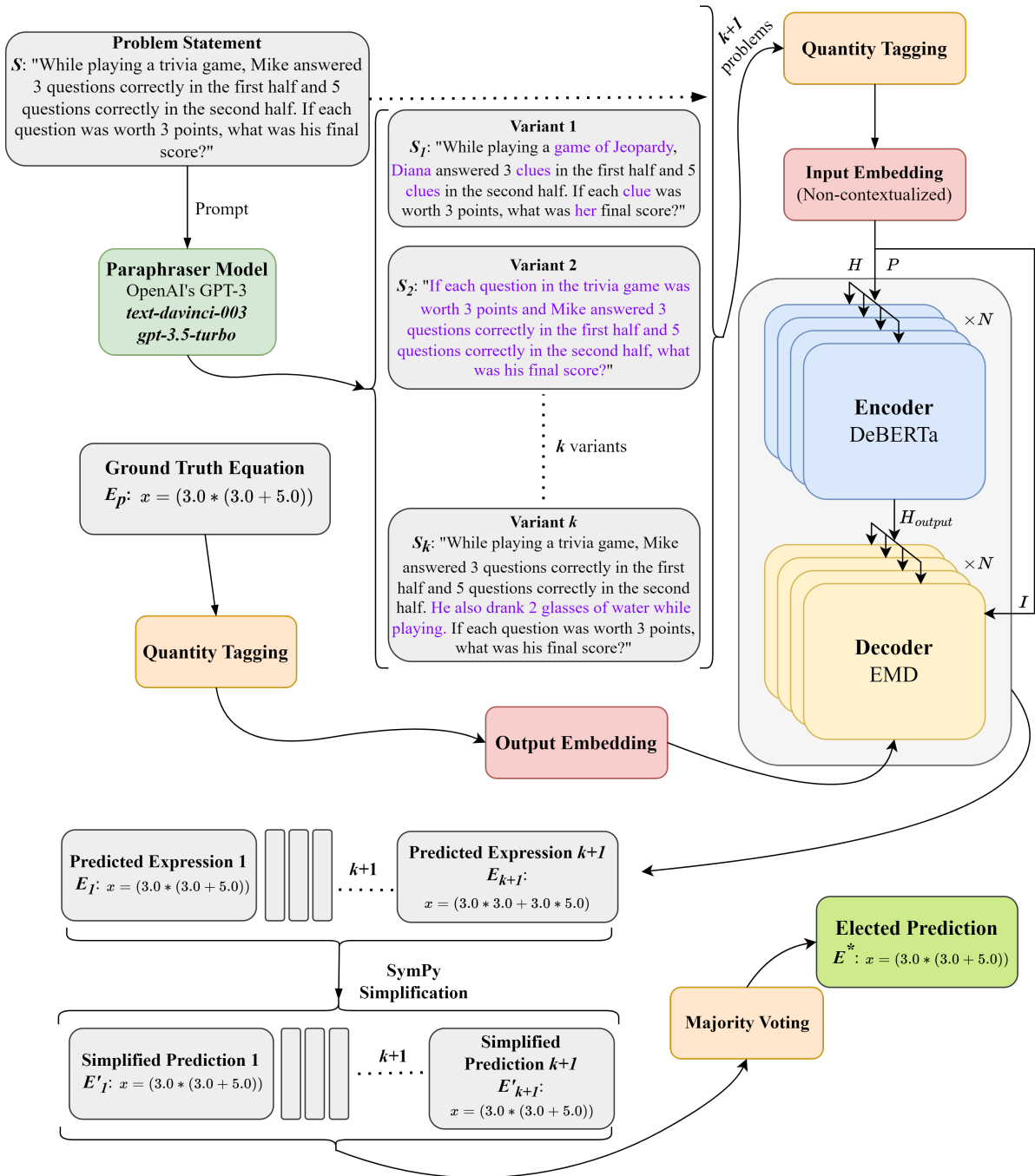
Figure 1: Overview of our proposed architecture.

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.
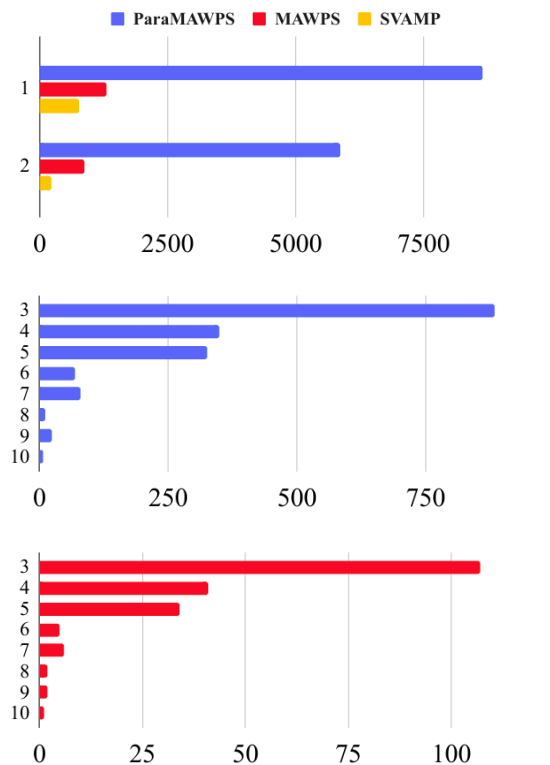

Figure 2: Operator count distributions of PARA-MAWPS, MAWPS, and SVAMP. We keep the distribution of PARAMAWPS somewhat similar to that of MAWPS to maintain a proper balance between easy and difficult problems.