ISLAMIC UNIVERSITY OF TECHNOLOGY (IUT)

Department of Computer Science and Engineering (CSE)

Variational Mathematical Reasoning: Enhancing Math Word Problem Solvers with Linguistic Variants and Disentangled Attention

Authors
Syed Rifat Raiyan — 180041205
Md. Nafis Faiyaz — 180041101
Shah Md. Jawad Kabir — 180041234

Supervised By

Md. Mohsinul Kabir
Assistant Professor

Dr. Hasan Mahmud
Associate Professor

Dr. Md. Kamrul Hasan
Professor

Systems and Software Lab (SSL)
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
A subsidiary organ of the Organization of Islamic Cooperation (OIC)

A thesis submitted to the Department of CSE
in partial fulfillment of the requirements for the degree of B.Sc. Engineering in Computer Science and Engineering (CSE)
Academic Year: 2021-2022
May 28, 2023

Declaration of Authorship

This is to certify that the work presented in this thesis titled **"Variational Mathematical Reasoning: Enhancing Math Word Problem Solvers with Linguistic Variants and Disentangled Attention"** is the outcome of the analysis and experiments carried out by Syed Rifat Raiyan, Md. Nafis Faiyaz, and Shah Md. Jawad Kabir under the supervision of Md. Mohsinul Kabir, Assistant Professor, Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Dhaka, Bangladesh. It is also declared that neither this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

Syed Rifat Raiyan
Student ID - **180041205**

Md. Nafis Faiyaz
Student ID - **180041101**

Shah Md. Jawad Kabir
Student ID - **180041234**

Supervisors:

Md. Mohsinul Kabir
Assistant Professor
Systems and Software Lab (SSL)
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)

Dr. Hasan Mahmud
Associate Professor
Systems and Software Lab (SSL)
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)

Dr. Md. Kamrul Hasan
Professor
Systems and Software Lab (SSL)
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)

Acknowledgements

We would like to express our gratitude to **Md. Mohsinul Kabir** for being our adviser and mentor. His profound knowledge, astute insights, and guidance have played a pivotal role in shaping the trajectory of our research. His intellectual acumen and attention to detail have constantly pushed us to delve deeper into the subject matter, ensuring the thoroughness and rigor of our work. We are grateful for his patience, as he meticulously reviewed our drafts, providing constructive criticism and invaluable suggestions that have greatly enriched the quality of our thesis. His mentorship has been transformative, instilling in us a passion for scholarly inquiry and an unwavering pursuit of academic excellence. Without his support and proper commitment, this research would never have been possible. We are forevermore grateful to him.

We would also like to extend our heartfelt gratitude to **Dr. Hasan Mahmud** and **Dr. Md. Kamrul Hasan** for their unwavering support which was instrumental in shaping the outcome of our research.

We convey our gratefulness to Allah Subhanahu Wa ta'ala and our parents for everything.

This work is dedicated to our parents.

Abstract

The art of mathematical reasoning stands as one of the most fundamental pillars of intellectual and scientific advancement, being a central catalyst in the cultivation of human ingenuity. Researchers have recently published a plethora of research works centered around the task of solving Math Word Problems (MWP). 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 at — https://github.com/Starscream-11813/Variational-Mathematical-Reasoning

Keywords — math word problem, natural language processing, paraphrasing, challenge dataset, DeBERTa, GPT-3, mathematical reasoning, linguistic variant

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Math word problem solving is one of the long-standing research problems in general artificial intelligence and a lot of research papers about them, from both industry and academia, have been published recently. A typical Arithmetic problem or Math Word Problem (MWP) is a textual narrative that states a problem description and poses a question about one or more unknown quantities. An NLP model capable of solving such problems has to translate the human-readable problem statement to a valid mathematical expression which can be evaluated to obtain the numeric answer. An example of a classic MWP is portrayed in Table 1.1, where the reader is asked to infer the revenue of a boutique shop.

Table 1.1: 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

Such problems are generally found in math textbooks of 1^{st} to 8^{th} grade students and are easily solvable by humans with decent mathematical aptitude. However, a lot of challenges manifest while designing an automated system for performing these tasks. The challenges that need to be addressed in the case of MWPs are:

1. Understanding the quantities in the problem and capturing their complex mathematical interconnections from a linear textual sequence, which can sometimes be ambiguous.

2. Diverse range of MWPs with differing difficulty levels, *i.e.*, varying number of unknown values and depth of the relationships between quantities.

3. Absence of crucial information and presence of irrelevant information in the problem statements [2].

4. Chronological and temporal ambiguities of the events happening in the problem statements.

5. MWPs that significantly differ from the training set in terms of semantic and syntactic structure.

In order to solve the problem described in Table 1.1, an ideal MWP solver model must be able to associate the quantity, *i.e.*, 69 handbags, with its price attribute of \$13, and understand 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 [3–5]. The frontier of research involving MWP solving is considered a crucial step towards general AI 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 [2] creating a faux impression of their mathematical reasoning capability.

1.2 Research Challenges

The prominent research challenges that are present in the realm of Math Word Problem solving are,

- The pre-trained language model-based approaches discussed in the Deep Learning subsection of the Literature Review section suffer from spurious correlations between input problem statement and output math expression.
- The existing models depend on shallow heuristics to yield high degree of accuracy in benchmark datasets, thus masquerading under the guise of faux mathematical reasoning. They can even infer solutions to problem statements just by looking at the superficial patterns present even if it is lacking word-order information or lacking the actual question text [2].
- Absence of large-scale English datasets with adversarial, challenging and inverse versions of MWP samples that require multiple deductive steps and operations to solve and have a good number of unique math expression templates.

1.3 Thesis Contributions

To account for the aforementioned limitations, we present our thesis work that yields the following contributions —

- 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) [6] 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 [1] 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) [7] 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. It is significantly more efficient in language translation tasks compared to architectures with recurrent or convolutional layers.

1.4 Motivation

Although there is a debate in the scientific community on whether or not we can model our consciousness using Artificial Intelligence due to Gödel's Incompleteness Theorem [8], computer scientists are very confident in being able to model the mathematical aptitude of humans in machines. The frontier of research involving Math Word Problem Solving is, therefore, a crucial step toward general AI. Deep Learning models capable of solving such problems can be of great utility in smart desktop and mobile applications for teaching students math. This was the principal talking point in the most recent Neural Information Processing Systems (NeurIPS) conference titled *"Math AI for Education (MATHAI4ED): Bridging the Gap Between Research and Smart Education."* One of the most popular challenges in the AI community at present is the *International Mathematical Olympiad (IMO) Grand Challenge*[9] where the challenge is to create AI models capable of solving Math Olympiad problems. The collective inspiration that we got from being avid problem-solvers ourselves as well as from the IMO Grand Challenge, incentivized us to pursue learning about NLP models for solving mathematical problems. So we are optimistic about the future of this domain of research and hope to try our best to work and contribute to this domain.

1.5 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.

1.6 Thesis Layout

The thesis is thoughtfully structured, adhering to a well-crafted layout that encompasses the following 6 chapters — In Chapter-1, we set the context for the reader by introducing the research problem and delineating the challenges encountered in this domain of scholarly inquiry. We formulate the research problem in precise terms, pristinely list down the contributions of this thesis work, and write about the circumstances that inspired us to pursue the topic in this chapter. This thesis layout section is a constituent of the chapter that aids in the readability of the thesis book by recapitulating a cohesive narrative of the work. In Chapter-2, we critically examine the existing scholarly works, theories, and empirical studies related to the research topic, establishing a strong theoretical foundation for the study. We provide a proper taxonomy and chronological development of the models used in the existing literature in this chapter and perform comparative analyses among them. In Chapter-3, we outline our proposed methodology with the aid of intuitive diagrams and explain the underlying mechanisms of the components that are present within the architecture. Chapter-4 elaborates on the experimental setup or procedures undertaken to collect empirical data. We provide a detailed statistical analysis of our dataset and delineate the criteria that we have taken into account whilst constructing the dataset. We also mention the models that we use as baselines and the intricate details pertaining to the training process in this chapter. Chapter-5 presents an in-depth analysis and interpretation of the obtained results along with a critical discussion of their implications and relevance. We also bring to light some of the weaknesses of the existing models and provide a comprehensive ablation study to convey a deeper understanding of the pipeline's working process to the reader in this chapter. The epilogue of our thesis work, Chapter-6, provides a concise summary of our research findings and insights into potential future directions for further exploration towards the apotheosis of this research domain.

Chapter 2

Literature Review

2.1 Math Word Problem Solving

The dawn of research on MWP solving was in the mid-1960s [10, 11]. Since then, numerous research papers attempting to perform the MWP task by adopting a multitude of approaches have been published. We categorize these existing MWP solvers into seven categories: Rule-based, Statistical, Tree-based, Semantic Parsing-based, Similarity-based, Template-based, and Deep Learning (DL)-based methods. *Rule-based methods* [12–14] are chronologically some of the earliest approaches to solving MWPs. These methods use a set of manually hard-coded rules about the language they are analyzing to find out regularities in the data. *Statistical methods* [15–21] 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* [22–25] utilize the inherent binary tree-like structure of expressions/equations and focus on constructing the structure of the resultant expression/equation's equivalent tree in a bottom-up manner. Other primitive categories of approaches that have now been rendered somewhat obsolete are *Parsing-based methods* [26, 27], *Similarity-based methods* [28], and *Template-based methods* [15, 18, 24, 29, 30].

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. [31] 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 [32, 33], using dense problem representation [34], adopting template-based methodologies [35], and incorporating group attention mechanisms [36]. Xie and Sun [37] 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 [38], BART [39], RoBERTa [40], as the encoder in some of the Seq2Tree approaches [41–49] brought about substantial performance improvements over the previous Seq2Seq methods. Cao et al. [50] devised a directed acyclic graph (Seq2DAG) model of the equations for the purpose of extracting the expression. Zhang et al. [51] incorporated the idea of Knowledge Distillation (KD) [52] in their proposed model where the *teacher network* is pre-trained to guide the learning behaviors of the *student networks*. Yu et al. [53] introduced 2 types of encoders in their model, which are *Pre-trained Knowledge* encoder and *Hierarchical Reasoning* encoder. Hong et al. [54] modified the work of [37] by incorporating a symbolic reasoning based *Learning-by-fixing* (Lbf) framework. Qin et al. [55] proposed a model that performs 4 auxiliary tasks, Number Prediction, Commonsense Constant Prediction, Program Consistency Checking, and Duality Exploitation, to integrate different levels of symbolic constraints. Huang et al. [56] attempted to emulate human-like analogical learning in their proposed memory-augmented model. Graph2Tree-based approaches [57, 58] fused the merits of Graph-based Transformer [59, 60] encoders with multiple Graph Convolutional Network (multi-GCN) modules [61], and tree-based decoders to solve MWPs. Chatterjee et al. [62] introduced a weakly supervised approach for MWP solving. [48] introduced a contrastive learning approach with pattern divergence to solve MWPs. Jie et al.

[63] formulated the MWP solving task as a complex relation extraction problem and leverages explainable deductive reasoning techniques to iteratively construct the desired target math expressions.

With the advent of LLMs, many innovative prompt-based methods [64–68] 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. [69] used verifiers with their GPT-3 [6] model to check the correctness of model outputs. Although LLMs excel at natural language understanding and have serendipitous emergent reasoning abilities [70], they are still lackluster in complex reasoning tasks [71]. Numerous studies on complex reasoning tasks have empirically demonstrated that the approach of fine-tuning smaller models is more effective [72] than adopting LLM prompting techniques like Chain of Thought (CoT) prompting [73]. Accordingly, our work attempts to leverage the strengths of GPT-3 to generate a more linguistically 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).

2.1.1 Rule-based Methods

Rule-based methods are chronologically some of the earliest approaches to solving MWPs. Fletcher [12] proposed a computer program WORDPRO coded in Interlisp-D, which could solve one-step arithmetic problems with 4 types of predefined schemata after translating the problem statement to a set of propositions. Later in 2007, Bakman [13] proposed a system named ROBUST, which could conceptualize free-format multi-step MWPs with extraneous information by building upon the work of [12] and introducing 6 types of predefined schemata. In 2010, [14] proposed MSWPAS which could solve multi-step addition and subtraction problems by converting the problem statements into Problem Frames consisting of the whole semantic information of the problems. The principal drawbacks of these methods is high dependency on manual features and inability to generate novel templates

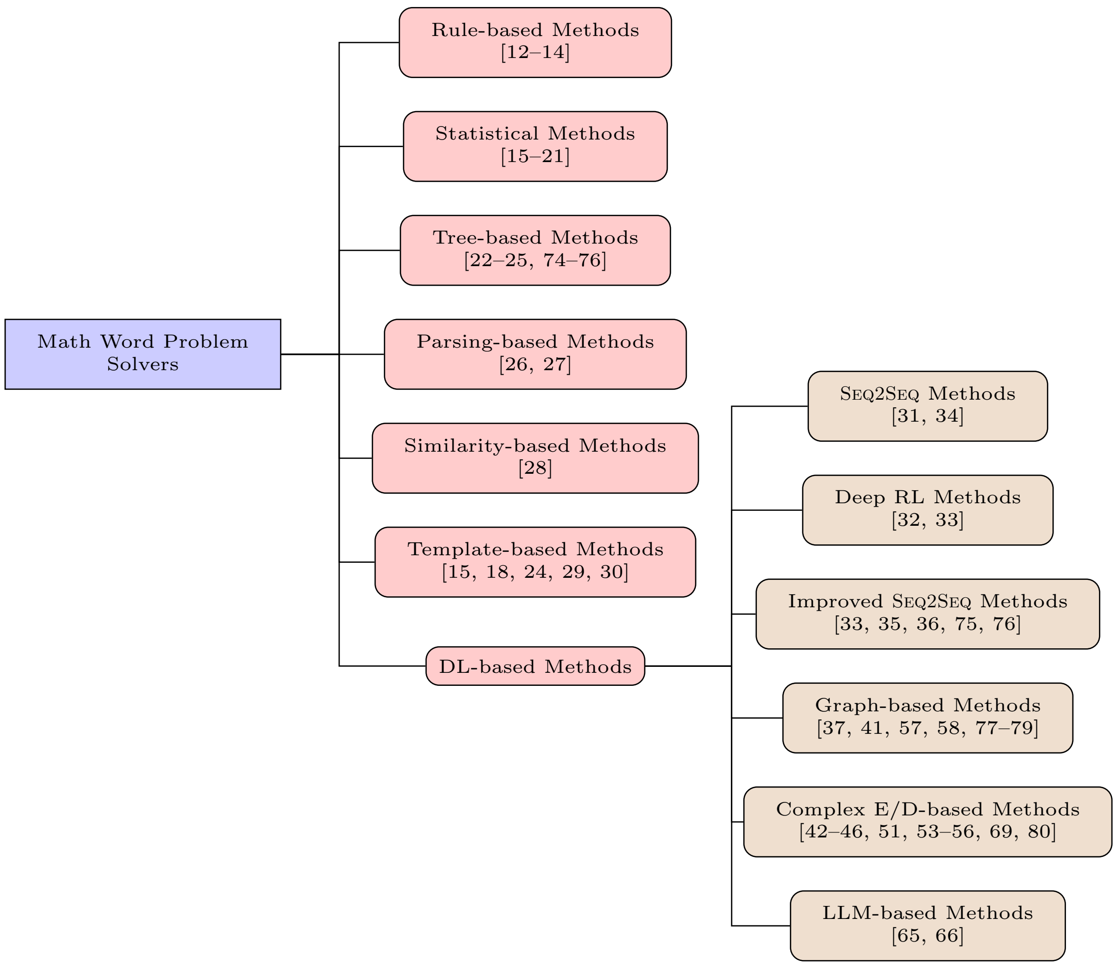

Math Word Problem Solvers
Rule-based Methods [12–14]
Statistical Methods [15–21]
Tree-based Methods [22–25, 74–76]
Parsing-based Methods [26, 27]
Similarity-based Methods [28]
Template-based Methods [15, 18, 24, 29, 30]
DL-based Methods
Seq2Seq Methods [31, 34]
Deep RL Methods [32, 33]
Improved Seq2Seq Methods [33, 35, 36, 75, 76]
Graph-based Methods [37, 41, 57, 58, 77–79]
Complex E/D-based Methods [42–46, 51, 53–56, 69, 80]
LLM-based Methods [65, 66]

Figure 2.1: Classification of MWP Solvers.

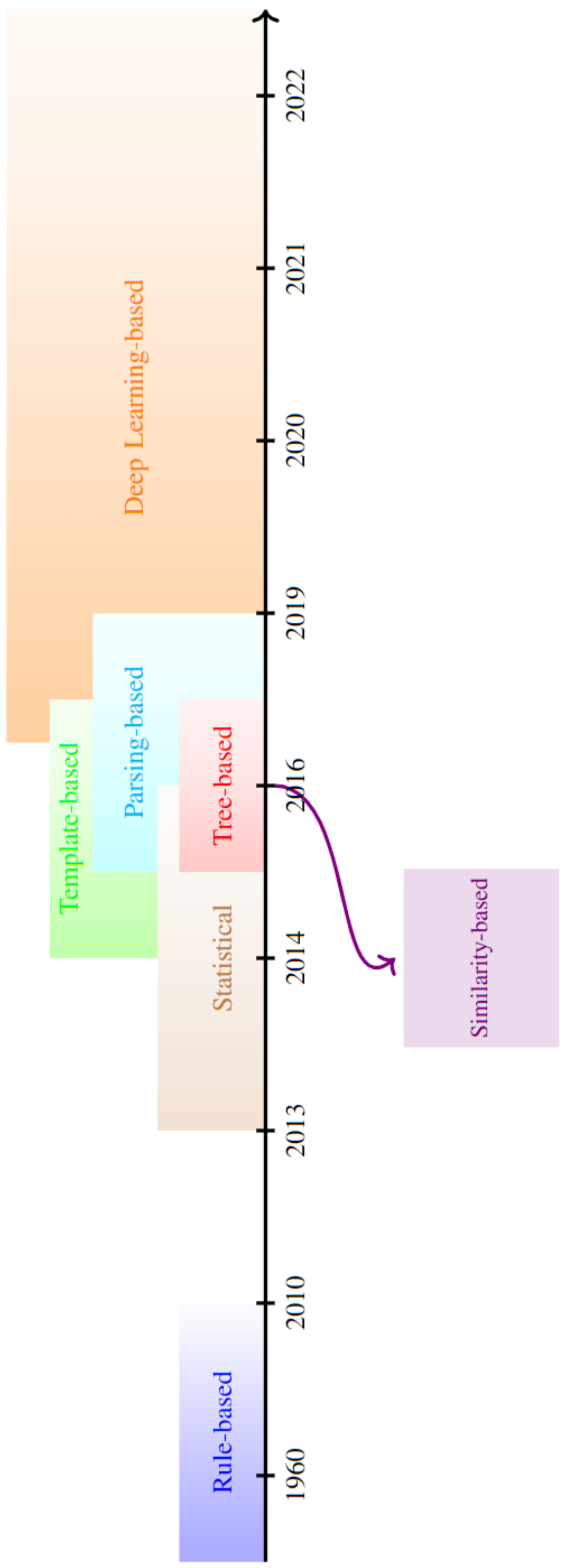


Figure 2.2: Evolutionary Trend of MWP Solvers.

for new problems. This paper provides a brief overview of these outdated methods but interested readers can check out [81] for detailed descriptions.

2.1.2 Statistical Methods

The Statistical methods use generic ML models to extract the entities, quantities and operators from the problem statement and infer the numeric answer with simple logic. Kushman et al. [15] proposed an algorithm that could reason across sentence boundaries and define a joint log-linear distribution over systems of equations including the interrelations between those equations and the problem text. They adopted a two-step process to map the word problems to equations, namely, selecting a template that defines the overall alignment of the equation set and instantiating that template with numerical values and entities from the problem statement. Using a newly procured corpus of 514 math word problems taken from *www.algebra.com*, they used supervised and semi-supervised learning methods to derive the equations. Although [15] is one of the earliest and pioneer works incorporating Semantic Interpretation and Information Extraction in MWP solving, it falters in cases of problems with new compositional language due to lack of sufficient background or world knowledge. Roy et al. [17] proposed a Quantity Entailment scheme which could solve word problems with a single operator. They used a cascade of 3 classifiers in their approach. The *Quantity Pair Classifier* outputs the pair of quantities required to obtain the answer, the *Operation Classifier* outputs which of the 4 fundamental mathematical operations $O = \{+, -, \times, \div\}$ is required and the *Order Classifier* (relevant for $-$ and $\div$) decides the most likely order of quantities in the operation. The obvious limitation of this QE scheme is the ability to produce only single operator expressions while incurring a huge computational overhead. Hosseini et al. [16] proposed a system ARIS, which was an early attempt at introducing more advanced logic templates to statistical methods of solving multi-step problems. ARIS could solve only addition (+) and subtraction ($-$) type MWPs by representing the problem text as a world state tuple $\langle E, C, R \rangle$, where E is the set of Entities (objects and attributes), C is the set of Containers (owners of the entities) and R is the set of Relations among the containers, entities, attributes and quantities. They were the first to introduce a 7-category *Verb Categorization* and synsets for predicting verb categories in sentences using syntactic and semantic features. They also introduced an addition-subtraction based dataset, named AI^3, consisting of 395 problems. A lot of manual effort is required to annotate each split sub-text in the training data with one of the 7 verb categories while training a classifier like ARIS. Another obvious drawback of ARIS is its inability to work with mathematical operators other than $+$ and $-$. Zhou et al. [18] proposed an algorithm that considers all possible equation systems in the hypothesis space and obtains a robust decision hyperplane using Maximum Margin Classifiers or Support Vector Machines (SVMs). They compared their model with the then state-of-the-art baseline [15] and stated that the latter's beam search fails to utilize all the training samples, making it a sub-optimal approach. In order to solve this constrained optimization problem of maximizing the margin between correct and incorrect assignments [18] used Quadratic Programming (QP). Although their accuracy trumped that of [15], the algorithm was still unable to resolve MWPs with complex noun phrases and lexical features. Mitra and Baral [19] proposed a novel system of learning to apply formulae on the higher level representation of MWPs to reach solutions. They categorized the formulae into 3 categories, *part whole*, *change* and *comparison* which are sufficient to deal with the MWPs in the ADDSUB data set, a corpus of standard primary school test arithmetic problems. The system learns to generate a scored $\langle formula, variables \rangle$ pair from the higher level representation of the problem statement during the training phase. Just like the previously discussed statistical methods, the obvious drawback of this system is that it cannot work with mathematical operators other than $+$ and $-$.

The ideas proposed by Liang et al. [20, 21] are somewhat similar ideas of annotation to solve MWPs. The core idea is to analyze the problem statement and transform both the problem scenario and the question at the end of the problem statement into their tag-based logic forms and eventually infer the solution. The main advantage of the approach taken by such MWP solvers is that they are less sensitive to irrelevant entities and quantities in the problem statement. Suppose, the problem statement contains the sentence *"Jordan bought 69 books."*, then the logic form mapping would look something like, $\mathrm{verb(v_1, buy)}$ & $\mathrm{nsubj(v_1, Jordan)}$ & $\mathrm{dobj(v_1, n_1)}$ & $\mathrm{head(n_1, book)}$ & $\mathrm{nummod(n_1, 69)}$.

2.1.3 Tree-based Methods

Algebraic and arithmetic expressions/equations have an inherent binary tree-like structure. Such binary trees are defined as *expression trees* and *equation trees* respectively.

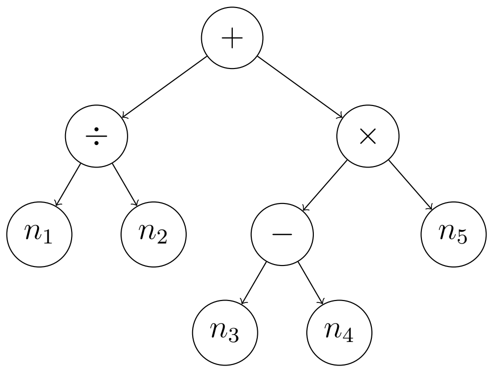


Figure 2.3: Expression Tree representing $n_1 \div n_2 + (n_3 - n_4) \times n_5$.

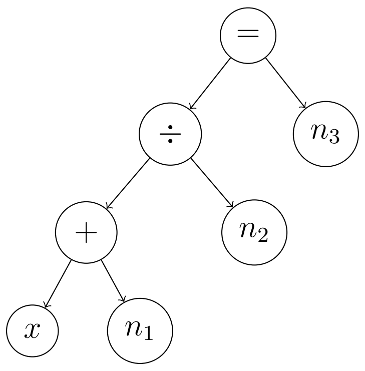


Figure 2.4: Equation Tree representing $(x + n_1) \div n_2 = n_3$.

The terminal nodes which are also known as *leaf nodes*, represent the constants or the variables in the expression/equation, whereas, the internal *non-leaf nodes* represent the mathematical operators. The operators' precedence/priority increase the lower/deeper they are in the tree. In expression trees, the root node represents the operator with the least priority and in equation trees, the root node represents the equal (=) sign and one or more of the leaf nodes represent unknown variable(s).

Tree-based methods of solving MWPs focus on constructing the structure of the resultant expression/equation's equivalent tree in a bottom-up manner. Roy and Roth [23] were the pioneers in using the concept of expression trees in MWP solvers. The first step of their proposed algorithm is to use a binary classifier to decide if an extracted quantity is relevant or not and thereby lessen the search space. The relevant quantities are later represented by the leaf nodes of the expression tree and the irrelevant quantities are discarded. The tree composition procedure is an aggregation of simple prediction problems, where the goal is to determine the Lowest Common Ancestor (LCA) of pair of quantities mentioned in the problem statement. The score of the expression E represented by the monotonic expression tree T is calculated as,

$$\begin{aligned}\mathbf{Score}(E) =& w_{\textsc{Irr}} \sum_{q \in I(E)} \textsc{Irr}(q) + \\ & \sum_{q_i, q_j \notin I(E)} \textsc{Pair}(q_i, q_j, \odot_{\textsc{LCA}}(q_i, q_j, T))\end{aligned} \tag{2.1}$$

where, $w_{\textsc{Irr}}$ is a scaling parameter, $\textsc{Irr}(q)$ is the likelihood of quantity q being an irrelevant quantity, $\textsc{Pair}(q_i, q_j, op)$ is the likelihood score given by a multi-class classifier for predictions of LCA operations and $I(E)$ is the set of all quantities that can be extracted from the problem statement but are not relevant in inferring the final solution. The final inference problem then becomes,

$$\underset{E \in C}{\operatorname{argmax}}\, \mathbf{Score}(E) \tag{2.2}$$

where C is the set of all valid expressions. [23] also introduced another concept called *quantity schema* which parses out the information relevant to each quantity mentioned in the problem statement and discards unnecessary portions of the text.

For evaluating their proposed system, they experimented on 3 datasets, which are, the AI2 dataset, the IL dataset and the COMMONCORE dataset. Interested readers can try out an online version of this system which was implemented by Roy and Roth [74]. Koncel-Kedziorski et al. [22] introduced a new system, ALGES, capable of generating equation trees from multi-sentence MWPs. ALGES adopts a more brute-force approach to take into consideration all the possible equation trees, generated using Integer Linear Programming (ILP), and scores them in terms of likelihood by learning local and global discriminative models.

As evident in the comparative analysis done by Wang et al. [32], ALGES possesses a significantly large computational overhead compared to [23]. ALGES uses a compact representation of each node referred to as a *Quantified Set* or *Qset* for modelling the correspondence between the quantities and their properties. The "best" equation tree t^* is computed as,

$$t^* \leftarrow \underset{t_i \in T}{\operatorname{argmax}} \left(\prod_{t_j \in t} L_{local}(t_j|w) \right) \times G_{global}(t|w) \tag{2.3}$$

scoring each tree $t_i \in T$, where $L_{local}(t_j|w)$ is the likelihood score of subtree t_j, forming pairwise Qset relationships, $G_{global}(t|w)$ is the likelihood score of the root node of tree t, representing the whole equation.

[24] proposed an efficient algorithm to parse the problem text to projective equations. It assumes the final output equation to consist of at most 2 variables and utilizes each quantity extracted from the problem text at most once in the final equation. These assumptions were made to simplify the tree composition process at the cost of sacrificing the extent of applicability. The relevant quantities and variables are determined using structural SVM classifiers with superset supervision and customized feature selection. The tree is constructed in a bottom-up manner by considering the quantities and variables as leaf nodes and combining the adjacent child nodes to form their parent node as an attempt to lessen the search space.

Roy and Roth [23] built upon their work in the following year and proposed the integration of *Unit Dependency Graphs (UDG)* for improving the scoring function[25]. They found UDGs to be very helpful in capturing the relationships and dependencies between the units of quantities mentioned in the problem statement thus alleviating the brittleness of extracting units.

A total of 6 types of edge associations are taken into consideration while determining whether two quantities are related with the same unit. The construction of UDGs introduce an extra computational overhead because of training binary SVMs and multi-class SVMs for nodes and edges respectively. The likelihood score of a Unit Dependency Graph G is computed as,

$$\begin{aligned}\textbf{Score}(G) = & \sum_{\substack{v\in V\\ \textsc{Label}(G,v)=\textsc{Rate}}} \textsc{Vertex}(v, \textsc{Rate}) + \\ & \lambda \times \sum_{v_i,v_j\in V, i<j} \textsc{Edge}(v_i, v_j, \textsc{Label}(G, v_i, v_j))\end{aligned} \tag{2.4}$$

where λ is a scaling parameter, $\textsc{Label}(F, v_i, v_j)$ determines if a node v is a RATE or NOT RATE. After finding the set of all possible Unit Dependency Graphs GRAPHS, the final inference problem becomes,

$$\operatorname*{argmax}_{G\in\textsc{Graphs}} \textbf{Score}(G) \tag{2.5}$$

Their proposed algorithm then mimics their previous algorithm[23] finds out the expression tree with the highest likelihood score following (2.1) and (2.2).

Wang et al. [75] and Chiang and Chen [76] used implicit tree structures and SEQ2SEQ models to improve expression tree generation. [75] took into consideration the uniqueness of expression trees and introduced an equation normalization method combined with an ensemble model comprising of a Bidirectional Long Short Term Memory (BiLSTM)[82], a CONVS2S Convolutional SEQ2SEQ model[83] and a Transformer model[84] and named it MATHEN. This equation normalization process proves to be helpful in significantly increasing the performances of the models.

The common advantage of the tree-based models discussed in this section is that additional manual annotations *e.g.* template, tags, or logic forms are not required. Researchers further worked upon this tactic of forming expression/equation trees or Abstract Syntax Trees (AST) and these works are discussed in the Deep Learning (DL) based methods section.

2.1.4 Parsing-based Methods

Shi et al. [26] were the first to build upon the tree-based approaches to solve a set of multiple equations by presenting a *semantic parsing* and reasoning approach based on a newly designed *meaning representation language* referred to as Dolphin Language (DOL). Their proposed system, namely SigmaDolphin, transforms the MWP text into DOL trees. The *reasoning* module of SigmaDolphin then derives the final expression from the constructed DOL tree representation.

The parsing algorithm implemented in SigmaDolphin is based on a mathematical system of modeling constituent structure in Natural Language (NL), known as Context-free Grammar (CFG)[85, 86]. This CFG parser is imbued with a total of 9600 grammar rules and the associations between the math concepts and these grammar rules were done manually in a semi-supervised manner. Every node of the DOL tree is scored during the parsing phase and the DOL tree with the maximum aggregate score is passed to the reasoning module of SigmaDolphin to obtain the final answer. The score of a tree T is the weighted mean of the scores of its sub-trees,

$$\mathbf{Score}(T) = \frac{\sum_{i=1}^{k} L(T_i) \cdot \mathbf{Score}(T_i)}{\sum_{i=1}^{k} L(T_i)} \cdot p(T) \tag{2.6}$$

where, T_i is a sub-tree, $L(T_i)$ is the number of total words to which T_i corresponds in the original problem statement and $p(T)$ is the type-compatibility property of the tree T. Shi et al. [26] also prepared a dataset consisting of 1878 math problems collected from two websites—algebra.com and answers.yahoo.com.

Zou and Lu [27] proposed a model named Text2Math which can semantically parse the MWP text to math expressions using end-to-end latent variable predictions without any a priori knowledge of the operators. Text2Math leverages a novel joint representation to automatically learn the association between the words in the problem text and math expressions which possesses semantic "closeness".

The common weakness of these parsing-based methods is that their applicability is limited to Number Word Problems. They show incompetence in grasping the complex relationships between quantities and entities seen on general MWPs.

2.1.4.1 Similarity-based Methods

Huang et al. [28] introduced a simple statistical method of MWP solving, called Sim, which works by calculating the similarity between a test set sample with samples in the training set. The equation system of the most similar training sample (*template selection*) is applied to the test sample problem under consideration (*template slot filling*). In the *template selection* step, each MWP is represented using a vector of word Term Frequency - Inverse Document Frequency (TF-IDF) scores and the similarity between two MWPs is computed by the weighted Jaccard Similarity Coefficient of their representative vectors. Huang et al. [28] also created a large-scale dataset called Dolphin18K which consists of 18,460 annotated MWPs collected from online fora and web pages. The drawback of such similarity-based methods is that they fail to work out problems that do not necessarily follow the same structural template as the problems in the training set.

2.1.5 Template-based Methods

The core idea behind template-based approaches is to identify a candidate template from a pre-defined corpus of equation templates and plug in the numeric and variable slots with quantities extracted from the problem statement. Some of the research works [15, 18, 24] discussed in the Statistical Methods section adopt this template-based idea. As the search space is exponential to the number of slots due to the fact that each quantity in the problem statement is a potential candidate for the numerical slots and each name/entity is a potential candidate for the variable slots, these researchers used Beam Search inference procedure to find the optimal template. The suitable candidates for each slot in the template are selected based on an a priori canonicalized ordering where the top-k partial derivations are considered. Upadhyay et al. [29] proposed an algorithm called MixedSP, which stands for Mixed Supervision, *i.e.*, it learns from both explicit supervision (*e.g.* equations) and implicit supervision (*e.g.* solutions) based in Structured-output Perceptron[87]. MixedSP takes as input both kinds of training signals and iteratively improves the model while using the intermediary model to figure out candidate equation sets for problems. Upadhyay et al. [29] also created a new dataset, named Sol2k, without any annotated equation sets containing only the problem text and the final numeric solution. The mapping strategy from the MWP to the equation template of MixedSP resembles the strategy proposed in [15] and [18]. One obvious drawback of such template-based methods is that they fail to work well in sparse training samples. Another weakness is that the learning process of the models used in such methods heavily relies on lexical and syntactic features with huge and sparse feature space, *e.g.*, the dependency edge/path between two slots of a template.

Huang et al. [30] integrated a finer granularity in the learning process to overcome this sparseness drawback. They proposed a novel approach to capture rich information from templates by parsing them into tree structures. In a constructed tree, they defined a *template fragment* as any subtree with at least a single operator and 2 operands. The mapping procedure was "fine grained" based on Longest Common Substring (LCS). Suppose, an excerpt from the problem statement is something like *"69% discount"*. This portion of the problem text can be mapped to a template fragment $1 - v_1$ where $v_1 = 0.69$. The template extraction process of this proposed method, namely FGExpression, is a semi-supervised process done using a RankSVM model[88] which selects the top-k templates and reduces the search space by a lot as a consequence.

2.1.6 Deep Learning-based Methods

The excellence of Deep Learning for which it is practised with such gravitas in the realm of Computer Science is due to its ability to learn effective feature representations without the need for any human intercession in a data-driven manner. The general pipeline of DL-based MWP solving approaches is portrayed in Figure-2.5.

2.1.6.1 Seq2Seq Methods

Wang et al. [31] were the pioneers of introducing deep learning to solve Math Word Problems (MWPs). Without resorting to any feature engineering, they used a Recurrent Neural Network (RNN) model combined with a similarity-based retrieval model in their proposed SEQ2SEQ model. They also created the first large-scale MWP dataset suitable for training Deep Neural Networks, MATH23K, which consists of 23,161 problems labeled with their corresponding equations and answers. The RNN-based SEQ2SEQ model is primarily responsible for transforming the problem statement into a math equation. The encoder uses Gated Recurrent Units (GRUs) [89] and the decoder uses Long Short-term Memory (LSTM) cells [90]. GRU works better as an encoder than LSTM because it has a lesser propensity to overfit the dataset. The SEQ2SEQ model is 5 layers deep, having 1 word embedding layer, a 2-layer GRU encoder, and a 2-layer LSTM as the decoder. The encoder and decoder both consist of a total of 512 nodes. In order to identify significant and insignificant numbers in the problem text, [31] also proposed a Significant Number Identification (SNI) model which is basically an LSTM-based binary classification model. It consists of 128 nodes and a length 3 symmetric window. The retrieval model computes the lexical similarity using Jaccard's Similarity Coefficient between the test sample and each problem in the training data. The equation is,

$$J(P_T, Q) = \frac{|P_T \cap Q|}{|P_T \cup Q|} = \frac{|P_T \cap Q|}{|P_T| + |Q| - |P_T \cap Q|} \tag{2.7}$$

where, P_T is the test problem and Q is a problem from the training data. The equation template with the highest similarity score is applied to that test sample.

Mishra et al. [34] introduced a novel method of dense representation of MWPs and they used that representation to generate appropriately ordered operands and operators. Their model, namely EquGener, is composed of an end-to-end memory network encoder and an equation decoder.EquGener is equipped to handle all 4 kinds of fundamental mathematical operations, $O = \{+, -, \times, \div\}$. The input sequence is a sequence of word vectors which are a concatenation of vector representations of pre-trained GloVe embeddings[91] and embeddings obtained by the network from the training corpus. The attention-based encoder and decoder used in the model possess Long Short-term Memory (LSTM)[90] cells.

2.1.6.2 Deep Reinforcement Learning (RL) Methods

Wang et al. [32] were the first to apply Deep Reinforcement Learning to solve MWPs. They proposed a Deep Q-network model named MathDQN with newly designed states, actions, reward function, and a 2-layered Feed-Forward Neural Network. In Reinforcement Learning (RL), given a set of internal states $S = \{s_1, \ldots, s_m\}$ and a set of actions $A = \{a_1, \ldots, a_n\}$, an agent takes action a at state s and transitions to a new state s' over multiple iterations till a termination condition holds true. This learning procedure is governed by policies or rules π extracted from the environment E. The true value of an action a in state s is computed as,

$$Q_\pi(s, a) = \mathbb{E}[R_1 + \gamma R_2 + \ldots | S_0 = s, A_0 = a, \pi] \tag{2.8}$$

where $\gamma \in [0, 1]$ is discount factor for future rewards. The action which yields a positive Q-value will be rewarded. MathDQN's first step is to extract, re-order and sort the relevant quantities. It uses the sorted quantities as the bottom level and the partial tree constructed up to this step is taken as a state. A positive Q-value that is worth rewarding is yielded if the next state or partial tree bears a closer resemblance to the final *ground-truth* tree otherwise a negative Q-value is yielded and is returned as punishment. The loss function that the model tries to minimize is,

$$\begin{aligned} L_t(\theta_t) &= \mathbb{E}_{s,a}[(y_t - Q(s,a;\theta_t))^2]; \\ y_t &= r + \gamma \max_{a'} Q(s',a';\theta_{t-1}) \end{aligned} \tag{2.9}$$

where y_t is the optimal Q-value, r is current reward and θ is the set of parameters. These parameters are learnt using the Gradient Descent optimization technique of the Loss function in 2.9. The Bellman Equation from which the optimal Q-value is determined is,

$$\nabla_{\theta_t} L_t(\theta_t) = \mathbb{E}[(y_t - Q(s,a;\theta_t))\nabla_{\theta_t} Q(s,a;\theta_t)] \tag{2.10}$$

The strategy that MathDQN adopts to choose which action to explore is the ϵ-greedy strategy. The goal is to obtain a balanced trade-off between exploration and exploitation after selecting a random action with probability ϵ.

The Seq2Seq models discussed in this section have 2 potential drawbacks—

- Generation of spurious numbers.
- Generation of numbers at wrong positions in the resultant equations.

2.1.6.3 Improved Seq2Seq Methods

Huang et al. [33] attempted to overcome these conundrums by proposing a model that incorporates Copy and Alignment mechanism to the Seq2Seq models, namely Cass. And just like MathDQN [32], this model also uses Reinforcement Learning and policy gradient to train itself. [33] also showed with empirical evidence that RL is a better approach than calculating Maximum Likelihood Estimation (MLE) as the objective function.

Li et al. [36] proposed a group attention[84] mechanism which aggregates 4 types of attention mechanisms with their Seq2Seq model, GroupATT. These 4 kinds of attention mechanisms are—*Global Attention* to extract global information, *Quantity-related Attention* to form relationships between quantities and adjacent/neighboring words, *Quantity-pair Attention* to model relationships between quantities, *Question-related Attention* to model the relationships between the problem statement and the quantities. The input problem is at first pre-processed to $X = \{x_1, \ldots, x_m\}$ and then morphed into $H^e = \{h_1^e, \ldots, h_m^e\}$ using Bidirectional Long Short-term Memory (Bi-LSTM). The group attention layer consists of 4 different types of multi-head attention modules. The output of this layer is,

$$O' = \text{GroupAtt}(Q = H^e, K = H^e, V = H^e) \tag{2.11}$$

The decoding process is similar to that of [75] which is,

$$y_t = \text{Softmax}(\text{Attention}(h_t^d, o_j)) \tag{2.12}$$

where, h_t^d is the hidden state at step t, o_j is the j^{th} entry or j^{th} vector from output matrix O of the group attention layer. The problem statement is at first divided and masked into the 4 inputs of group attention, $\{Q_g, K_g, V_g\}$, $\{Q_c, K_c, V_c\}$, $\{Q_p, K_p, V_p\}$, $\{Q_q, K_q, V_q\}$ for Global Attention, Quantity-related Attention, Quantity-pair Attention, and Question-related Attention respectively. Then, using Scaled Dot-Product Attention (SDPA) modules $\{O_g, O_c, O_p, O_q\}$ are computed and the output O is formed by concatenating these as,

$$O' = \text{Concat}(O_g, O_c, O_p, O_q) \tag{2.13}$$

The pre-trained word embeddings used in GroupATT are 128-dimensional and the 2-layered Bi-LSTM has 256 hidden units.

Wang et al. [35] proposed a template-based approach using Recursive Neural Network (RNN) by combining the merits of [31] and [32]. Their model, namely T-Rnn, at first predicts a tree-structured template using a method similar to Seq2Seq models. The tree's leaf nodes are numerical values and internal nodes are unknown operators. Then, an RNN is used to encode the quantities with Bi-LSTM and self-attention for inferring the unknown operators in the internal nodes of the tree in a bottom-up manner. The encapsulation of the operators in the tree template as $\langle op\rangle$ (*e.g.* predicting $n_1\langle op\rangle(n_2\langle op\rangle n_3)$ instead of $n_1 \times (n_2 - n_3)$)

and normalizing the equation as done in [75] aid in reducing the template space by a lot and makes the whole expression generation process computationally less expensive compared to [31]. In the RNN, given every quantity's embeddings W_q, every non-leaf node's representation is calculated as,

$$q_c = \tanh(W_q[q_l, q_r] + b) \tag{2.14}$$

where q_l and q_r are the left and right child's representations respectively. After this, the probability of an operator for each internal node is computed using softmax as,

$$P(o_c|q_l, q_r) = \text{softmax}(W_q \cdot q_c) \tag{2.15}$$

The calculations of (2.14) and (2.15) are performed recursively till all the operators of the internal nodes of the initial tree template is determined. Then, the in the answer generation module, the loss function that is minimized is,

$$J(\theta) = -\frac{1}{k}\sum_{i=1}^{k} P(o_c(i)|q_l(i), q_r(i)) \tag{2.16}$$

where k is the total internal nodes in the tree. The expression generation process of T-Rnn is still faulty as it is not good at predicting large tree templates and solving problems that require more external knowledge and semantic understanding.

Chiang and Chen [76] proposed a neural symbolic model based on an encoder-decoder framework, namely S-Aligned which works with the semantic meanings of operators and operands and has a novel decoding process. The decoding process generates equations using stack operations which mimics the way human beings solve such problems. The Encoder extracts the contextual semantic representation of every numeric value required for solving the MWP and for this, Bi-LSTM [90] is considered as the encoder layer.

In order to imitate the reasoning process of a human, the decoder at first generates the equation in a post-fix manner using a stack. The stack consists of both the symbolic and semantic representations of the operands as,

$$S = [(v^S_{l_t}, e^S_{l_t}), (v^S_{l_t-1}, e^S_{l_t-1}), \ldots, (v^S_{l_1}, e^S_{l_1})] \tag{2.17}$$

where, each v^S is the symbolic portion and each e^S is the semantic representation. There are 4 types of stack actions that the stack does— *Variable Generation* which generates variables with attention mechanism, *Push* which pushes the operand selected by the Operand Selector to the stack, *Operator* $\diamond$ *Application* which pops the top 2 elements, (v_i, e_i) and (v_j, e_j), forms $v_k = v_i \diamond v_j$ and calculates the semantic transformation function $e_k = f_\diamond(e_i, e_j)$ following (2.18), *Equal Application* which completes the equation by popping the top 2 elements (v_i, e_i) and (v_j, e_j), and records $v_i = v_j$ contingent on the condition that one of them is an unknown variable. The semantic representation of a new symbol is calculated by the proposed Semantic Transformer as,

$$f_\diamond(e_i, e_j) = \tanh(U_\diamond \, \mathrm{ReLU}(W_\diamond[e_i; e_j] + b_\diamond) + c_\diamond) \tag{2.18}$$

where, $W_\diamond, U_\diamond, b_\diamond$ and $c_\diamond$ are the parameters of the model. In the case of Semantic Transformers, the parameters are different for different operators which results in modelling different transformations. Although S-Aligned is one of the more sophisticated models, it falters in cases of language ambiguity and operand ordering in complex comparison problems.

2.1.6.4 Graph-based Methods

Xie and Sun [37] also proposed a novel neural model that tries to imitate the humans. The model, namely Gts, focuses on the goal-driven mechanism in problem-solving. It generates a recursive expansion tree where the root node is the main goal and each node gets subdivided into two child nodes with simpler goals until the goal is simple enough. At first, in the *Encoding* step, a left-to-right, then a right-to-left Gated Recurrent Unit (GRU) over word embeddings of problem text is used to encode the problem. Then, in the *Root Goal Initialization* step, the root goal is initialized as the sum of final hidden states of forward and backward sequence respectively. Next, in the *Token Embedding* step, token embeddings are generated using matrices for operators and constants and encoded numerical values from problem statements. Then, in the *Top-Down Goal Decomposition* step, each goal is decomposed into two simpler goals until the predicted tokens are numeric values. Then, in the *Subtree Embedding* step, a subtree is encoded using bottom-up manner using RNN. GTS trumps over SEQ2SEQ models due to its ability to avoid mathematically invalid equations and spurious numbers. GTS's Subtree Embedding module prevents generating the same subtree as its left sibling when the internal node is $+$ or $\times$. The main drawbacks of GTS are — it doesn't recognize various mathematical laws such as Commutative Law in Addition and Multiplication and it fails to generate more than one valid solution expression for a problem.

Liu et al. [41] proposed a tree-structured top-down hierarchical SEQ2TREE model with a decoding method that generates the Abstract Syntax Tree (AST) of the equation by utilizing an auxiliary stack. The encoder layer of this model, namely AST-DEC, is a Bi-LSTM layer which encodes the sequential information. The tree-structured decoder uses LSTM to generate the final equation template maintaining a top-down approach and due to the use of auxiliary stack in the decoding process the resultant equation is of prefix notation. AST-DEC falters in the case of problems with long equation templates and Profit/Geometry problems of the MATH23K dataset due to the lack of better external knowledge.

Meng and Rumshisky [77] pioneered the use of Transformer as a decoder to generate math equations. Their proposed model, namely D-DECODER, has two Transformer decoders operating in opposite directions. Unlike [33], D-DECODER doesn't resort to any Copy and Align modules.

The canonical Transformer model is used as the encoder in D-DECODER and its training performance improves by a significant margin due to the use of two decoder transformers, very similar to the masked language model in Bidirectional Encoder Representations from Transformers (BERT)[38].

Zhang et al. [57] proposed a novel deep learning model, namely GRAPH2TREE, that fuses the merits of Graph-based Transformer[59, 60] Encoders and Tree-based Decoders to solve MWPs. The principal advantage of using Graph-based Encoder is that it can capture the relationships and associations between quantities better than Sequence-based Encoder. GRAPH2TREE uses *Quantity Cell Graph* and *Quantity Comparison Graph* to improve the quantity representations and model the relationships among the attributes and quantities (along with their numerical quality) respectively by leveraging certain heuristics. In the encoding step, GRAPH2TREE encodes the given MWP statement using Bi-LSTM and builds the Quantity Cell Graph and the Quantity Comparison Graph using the word-level representations of the Bi-LSTM as their nodes. The multiGCN (Graph Convolutional Networks)[61] module of the Graph Transformer then learns the entire graph representation of the MWP based on the Quantity Cell Graph and the Quantity Comparison Graph. Then, pooling is used to combine all the nodes into a pool-based graph embedding vector which is output of the Graph Transformer. This graph representation and updated node representations are then used by the tree-structured decoder to infer the resultant expression tree. GRAPH2TREE uses a K-head graph convolution setup. So the parameters that a single GCN learns are $W_{gk} \in \mathbb{R}^{d \times d_k}$, where $d_k = \frac{d}{K}$, and the learning is done as,

$$\begin{aligned} GCN(A_k, X) &= GConv_2(A_k, GConv_1(A_k, X)); \\ GConv(A_k, X) &= \text{ReLU}(A_k X^T W_{gk}), k \in [1, \dots, K] \end{aligned} \tag{2.19}$$

where, A_k is the adjacency matrix of the k-th graph, X is the feature matrix. Each $GCN(A_k, X)$ operation yields a d_k dimensional output which are then concatenated together to form the final output.

$$Z = \Big\Vert_{k=1}^{K} GCN(A_k, H) \tag{2.20}$$

Z is then passed to a Feed-forward network, Layer-norm layer, Residual layer, and a Min-pooling layer,

$$\begin{aligned}\hat{Z} &= Z + \text{LayerNorm}(Z) \\ \bar{Z} &= \hat{Z} + \text{LayerNorm}(FFN(\hat{Z}))\end{aligned} \tag{2.21}$$

where, $FFN(x)$ is a 2-layered Feed-forward Network with the ReLU activation function between the layers,

$$FFN(x) = \max(0, xW_{f_1} + b_{f_1})W_{f_2} + b_{f_2} \tag{2.22}$$

where, f_1 and f_2 are the notations for the 2 layers and W and b are the weights and biases respectively. After min-pooling and feeding to a Fully-connected Neural Network (FC), the encoding process is complete and the graph representation is generated,

$$z_g = FC(\text{MinPool}(\bar{Z})) \tag{2.23}$$

The tree-structured decoder of GRAPH2TREE is very similar to the one in GTS proposed by [37]. The loss function, which is the sum of the negative log-likelihoods of probabilities, that is minimized using Adam optimizer[92] is,

$$L(T, P) = \sum_{t=1}^{E} -\log(\text{P}(y_t|q_t, G_c, P)) \tag{2.24}$$

where, q_t is the goal vector, G_c is the contextual Global Graph, y_t is the prediction token of node t, E is the total token present in tree T and P is given problem statement. Although GRAPH2TREE is one of the better models to solve MWPs, it still falters in the cases of long solution expressions and problems that require complex reasoning to solve.

Li et al. [58] also proposed another GRAPH2TREE model with a hierarchical tree decoder consisting of Sub-decoders which perform *Parent Feeding* and *Sibling Feeding*. They also proposed the *Separate Attention Mechanism* to locate source sub-graphs in the decoding process.

Liu et al. [78] proposed a novel data augmentation method, namely RODA, that aids MWP solver models to learn mathematical reasoning logic with better effectiveness. This method attempts at mimicking the behavioral proclivity of humans to double-check their math problem solutions to guarantee their correctness. This effect is achieved by restructuring the problem statement of the MWP *e.g.* the original problem text is *"Jeff has 2 apples. Felix gives him 3 more apples. How many apples does Jeff have?"* and the MWP solver finds the solution to be $x = 2 + 3 = 5$. Now, to double-check this solution, RODA will reformulate the original question by converting the interrogative sentence to a declarative sentence by plugging in the obtained answer and the newly formulated problem text will ask for one of the known numeric values. So the given example gets restructured as *"Jeff has 2 apples. Felix gave him some apples and now Jeff has 5 apples. How many apples did Felix give Jeff?"*, and the MWP solver will obtain the answer $y = x - 2 = 5 - 2 = 3$ which matches with the known value. This reversion-based data augmentation tactic has 3 distinct merits—it is computationally simple and reliable, at the same time it helps to garner more knowledge points which help enhance the model's mathematical reasoning capability, and moreover MWP datasets and corpora with greater complexity can be used to train the model.

Hong et al. [79] also attempted to replicate the mental states of a human during problem-solving in their proposed model SMART. In order to represent situation models for MWP solving, SMART uses Attributed Grammar. [79] also unveiled a new dataset ASP6.6K and designed an Out-of-Distribution (OOD) evaluation procedure. SMART was inspired by the cognitive concept of *Situation Model* [93] which is used to get an abstract idea about the mental states of human beings while problem-solving in the realm of psychology and behaviorism. At first, using context-free grammar, a hierarchical parse graph is constructed from the problem text and the nodes of that graph are the world, agents, entities, quantities, and events mentioned in the problem text. The second stage of SMART's learning process is an iterative method to generate pseudo-gold graphs which augment the supervision for the subsequent iteration. This step helps in strengthening the information extraction module in SMART and consequently the model achieves good interpretability.

2.1.6.5 Complex Encoder-Decoder based Methods

In order to overcome the problems of expression fragmentation and operand-context separation, Kim et al. [80] proposed a pure neural model, EPT, which stands for Expression-Pointer Transformer and is the first of its kind for solving MWPs. The encoder layer in EPT is the ALBERT model[94] which is a pre-trained natural language model. The decoder uses the encoder's hidden state vectors as memories and generates Expression tokens as output. These expression tokens are then used to construct the final solution expression. The operand-context pointer aids in differentiating between different expressions as it directly points to the operands' contextual information.

Shen and Jin [42] proposed a model with multiple encoders and decoders to further enhance text representations and expression derivations. Their model, namely MULTIE/D, uses two types of encoders — Sequence-based and Graph-based encoders and also uses two types of decoders — Sequence-based and Tree-based decoders. Due to the integration of Graph-based encoder alongside Sequence-based encoder, the model achieves an enhanced representation of the problem text through a dependency parse tree and numerical comparison information.

The sequence-based encoder layer uses Bi-GRU and the sequence-based decoder layer uses GRU. The graph-based encoder is a GRAPHSAGE model [95] which is a flexible Graph Neural Network (GNN). The tree-based decoder is similar to the one proposed in [37].

Wu et al. [43] proposed a model which can integrate external knowledge and global expression information. This novel model, namely KA-S2T abbreviation for Knowledge-aware Sequence to Tree, is a SEQ2TREE model in which there is a Bi-LSTM encoder that constructs an entity graph using the entities extracted from the MWP text and passes to a Graph Attention Network which captures the Knowledge-aware problem representations. The decoder is a tree-based decoder combined with a state aggregation procedure that is effective in extracting the long-distance dependency and global expression information and it outputs the math expression by performing pre-order traversal in the generated tree.

The research works done in the aforementioned papers consider only one expression from the dataset and try to fit it. Zhang et al. [51] introduced a new model, Teacher-Student Networks with Multiple Decoders (Tsn-Md), with a *Teacher* network and multiple decoder *Student* networks to derive multiple correct expressions for the same problem text. The teacher network is pre-trained to guide the learning behaviors of the student networks. The student networks participate in a vote and democratically choose one of the multiple candidate expressions generated by the decoders. This model can thus generate diversified solutions for the same problem statement using the idea of Knowledge Distillation (KD)[52] in the teacher network which is very similar to the Gts model in[37]. Tsn-Md can produce expressions whose templates are not present in the original dataset but the model doesn't penalize the generation of such expressions if they end up yielding the correct final answer.

Yu et al. [53] proposed a Reasoning with Pre-trained Knowledge and Hierarchical Structure (Rpkhs) network which has special encoders. There are 2 types of encoders in Rpkhs and they are *Pre-trained Knowledge* encoder and *Hierarchical Reasoning* encoder. The hierarchical encoder uses the textual embeddings of the problem text to form relationships between word and sentence thus integrating semantics between entity and context. The knowledge encoder extracts knowledge points from the linguistic world and incorporates them into the input embeddings to enhance the representation. The outputs of these two encoders are concatenated and fed to the tree-based decoder to derive the final expression. Rpkhs holds the highest recorded accuracy of 89.8% in the Mawps dataset.

Hong et al. [54] modified the work of Xie and Sun [37] by incorporating a novel symbolic reasoning based *Learning-by-fixing* (Lbf) framework. In essence, the idea is to traverse through an incorrect expression tree from the root to the inner nodes and determine the most probable *fix* that will make the tree yield the correct expression (*exploring stage*). In order to generate diverse solutions, Lbf uses *Tree Regularization* and to make the process memory-efficient, it also uses *memory buffer*.

Lin et al. [44] proposes a novel Hierarchical Math Solver (HMS) that has a hierarchical word-clause-problem encoder that imitates the reading habits of humans and has a novel hierarchical attention mechanism-based tree decoder that generates the math expressions.

Qin et al. [55] proposed the novel Neural Symbolic Solver (NS-SOLVER) that performs 4 auxiliary tasks to integrate different levels of symbolic constraints (*e.g.* commonsense constants and formulation regularization). The model has 3 main modules—the problem encoder, the symbolic equation generator decoder and a symbolic executor. The 4 auxiliary objectives are—Number Prediction, Commonsense Constant Prediction, Program Consistency Checking, and Duality Exploitation. The authors also created the CM17K benchmark dataset with 4 types of MWPs.

Huang et al. [56] attempted to emulate human-like analogical learning in their proposed model REAL. They proposed a novel memory-augmented framework consisting of memory modules that are used for retrieving math problem statements similar to a test sample. Each retrieved problem along with the test sample is encoded using a representation module. This concatenated problem set is then fed to the analogy and reasoning modules to obtain the output expression.

Cobbe et al. [69] from OpenAI proposed the use of verifiers to guesstimate the correctness of model outputs. The incorporation of verifier modules results in a performance improvement comparable with a 30x model size expansion. The finetuning baseline method of their model utilizes the same modeling objective as GPT-3 [6]. The verification process is basically sampling and scoring a multitude of high-temperature solutions and thereby outputting the highest-scored solution as the answer.

Liang et al. [46] proposed the MWP-BERT model. It takes advantage of the perfect Natural Language Understanding done by Bidirectional Encoder Representations from Transformers (BERT)[38]. The pre-trained token representations are used by the model to capture the relationship between the problem text and mathematical logic. The model has a BERT-based encoder which is further trained on the APE210K dataset to perform Masked Language Model (MLM) so that the encoder can learn to understand mathematical logic besides natural language. The resultant problem embedding is obtained after Mean-pooling and then fed to the tree-based decoder to generate the correct expression tree. In terms of performance, to the best of our knowledge, this MWP-BERT is the best among the lot. It yielded the then highest known accuracy scores of 84.4% (96.2% in evaluation) and 84.3% across two of the largest benchmark datasets MATH23K and APE210K respectively. So, in the domain of MWP solving, MWP-BERT was regarded as the State-of-the-Art (SOTA), especially for the APE210K dataset. However, it was surpassed by the SCR model proposed by Xiong et al. [49] which now boasts accuracy values of 86.8% and 76.7% in the MATH23K and APE210K datasets respectively. It utilizes a RoBERTa-based encoder and a Tree-based decoder.

A fair inquisitive pursuit is to see if this model eventually learns Math or Language. Patel et al. [2] carried out significant probing tests to expose the deficiencies in linguistic and mathematical modeling of the baseline BERT model. The model can easily solve a simple test sample from MAWPS[1] such as, "Jerry had 135 pens. John took 19 pens from him. How many pens does Jerry have left?" and provide the correct math expression "$x = 135 - 19$" as output. However, if questions are posed in a different manner, for example, "Jerry had 135 pens. John took 19 pens from him. Who has more pens?" or "What should be added to 19 to make it 135?" the model fails to provide the correct output, which suggests that these translation models are not sufficiently adept at modeling either language or math.

Shen et al. [45] proposed a new "Generate & Rank" method, which is a BART-based [39] multi-task framework for MWPs. BART is a pre-trained language model with an illustrious record of achieving SOTA performances in a diverse range of NLP research domains. In accordance with that, the "Generate & Rank" method managed to surpass MWPBERT by a slight margin and achieved a new SOTA accuracy of 85.4% in the MATH23K dataset. It also achieved an accuracy of 84.0% in the MAWPS dataset which is the second highest recorded accuracy as far as Mawps is concerned, just behind Rpkhs[53]. The idea here is to generate a set of candidate expressions from a Math Word Problem (MWP) statement and rank those candidate expressions to determine the most correct one. The ranking process is learnt through exposure to training examples from an expression bank consisting of several correct expressions. This joint training task coupled with tree-based disturbance and online updates makes the model learn from its mistakes and learn to generate correct math expressions by improving its ability to discriminate between math expressions in terms of correctness and validity.

Jie et al. [63] proposed the RoBERTa DeductReasoner model which views the MWP solving task as a *complex relation extraction* problem. They developed a deductive reasoning module at the downstream of their model which yields human-understandable deductive reasoning steps iteratively and constructs the final output expression. This procedure of deduction enables the RoBERTa DeductReasoner model to make correct equation predictions on problem statements that require a somewhat deep and complex line of reasoning. Their model boasts the SOTA accuracy on the popular English MWP dataset Mawps [1] and the challenge dataset Svamp [2] among the non-gigantic language models. The accuracy values are 97.0% and 47.3% respectively.

Deep Learning-based translation models generally learn some shortcuts by identifying patterns in the training data[96]. This works as a double-edged sword because the model may provide an output for a peculiar input by resorting to shallow generalizations. Patel et al. [2] duly exhibited how deep learning-based MWP solvers provide the supposed "correct" output even though the input prompt was simply a situational context of a problem scenario without any questions posed at the end. The most obvious reason for this is the high lexical and template overlap seen in the datasets these models are trained with. It also insinuates some issues with the internal design of the models as pointed out by Patel et al. [2].

A summary table of some of the Deep Learning Encoder-Decoder based models is shown in table (2.1).

Table 2.1: The encoders and decoders designed by various models.

Model	**Seq-Encoder**	**Graph-Encoder**	**Seq-Decoder**	**Tree-Decoder**
DNS	✓		✓	
Math-EN	✓		✓	
T-RNN	✓		✓	
S-Aligned	✓			
Group-ATT	✓			
D-Decoder	✓			
AST-Dec	✓			✓
GTS	✓			✓
Graph2Tree	✓	✓		✓
MultiE/D	✓	✓	✓	✓
TSN	✓			✓
MWP-BERT	✓			✓

2.1.6.6 Large Language Model-based Methods

With the advent of Large Language Models (LLM), the landscape of Natural Language Understanding (NLU) evolved to a great extent. LLMs like GPT-1 with 117 million parameters [97], GPT-2 with 1.5 billion parameters [98], T5 with 11 billion parameters [99], and finally GPT-3 with 175 billion parameters [6] are some of the gold-standard language models in the industry. They are mainly prompt-based models which enable them to perform zero-shot, one-shot or few-shot learning. Wang et al. [66] used OpenAI's *code-davinci-002* model which is a variant of the GPT-3 model to check its mathematical reasoning performance on a series of English MWP datasets. With their *Self-Consistency* prompting module, they were able to reach astounding accuracies of 91.6%, 100.0%, 87.8%, 52.0%, 86.8% and 78.0% on the datasets AddSub, MultiArith, ASDiv, AQuA, Svamp and GSM8k respectively. Li et al. [65] developed a DiVeRSe (Diverse Verifier on Reasoning Step) approach to enable few-shot learning of MWP solving in LLMs. The approach is essentially prompting the LLM to generate a variety of reasoning paths to solve a problem statement and then selecting the most voted reasoning path using a verifier.

2.1.7 Relevant Features Extraction

A variety of features were proposed in the pre-Deep Learning research works discussed in this paper. We categorize these features and identify the MWP solvers that work with those corresponding features.

2.1.7.1 Quantity-related Features

As mentioned earlier in this paper, quantities are determinate numerical values that signify the amount of some object, entity, rate or unit. As this is the most obvious feature that needs to be extracted in order to solve a math problem, all the MWP solvers extract these values as operands in the final expression. However, an extra effort to determine whether a numerical value is a rate or not has been adopted in recent works [18, 23, 25, 29, 32]. These *rate* quantities are generally associated with the $\times$ and $\div$ operators. The quantity feature is also important to determining relevancy of the numerical values. Quantities written in textual format in the problem statement, *e.g.*, “six”, “nine”, have low likelihood to be relevant or be included in the final solution equation [15, 18, 29, 30]. For example, in the statement, “The sum of *two* numbers is 420.”, the quantity “two” isn’t relevant to the solution expression of the problem.

2.1.7.2 Context-related Features

The contextual information of the quantities need to be extracted to correctly solve a math problem. MWP solvers generally leverage the word lemmas, Part of Speech (POS) tags and dependence types within the text window centered at a particular quantity. A good rule of thumb used in works like [23, 25, 32] is to check for comparative adverb terms, such as, “more”, “less”, “than” and these keywords are generally associated with operators like, $+$ and $-$.

2.1.7.3 Quantity-pair Features

Some of the operators in the solution expression of a problem is determined by checking the relative associations between two quantities. As discussed before, quantities with the same unit generally have addition (+) or subtraction (−) operations between them, whereas, quantities that are units and quantities that represent some rate are usually multiplied (×) or divided (÷)[22, 23, 25, 32]. Zhou et al. [18], Upadhyay et al. [29] introduced two types of quantity-pair features, namely, Numeric relation and Context Similarity. The numeric relation feature forms two sets of entity nouns situated in the same sentence of the problem statement and arranges them in terms of distance in the dependency tree. Then, scoring functions like Jaccard Similarity Coefficient is used to measure the similarity between these lists of entity nouns. The context similarity feature is useful for determining the expression template needed for the solution of the problem. The higher the contextual similarity between two quantities is, the more likely they are to be placed in symmetric slots in the final equation[15, 18]. For example, if the problem statement says something like, "A pencil costs 10 cents and a pen costs 20 cents.", the price quantities "10" and "20" are deemed contextually similar. Another quantity-pair feature that is taken into consideration is to check whether one quantity from the pair is greater than the other or not[15, 18, 23, 25, 29, 32]. This is done to correctly set the order of the operands in the solution expression for operators like, − and ÷.

2.1.7.4 Question-related Features

The problem statements of MWPs generally pose a question asking for the value of one or more unknown variables. In order to figure out what those unknown variables signify, some question-related features need to be extracted. If the question consists of a particular unit or noun phrase, the quantities associated with that unit or noun phrase throughout the problem statement, are likely to be relevant[15, 17, 22, 23, 25, 32]. The relevancy of units and noun phrases of a quantity can also be measured by checking if they have the highest number of match tokens with the question posed at the end of the problem statement. The presence of some keywords in the question that insinuate the rate of an entity, *e.g.*, "each" or "per" are also taken into consideration and are useful for including

$\times$ and $\div$ operators in the final expression. Other comparison-related keywords such as, “more” or “less” are also extracted for inclusion of $+$ and $-$ operators in the final expression[18, 29].

2.1.7.5 Verb-related Features

Verbs form the main part of the predicate of the sentences in a problem statement and they generally describe a state, an action or an occurrence. These type of words have to be extracted and identified for determining the correct operators in the final expression. For example, in the problem statement excerpt “Rifat has 20 books. Nafis takes 4 books from Rifat.”, the verb “takes”, which is the plural form of “take”, implicates a loss of quantity for the entity “Rifat” in the aforementioned scenario. In terms of operators, it is obvious that the subtraction operator ($-$) has to be used to indicate this loss of quantity. The *dependent verb* of a quantity, which is used as a feature, is the verb closest to that given quantity in the dependency tree[22, 23, 25, 32]. If the dependent verb isn’t verbatim present in the pre-defined set of verbs useful for determining mathematical operators for the final expression, another feature is used which is a vector representing the distance between the dependent verb and the constituents of the aforementioned set of verbs[16, 20, 22]. The quantities that share the same dependent verbs are deemed closely associated in terms of undergoing the same action or being in the same state[23, 25, 32]. For example, in the statement “Rachel bought 10 coloring books last month. This month she bought 5 coloring books.”, the verb “bought” acts as the dependent verb for for both the quantities 10 and 5. In some cases, there may be a single presence of the dependent verb but nonetheless acts as dependent verb for multiple quantities[23, 25, 32]. For example, in the statement, “Rachel drew 2 cats and 3 rabbits.”, the verb “drew”, which is the past tense of “draw”, manifests only once but acts as the dependent verb for both quantities 2 and 3.

2.1.7.6 Global Features

Some of the existing research works[15, 17, 24, 25, 32] define some overarching global features in the document-level. One part of this feature space is the tally count of quantities in the problem statement. Another thing to consider for understanding the sentences in the problem statement is the n-gram contiguous sequence of words. In [15, 17] the authors proposed using unigrams ($n = 1$) or bigrams ($n = 2$) for the problem text. These features are indispensable for determining which quantities are relevant and their relative order in the final expression.

2.1.8 Dataset Repository and Performance Analysis for MWPs

2.1.8.1 AI2

Hosseini et al. [16] created this dataset consisting of 395 Math Word Problem from 3rd-5th grade children's books. The problems were scraped from two websites: www.math-aids.com and www.ixl.com. It is partitioned into 3 subsets based on difficulty. All in all, the dataset consists of 13,632 words, 118 verbs and 1,483 sentences. This dataset is also referred to as ADDSUB.

2.1.8.2 IL

It was created by [17] and consists of 562 single step math word problems. The problems were harvested from two websites k5learning.com and dadsworksheets.com. The dataset has problems that require various combinations of the 4 fundamental mathematical operators to solve. All the problem statements were formulated to keep irrelevant quantities away. So models without any relevance classifier can train on this dataset.

2.1.8.3 SingleEQ

This is the first dataset to consist of both single and multi-step math word problems. A subset of the problems in this dataset are also within the AI2 dataset.

This was created by [22].

2.1.8.4 AllArith

This dataset is the amalgamation of 4 other datasets which are, AI2, IL, CC and SINGLEEQ. The dataset consists of mostly unique problems as near-duplicate problems were manually removed. It was created by [25].

2.1.8.5 ALG514

As the name suggests, this dataset consists of 514 problems harvested using web-crawlers on www.algebra.com. This was created by [15]. They did not include any problems that required explicit background knowledge or problems whose templates didn't appear in the dataset $\geq$ 6 times. After imposing these data cleaning criteria on the randomly selected pool of 1,024 problems, the authors ended up with 514 problems with a total of 1,616 sentences and 19,357 words. The vocabulary size was 2,352 with an average of 37 words, 3.1 sentences and 13.4 nouns per math problem. There were a total of 28 unique equation templates with each equation template system having 7 slots on average. The mean derivations for each problem was 4,000,000. This dataset is also referred to as SIMULEQ-S.

2.1.8.6 Dolphin1878

The problems of this dataset are from www.algebra.com and answers.yahoo.com. The dataset also contains manually annotated answers and equations. In total, the dataset has 1,878 problems as the name suggests and there are 1,183 equation templates. Shi et al. [26] created this dataset to test out their semantic parsing based model.

2.1.8.7 DRAW1K

The authors of DRAW1KUpadhyay et al. [29] were incentivized to create this dataset due to the lack of diversity in the DOLPHIN1878 dataset. This dataset consists of 1,000 linear equation problems scraped from www.algebra.com.

2.1.8.8 Dolphin18K

This dataset was constructed semi-automatically with a lot of manual annotations. Huang et al. [28] prepared this dataset in 4 stages. It is one of the largest benchmark datasets consisting of 18,460 math problems with 5,871 equation templates. The problems were scraped from online fora like, Yahoo! Answers.

2.1.8.9 AQuA

This dataset was published by DEEPMIND after being created by Ling et al. [100]. This dataset consists of a total of around 100,000 annotated math problems of which 34,202 were collected by the authors and 70,318 were collected and annotated by volunteers. The problems were collected from Graduate Management Admission Test (GMAT) and Graduate Record Examination (GRE) question papers and so they are pretty challenging.

2.1.8.10 MathQA

This dataset was created by Amini et al. [101] and it consists of MWPs in the form of Multiple Choice Questions (MCQ). A total of 37,295 questions of various categories were included in this dataset. The categories include Geometry, Physics, Probability, Gain-Loss, etc. which are domain-specific types of problems that require some knowledge about the respective domains to solve. For example, knowing the value of $\pi = 3.1415...$ for solving geometry problems that ask for the area of a circle given the radius, diameter, or circumference.

2.1.8.11 HMWP

This dataset was created by Qin et al. [102] with problems extracted from a Chinese K12 math word problem bank book. The full-form of HMWP is ***H****ybrid* ***M****ath* ***W****ord* ***P****roblem.* This dataset consists of 3 types of MWPs: arithmetic, equation-set and non-linear equation problems. Out of the total 5,471 problems, 2,955 are 1-unknown variable linear problems, 1,636 are 2-unknown variable linear problems and 900 are 1-unknown variable non-linear problems. The creators of this dataset claim the sufficiency of this dataset as a testbed for testing the universality of MWP solver models.

2.1.8.12 CM17K

Qin et al. [55] created this dataset with the same intentions behind creating HMWP[102], which is to validate the universality of an MWP solver model. The problems in this dataset were collected from Chinese math textbooks for grade 6-12 students. An aggregate of 17,035 problems of 4 types are present in this dataset. These types are, 6,215 arithmetic MWPs, 5,193 1-unknown linear MWPs and 2,498 equation-set problem statements. The authors claim that CM17K is better suited than MATH23K[31] to test the reasoning ability and universality of MWP solver models, due to its problem-type diversity.

2.1.8.13 ASDiv

This dataset was created by Miao et al. [103]. The full-form of ASDIV *is* ***A****cademia* ***S****inica* ***Div****erse MWP Dataset.* The incentive behind creating this dataset was the lack of linguistic and problem-type diversity in the then existing MWP datasets. It consists of a total of 2,305 problems with varying lexical patterns, difficulty levels and problem-types collected from 28 websites. There are a total of 19 unique equation templates and on average each problem requires 1.23 operations to solve.

2.1.8.14 SVAMP

Patel et al. [2] created this challenge dataset to test the robustness of the then existing MWP solver models. The elaboration of SVAMP is ***S****imple* ***V****ariations on* ***A****rithmetic* ***M****ath Word* ***P****roblems.* The principal reason for creating this dataset was to expose the brittle nature of the SOTA MWP solver models and to demonstrate that those models rely on simple heuristics learnt from the training data to generate the predicted math expressions. The initial seed examples of this dataset were taken from the ASDIV-A[103] dataset and certain variations of those problem statements were used to build the entire dataset. The variations were introduced in the form of structural difference, object difference, addition of relevant/irrelevant information, changing entities and quantities, changing the order of objects, changing the order of phrases and inverting the operations. These variations test a solver model's *Question Sensitivity*, *Reasoning Ability* and *Structural Invariance*. This dataset has a total of 1,000 problems, maintains a single operation to multiple operations problem ratio of 1.24:1 and has 26 unique templates. SVAMP has a lower Corpus Lexicon Diversity (CLD) score than ASDIV-A[103], but the creators of SVAMP argue that Corpus Lexicon Diversity (CLD) is not a reliable way to measure the quality of MWP datasets.

2.1.8.15 MAWPS

This is one of the most popular benchmark datasets consisting of 2,373 math word problems. The problem are of different levels of difficulty and reasoning complexity. Any model that performs good on this dataset can be considered quite a robust model as this dataset is a really good testbed. This was created by Koncel-Kedziorski et al. [22] and Koncel-Kedziorski et al. [1].

2.1.8.16 DolphinS

This dataset is a fragment of the DOLPHIN18K[28] dataset. It consists of a total of 7,070 problem out of which 115 are single operator problems and the rest 6,955 are multiple operator problems.

2.1.8.17 GSM8K

Cobbe et al. [69] created this dataset to test their verifier incorporated GPT-3 based model. It consists of 8,500 math word problem statements manually annotated and created by human problem authors. These problems are diversified in terms of number of steps required to solve them but the difficulty level of the problems is not that high as a generic middle-school student with decent mathematical aptitude is able to solve all of them.

2.1.8.18 Math23K

This dataset, created by Wang et al. [31], is an important dataset to test whether an MWP solver model is language agnostic or not because the problems in this dataset are written in Mandarin. The problems were taken from Chinese elementary school text-books and a myriad of educational websites. Although it initially had 60,000 problems, the authors manually discarded many unabiding problems and reduced the problem number down to 23,161 with 2,187 templates which still makes it a huge benchmark dataset suitable for training Deep Learning models.

2.1.8.19 Ape210K

This huge benchmark dataset was recently created by Liang et al. [46] who implemented the MWP-BERT. This dataset consists of 210,488 math word problems and is the largest MWP dataset created till date. This dataset consists of problems that require commonsense knowledge which makes it a good testbed for models with Keyword-based prompt-matching mechanisms to learn commonsense knowledge points. Many of the problems in the dataset are either not annotated with the correct equation or not annotated with the correct answer or both. The authors willingly did this to make the test-set more challenging for the models. The unclean version of this dataset is called APE210K while the clean version in which all the problems are annotated is called APECLEAN. APECLEAN has a total of 125,675 problems out of which 122,588 are training samples and the rest 3,087 are test samples.

A summary of the datasets is portrayed in table (2.6) and a comparative analysis of the performance of the discussed models on the benchmark datasets is portrayed in table (2.7).

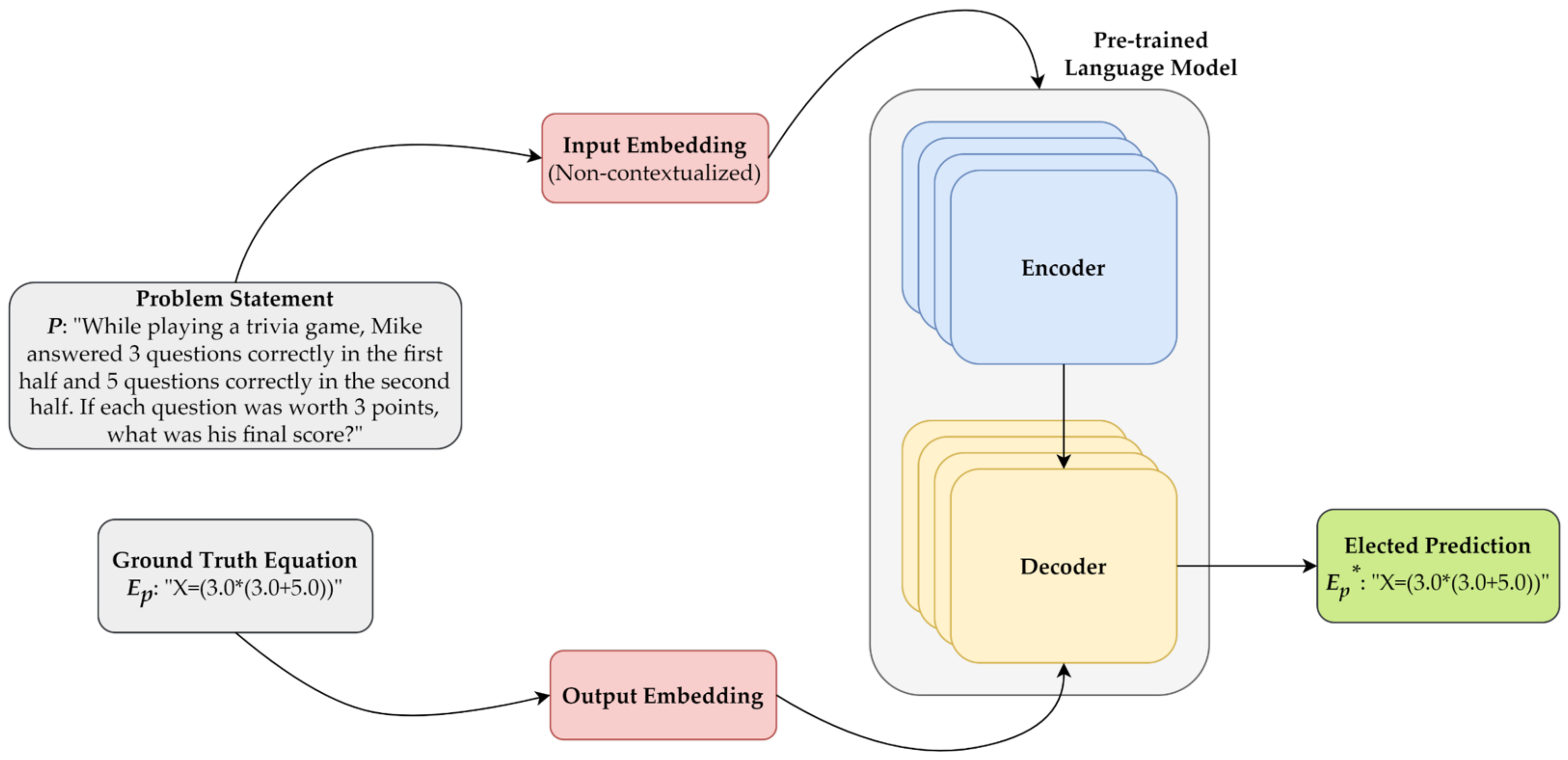
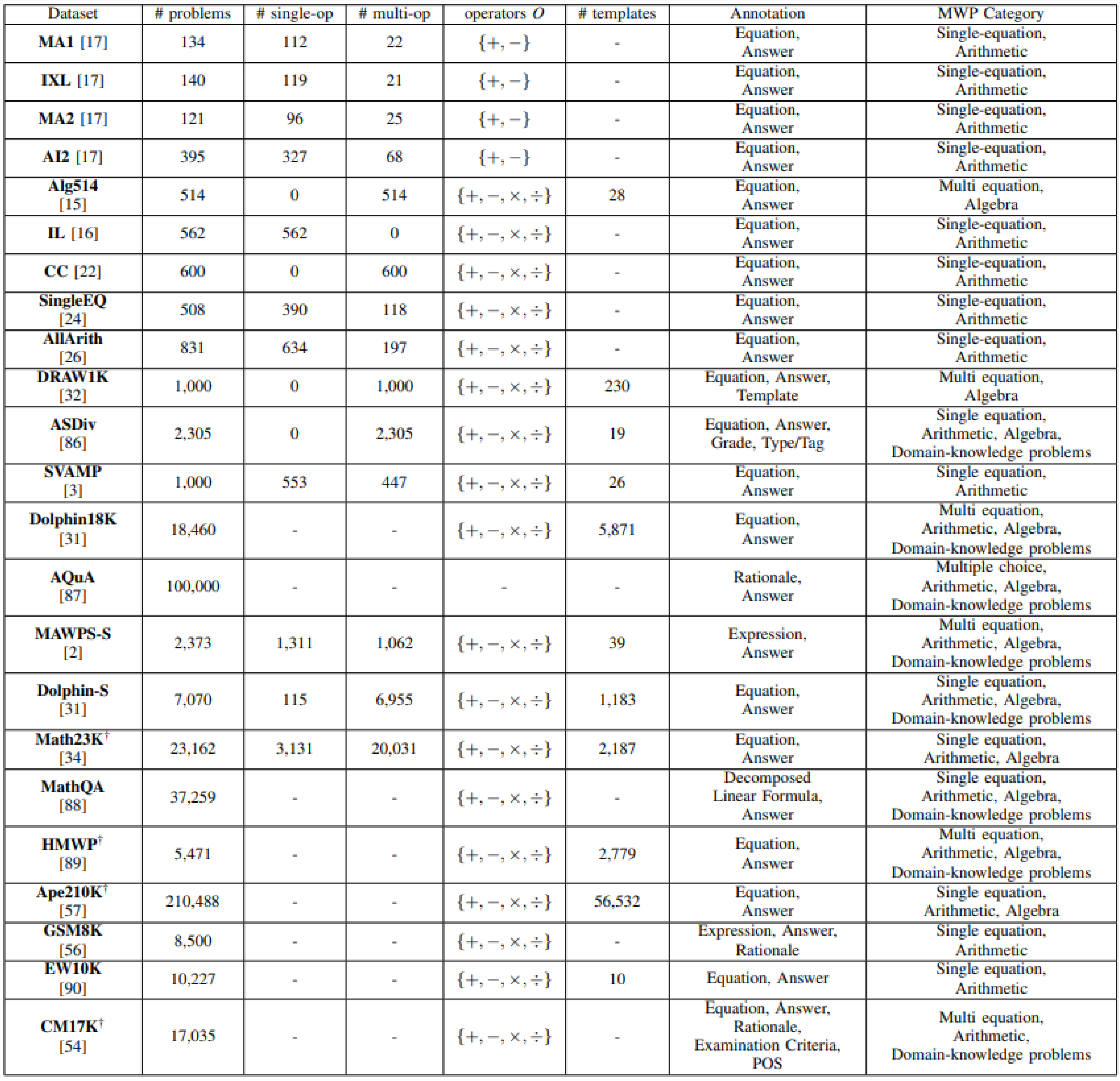

Pre-trained
Language Model
Input Embedding
(Non-contextualized)
Encoder
Problem Statement
P: "While playing a trivia game, Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half. If each question was worth 3 points, what was his final score?"
Decoder
Elected Prediction
E_p^*: "X=(3.0*(3.0+5.0))"
Ground Truth Equation
E_p: "X=(3.0*(3.0+5.0))"
Output Embedding

Figure 2.5: General Pipeline of DL-based pre-trained language models for MWP solving.



Dataset	# problems	# single-op	# multi-op	operators O	# templates	Annotation	MWP Category
MA1 [17]	134	112	22	$\{+,-\}$	-	Equation, Answer	Single-equation, Arithmetic
IXL [17]	140	119	21	$\{+,-\}$	-	Equation, Answer	Single-equation, Arithmetic
MA2 [17]	121	96	25	$\{+,-\}$	-	Equation, Answer	Single-equation, Arithmetic
AI2 [17]	395	327	68	$\{+,-\}$	-	Equation, Answer	Single-equation, Arithmetic
Alg514 [15]	514	0	514	$\{+,-,\times,\div\}$	28	Equation, Answer	Multi equation, Algebra
IL [16]	562	562	0	$\{+,-,\times,\div\}$	-	Equation, Answer	Single-equation, Arithmetic
CC [22]	600	0	600	$\{+,-,\times,\div\}$	-	Equation, Answer	Single-equation, Arithmetic
SingleEQ [24]	508	390	118	$\{+,-,\times,\div\}$	-	Equation, Answer	Single-equation, Arithmetic
AllArith [26]	831	634	197	$\{+,-,\times,\div\}$	-	Equation, Answer	Single-equation, Arithmetic
DRAW1K [32]	1,000	0	1,000	$\{+,-,\times,\div\}$	230	Equation, Answer, Template	Multi equation, Algebra
ASDiv [86]	2,305	0	2,305	$\{+,-,\times,\div\}$	19	Equation, Answer, Grade, Type/Tag	Single equation, Arithmetic, Algebra, Domain-knowledge problems
SVAMP [3]	1,000	553	447	$\{+,-,\times,\div\}$	26	Equation, Answer	Single equation, Arithmetic
Dolphin18K [31]	18,460	-	-	$\{+,-,\times,\div\}$	5,871	Equation, Answer	Multi equation, Arithmetic, Algebra, Domain-knowledge problems
AQuA [87]	100,000	-	-	-	-	Rationale, Answer	Multiple choice, Arithmetic, Algebra, Domain-knowledge problems
MAWPS-S [2]	2,373	1,311	1,062	$\{+,-,\times,\div\}$	39	Expression, Answer	Multi equation, Arithmetic, Algebra, Domain-knowledge problems
Dolphin-S [31]	7,070	115	6,955	$\{+,-,\times,\div\}$	1,183	Equation, Answer	Single equation, Arithmetic, Algebra, Domain-knowledge problems
Math23K† [34]	23,162	3,131	20,031	$\{+,-,\times,\div\}$	2,187	Equation, Answer	Single equation, Arithmetic, Algebra
MathQA [88]	37,259	-	-	$\{+,-,\times,\div\}$	-	Decomposed Linear Formula, Answer	Single equation, Arithmetic, Algebra, Domain-knowledge problems
HMWP† [89]	5,471	-	-	$\{+,-,\times,\div\}$	2,779	Equation, Answer	Multi equation, Arithmetic, Algebra, Domain-knowledge problems
Ape210K† [57]	210,488	-	-	$\{+,-,\times,\div\}$	56,532	Equation, Answer	Single equation, Arithmetic, Algebra
GSM8K [56]	8,500	-	-	$\{+,-,\times,\div\}$	-	Expression, Answer, Rationale	Single equation, Arithmetic
EW10K [90]	10,227	-	-	$\{+,-,\times,\div\}$	10	Equation, Answer	Single equation, Arithmetic
CM17K† [54]	17,035	-	-	$\{+,-,\times,\div\}$	-	Equation, Answer, Rationale, Examination Criteria, POS	Multi equation, Arithmetic, Domain-knowledge problems

Figure 2.6: Summary of the MWP datasets. Datasets marked with † are in the Mandarin Chinese language.

	AI2 (%)	IL (%)	CC (%)	SingleEQ (%)	AllArith (%)	ASP6.6K (%)	Dolphin-S (%)	MAWPS (%)	Math23K (%)	Math23K* (%)	Ape210K (%)
ARIS	77.7	-	-	48	-	-	-	-	-	-	-
Schema	88.64	-	-	-	-	-	-	-	-	-	-
Formula	86.07	-	-	-	-	-	-	-	-	-	-
LogicForm	84.8	80.01	53.5	-	-	-	-	-	-	-	-
ALGES	52.4	72.9	65	72	60.4	-	-	-	-	-	-
ExpressionTree	72	73.9	45.2	66.38	79.4	-	28.78	-	-	-	-
UNITDEP	56.2	71.0	53.5	72.25	81.7	-	28.78	-	-	-	-
MathDQN	78.5	73.3	75.5	52.96	72.68	-	30.06	60.25	-	-	-
Seq2SeqET	-	-	-	-	-	-	-	-	66.7	-	-
StackDecoder	-	-	-	-	-	-	-	-	-	65.8	52.3
DNS	-	-	-	-	-	-	-	59.5	-	58.1	-
Math-EN	-	-	-	-	-	67.8	-	69.2	66.7	-	-
T-RNN	-	-	-	-	-	-	39.1	66.8	66.9	-	-
S-Aligned	-	-	-	-	-	-	-	-	-	65.8	-
Group-ATT	-	-	-	-	-	67.4	-	76.1	69.5	66.9	-
AST-Dec	-	-	-	-	-	-	-	-	69.0	-	-
GTS	-	-	-	-	-	76.8	-	82.6	75.6	74.3	56.5
KA-S2T	-	-	-	-	-	-	-	-	76.3	-	-
IRE	-	-	-	-	-	-	-	-	76.7	-	-
Graph2Tree	-	-	-	-	-	76.8	-	83.7	77.4	75.5	-
Multi-E/D	-	-	-	-	-	-	-	-	78.4	76.9	-
MWP-BERT w/o MLM	-	-	-	-	-	-	-	-	83.8	82.0	-
MWP-BERT w MLM	-	-	-	-	-	-	-	-	84.4	82.3	84.3
SMART	-	-	-	-	-	79.5	-	-	-	-	-
REAL	-	-	-	-	-	-	-	-	82.3	80.8	77.2
RPKHS	-	-	-	-	-	-	-	89.8	83.9	82.2	-
Generate & Rank	-	-	-	-	-	-	-	84.0	85.4	84.3	-
RoBERTa-DeductReasoner	-	-	-	-	-	-	-	92.0	85.1	83.0	-

Figure 2.7: Performance Analysis of MWP solvers — Accuracies in the relevant datasets. Math23K* denotes 5-fold Cross Validation

Chapter 3

Proposed Methodology

3.1 System Architecture

Figure-3.3 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.

3.1.1 Paraphraser Model

The task of correctly reformulating a Math Word Problem statement requires a good level of language understanding. 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) [6] 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.

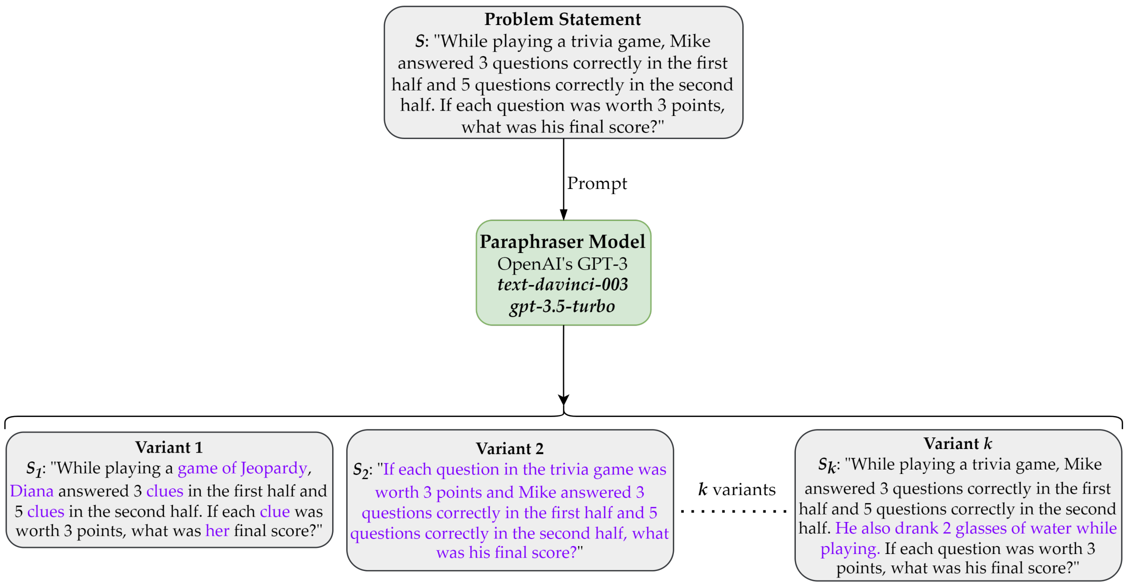

Problem Statement
S: "While playing a trivia game, Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half. If each question was worth 3 points, what was his final score?"
Prompt
Paraphraser Model
OpenAI's GPT-3
text-davinci-003
gpt-3.5-turbo
Variant 1
S1: "While playing a game of Jeopardy, Diana answered 3 clues in the first half and 5 clues in the second half. If each clue was worth 3 points, what was her final score?"
Variant 2
S2: "If each question in the trivia game was worth 3 points and Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half, what was his final score?"
k variants
Variant k
Sk: "While playing a trivia game, Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half. He also drank 2 glasses of water while playing. If each question was worth 3 points, what was his final score?"

Figure 3.1: Paraphraser Model (GPT-3).

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. A detailed discussion on the types of problem variations is delineated in Section-4. Figure-3.1 portrays the problem variant generation process.

3.1.1.1 Prompts and System Task Description

Prompts and system task descriptions play important roles in guiding the generation of text in large language models (LLMs) like GPT-3 [6]. By definition, prompts are initial instructions, questions, or partial sentences that serve as cues for generating text from a language model. They provide the starting point or context for the model to produce a coherent response or continuation. Prompts can be specific or open-ended, depending on the desired output. System task descriptions refer to detailed instructions or specifications provided to a language model regarding a specific task or objective it is expected to accomplish. These descriptions outline the desired input-output behavior, constraints, and requirements of the task. 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, $k = k_1 + k_2 + k_3$ is not fewer than 5 but not more than 15 i.e. $5 \leq k \leq 15$.

3.1.2 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. This 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.

3.1.3 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. [7] 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.

3.1.3.1 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.

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,

$$\begin{aligned} A_{ij} &= \langle H_i, P_{i|j} \rangle \times \langle H_j, P_{j|i} \rangle^{\top} \\ &= \underbrace{H_i H_j^{\top}}_{C2C} + \underbrace{H_i P_{j|i}^{\top}}_{C2P} + \underbrace{P_{i|j} H_j^{\top}}_{P2C} + \underbrace{P_{i|j} P_{j|i}^{\top}}_{\substack{P2P \\ (omitted)}} \end{aligned} \tag{3.1}$$

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 (3.1) 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. [84] 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. So, Self-attention can be trivialized as a *soft* dictionary lookup which returns a weighted sum of the values in the corpus. This weight represents the usefulness of a particular token in embedding/contextualizing the query token. If $H \in \mathbb{R}^{N \times d}$ represents the content and $P \in \mathbb{R}^{2k \times d}$ represents the positional information of a sentence with N tokens each having an embedding dimension of d, while k is half the size of the context window or the maximum relative distance, then the query, key, and values are calculated as,

$$\begin{aligned} &Q_c = HW_{c_Q}, K_c = HW_{c_K}, V_c = HW_{c_V} \\ &Q_r = PW_{r_Q}, K_r = PW_{r_K} \end{aligned} \tag{3.2}$$

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,

$$\delta(i,j) = \begin{cases} 0, & \text{if } i - j \leq k \\ 2k-1, & \text{if } i - j \geq k \\ i-j+k, & \text{otherwise} \end{cases} \tag{3.3}$$

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 (3.2) in the following manner,

$$\bar{A}_{ij} = \underbrace{Q_i^c K_j^{c\top}}_{C2C} + \underbrace{Q_i^c K_{\delta(i,j)}^{r\top}}_{C2P} + \underbrace{K_j^c Q_{\delta(j,i)}^{r\top}}_{P2C} \tag{3.4}$$

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,

$$H_{output} = \textbf{softmax}\left(\frac{\bar{A}}{\sqrt{3d}}\right) V_c \tag{3.5}$$

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 [7, 84].

3.1.4 Decoder

He et al. [7] postulates that the premature integration of absolute positions, which is employed by BERT [38] 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 pretrained 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 (see Figure-3.2) is dubbed an Enhanced Mask Decoder (EMD) and it demonstrably outperforms the decoder counterparts of its predecessor PLMs [7].

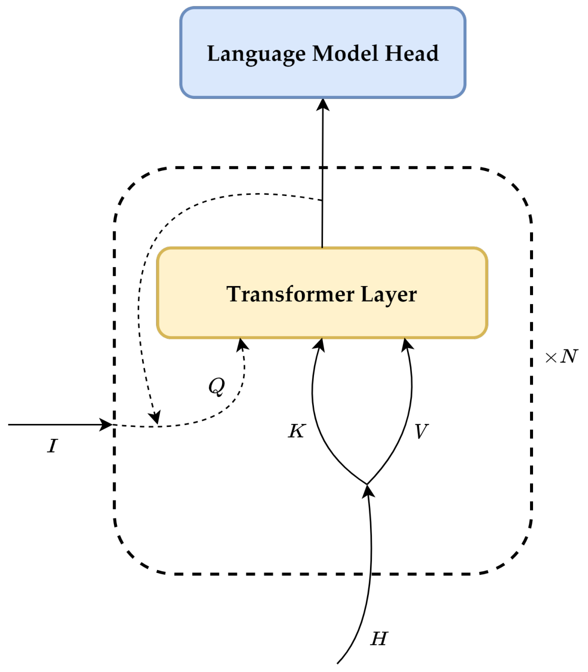


Figure 3.2: Enhanced Mask Decoder.

3.1.5 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`. 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 testing/validation phases or during inference.

$$E^* \leftarrow \underset{E'}{\operatorname{argmax}} \, \mathbf{Votes}(E'_i); \quad i = 1, 2, \ldots, k + 1 \tag{3.6}$$

Equation-(3.6) states that E^* is assigned the argument E' that maximizes the **Votes** function for each value of i from 1 to $k + 1$, where each i is the index of a generated expression E'_i.

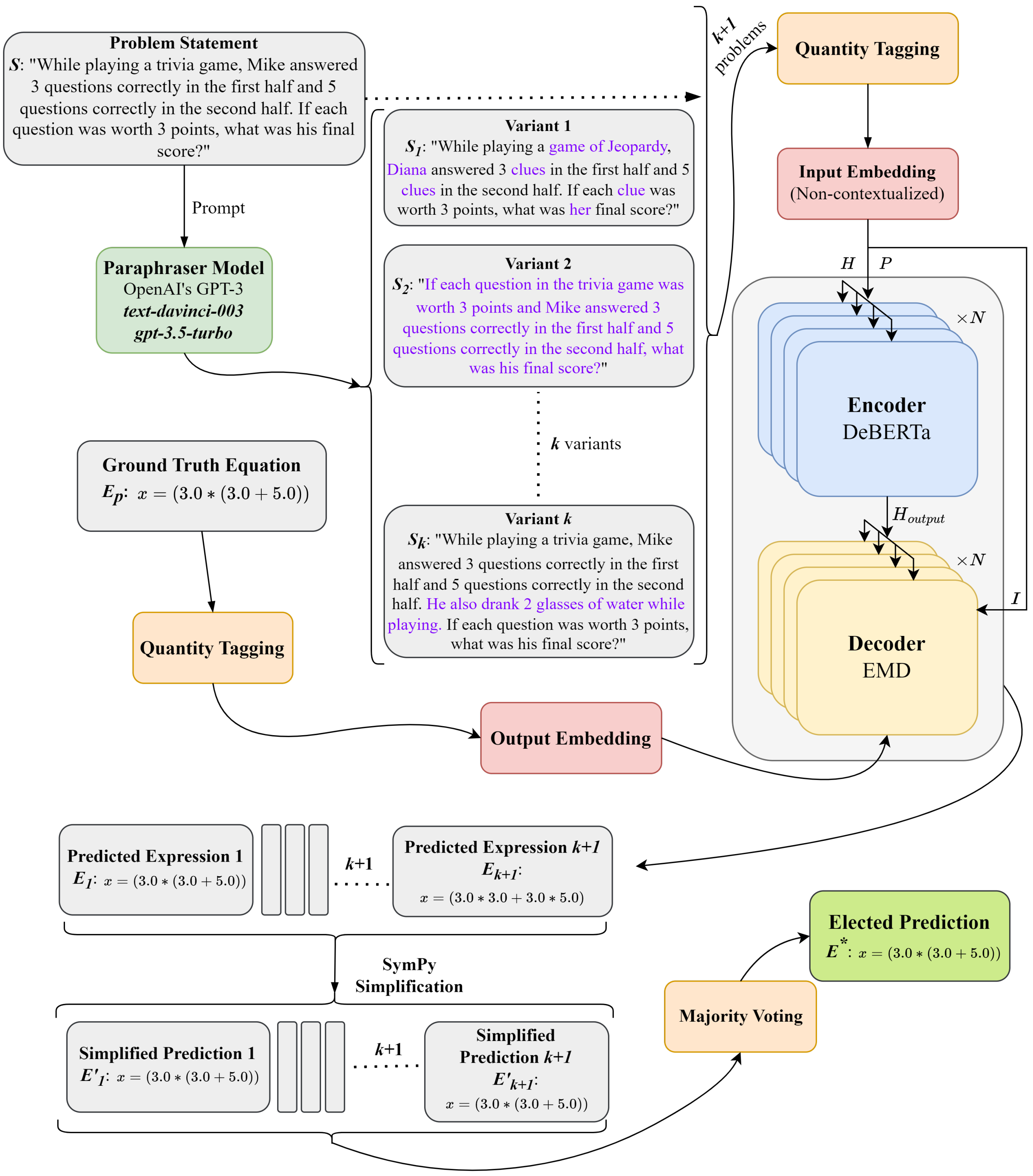

Problem Statement
S: "While playing a trivia game, Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half. If each question was worth 3 points, what was his final score?"
Prompt
Paraphraser Model
OpenAI's GPT-3
text-davinci-003
gpt-3.5-turbo
Variant 1
S1: "While playing a game of Jeopardy, Diana answered 3 clues in the first half and 5 clues in the second half. If each clue was worth 3 points, what was her final score?"
Variant 2
S2: "If each question in the trivia game was worth 3 points and Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half, what was his final score?"
k variants
Variant k
Sk: "While playing a trivia game, Mike answered 3 questions correctly in the first half and 5 questions correctly in the second half. He also drank 2 glasses of water while playing. If each question was worth 3 points, what was his final score?"
k+1 problems
Quantity Tagging
Input Embedding (Non-contextualized)
H
P
×N
Encoder DeBERTa
H_output
×N
I
Decoder EMD
Ground Truth Equation
Ep: x = (3.0 ∗ (3.0 + 5.0))
Quantity Tagging
Output Embedding
Predicted Expression 1
E1: x = (3.0 ∗ (3.0 + 5.0))
k+1
Predicted Expression k+1
Ek+1:
x = (3.0 ∗ 3.0 + 3.0 ∗ 5.0)
SymPy Simplification
Simplified Prediction 1
E'1: x = (3.0 ∗ (3.0 + 5.0))
k+1
Simplified Prediction k+1
E'k+1:
x = (3.0 ∗ (3.0 + 5.0))
Majority Voting
Elected Prediction
E*: x = (3.0 ∗ (3.0 + 5.0))

Figure 3.3: Overview of our proposed model.

Chapter 4

Experimental Setup

4.1 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 [1] 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. 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-4.1).

4.1.1 Linguistic Variations

4.1.1.1 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.

4.1.1.2 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 [104].

4.1.1.3 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 [105].

4.1.1.4 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. [106] used such problem samples in their work.

Table 4.1: 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.

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?

4.1.2 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 [31] and APE210K [46] consist of math problems written in Chinese Mandarin. We also aim to diversify the samples in MAWPS to enable better training for MWP solvers [107, 108]. SVAMP, created by Patel et al. [2] 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-4.2 and their operator count distributions are portrayed in Figure-4.1.

Table 4.2: Comparison of the datasets used.

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

4.2 Model Implementation Details and Training

4.2.1 Baseline Models

We implement the DeBERTa model using Microsoft's *deberta-base* that is publicly available in Hugging Face[1]. The other baseline MWP solver models are implementations already available in the open-source `MWPToolkit`[2] developed by Lan et al. [109]. We use an extensive set of baseline models, Transformer [84], DNS [31], MathEN [75], GroupATT [36], RNNEncDec [110], RNNVAE [111], BERT [38], RoBERTa [40], and compare them with the performance of the DeBERTa model.

4.2.2 Dataset Analysis

For the purposes of training and testing the models, we used the MAWPS dataset [1], the SVAMP dataset [2] and our dataset PARAMAWPS. The authors of the MAWPS dataset controlled 3 important data characteristics so that the dataset can aid in making models robust. These 3 characteristics are — Lexical Overlap, Template Overlap and Grammaticality. The reuse of lexemes among problem statements in a dataset is defined as Lexical Overlap and this characteristic is reduced to an extent in MAWPS. Some of the data samples willingly have small grammatical errors to emulate real-life scenarios. They also include morphological agreement failures, incorrect spellings, and other ungrammatical usage of the English language.

The pairwise lexical overlap of a dataset's problem statements p and q is defined as,

$$\text{PairLex}(p, q) = \frac{|W(p) \cap W(q)|}{|W(p) \cup W(q)|} \tag{4.1}$$

where, $W(x)$ is the set of unique unigrams along with bigrams in the problem statement x. Then, formally, the overall lexical overlap of a dataset D is delineated

[1]https://huggingface.co/microsoft/deberta-base

[2]https://github.com/LYH-YF/MWPToolkit

as,

$$\text{Lex}(D) = \frac{1}{N} \sum_{\substack{p_i, p_j \in D \\ i<j}} \text{PairLex}(p_i, p_j) \tag{4.2}$$

where, $N = {}^{|D|}C_2$ is the total number of possible combinations of problem statement pairs in the dataset. The authors constructed the MAWPS dataset by minimizing (4.2).

The template overlap of a dataset is pretty similar as well and the MAWPS dataset was created with an attempt to minimize this as well. The pairwise template overlap of a dataset's problem statements p and q is defined as,

$$\text{PairTempl}(p, q) = \begin{cases} 1, & \text{if } p \text{ and } q \text{ have the same template.} \\ 0, & \text{otherwise.} \end{cases} \tag{4.3}$$

Then, formally, the overall lexical dataset of a dataset D is delineated as,

$$\text{Tmpl}(D) = \frac{1}{N} \sum_{\substack{p_i, p_j \in D \\ i<j}} \text{PairTempl}(p_i, p_j) \tag{4.4}$$

where, $N = {}^{|D|}C_2$ is the total number of possible combinations of problem statement pairs in the dataset. Overall, the arithmetic mean of the values obtained in (4.2) and (4.4) is reduced and it is expressed as,

$$\text{H}(D) = \frac{1}{2}(\text{Lex}(D) + \text{Tmpl}(D)) \tag{4.5}$$

The eventual MAWPS dataset can then be referred to as D^*, where,

$$D^* \leftarrow \underset{D}{\text{argmin}}\, H(D) \tag{4.6}$$

4.2.3 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

Table 4.3: Characteristics of the Mawps Dataset[1].

Dataset	# Probs |D|	# Gramm.	Lexical Overlap (Lex)			Template Overlap (Tmpl)		
			k = |D|/2	k = |D|	Reduction	k = |D|/2	k = |D|/2	Reduction
AddSub	395	357	6.1	7.9	22.8	33	37.2	11.3
SingleOp	562	491	6.1	7.8	21.8	24.7	25.4	2.8
MultiArith	600	526	7.8	9.4	17.0	19.7	22.1	10.9
SingleEq	508	434	5.4	6.8	20.6	11	17.9	38.5
SimulEq-S	514	437	4.7	6	21.7	2.9	12.5	76.8
SimulEq-L	1155	980	4.4	5.7	22.8	0.1	3.3	97.0

Koncel-Kedziorski et al. [1]. 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 [103] dataset.

4.2.4 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.

4.2.5 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.

4.2.6 Optimizer

We use Adam [92] with a StepLR learning rate scheduler as our optimizer. The learning rate lr is set according to Vaswani et al. [84], $lr = d^{-0.5} \cdot \min(n^{-0.5}, n \cdot w^{-1.5})$ 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 $step\_size = 5$.

4.2.7 Hardware and Schedule

We have used the NVIDIA RTX 3090 GPU equipped with 25GB of VRAM and an 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 [7], it takes the longest time to train.

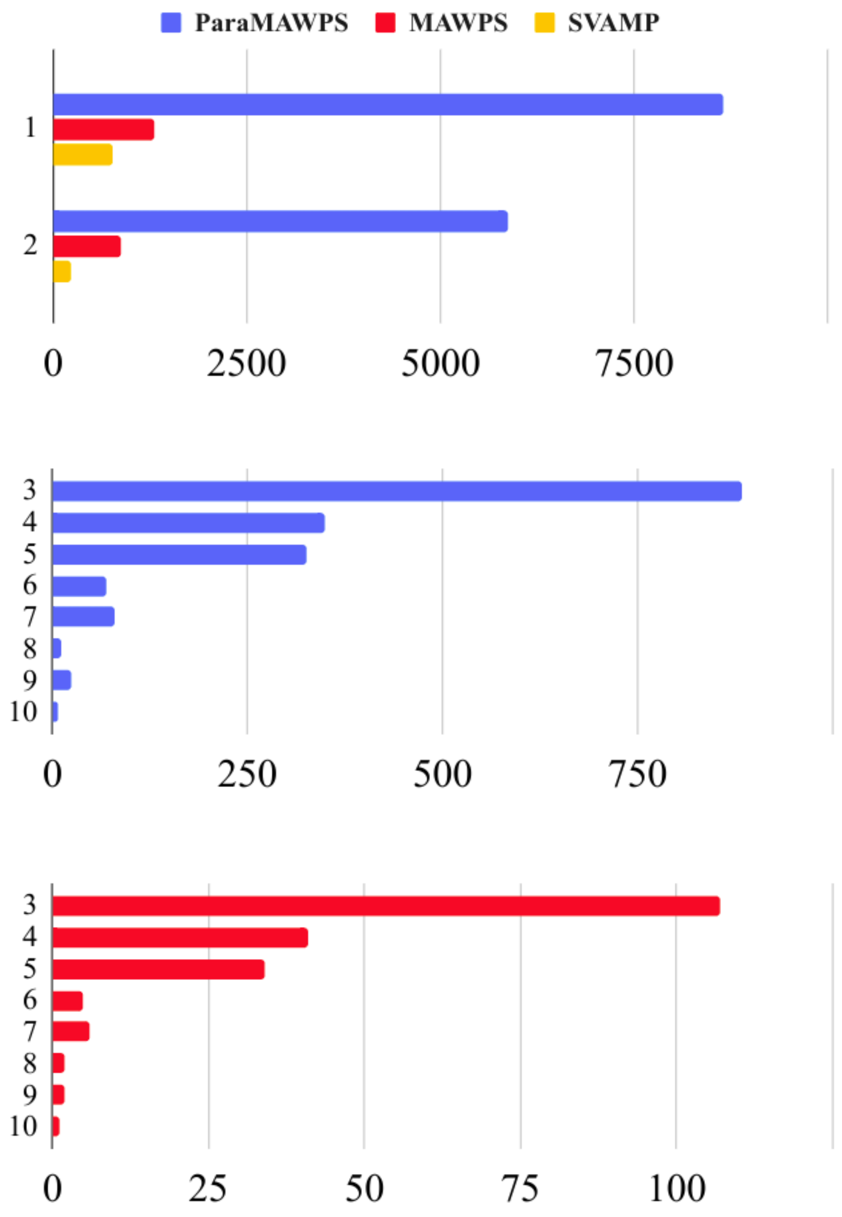


Figure 4.1: Operator count distributions of PARAMAWPS, MAWPS, and SVAMP. We keep the distribution of PARAMAWPS somewhat similar to that of MAWPS to maintain a proper balance between easy and difficult problems.

Chapter 5

Result Analysis and Discussion

5.1 Experimental Results

Table-5.1 shows the performance comparison of the DeBERTa model and the baseline models mentioned in Section-4.2.1. The DeBERTa model coupled with the Paraphrasing model and the Voting Mechanism outperforms all the baseline models in the MAWPS [1] dataset with an accuracy of 91.0%. The Paraphrasing Model and the Voting Mechanism contributed to a 0.3% increase in accuracy.

Table 5.1: 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.

Methods	MAWPS† (%)	SVAMP (%)	PARAMAWPS† (%)
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$_{\text{PM + VM}}$	**91.0**	-	-
DeBERTa$_{\text{VM}}$	-	-	**79.1**

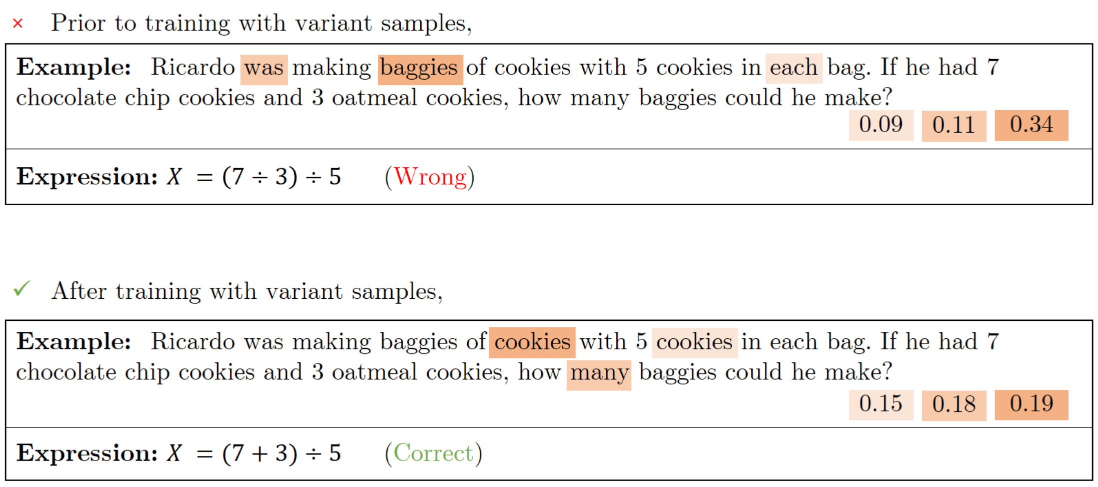
The vanilla DeBERTa model also outperforms the baseline models in our ParaMAWPS dataset by boasting an accuracy of 74.1%. With the voting mechanism at the tail-end of the pipeline, we are able to yield an improvement of the accuracy by 5.04% making the accuracy 79.1%. We test the robustness of the vanilla DeBERTa model on the Svamp [2] 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. [63] ($92.0 \pm 0.20\%$) 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.

5.2 Enhancing the Reasoning Proficiency

In order to determine the extent to which the model is improving in mathematical reasoning, we need to perform an "under-the-hood" analysis and determine if the model is paying attention to the desired keywords or not. Figure-5.1 shows a cherry-picked problem sample that was wrongly solved before any linguistic variants were introduced during training but the problem was correctly solved after the training process involved variant problems. In the first case, we notice that the model paid a significant amount of attention or gravitas to the wrong entity/object *"baggies"*. It also paid considerable attention to the word *"each"* which caused it to predict the division operator ($\div$) in the expression rendering it incorrect.

On the contrary, after being trained with linguistic variants of problem statements, the model assigned nearly equal and non-localized attention scores for all the tokens throughout the sentence. This translates to the fact that the model is now taking the whole sentence's context and meaning into consideration instead



× Prior to training with variant samples,

Example: Ricardo was making baggies of cookies with 5 cookies in each bag. If he had 7 chocolate chip cookies and 3 oatmeal cookies, how many baggies could he make?

0.09 0.11 0.34

Expression: $X = (7 \div 3) \div 5$ (Wrong)

✓ After training with variant samples,

Example: Ricardo was making baggies of cookies with 5 cookies in each bag. If he had 7 chocolate chip cookies and 3 oatmeal cookies, how many baggies could he make?

0.15 0.18 0.19

Expression: $X = (7 + 3) \div 5$ (Correct)

Figure 5.1: Answer and attention heatmap visualization of last decoder layer (top 3 values avg. across all heads) `[CLS]` and `[SEP]` tokens omitted.

of just looking for superficial keywords. We observe that the correct entity/object *"cookies"* now gets a significant attention score, and the keyword *"many"* incentivized the model to think that a total or aggregate of something is needed which is why it predicted the addition operator (+) in the expression rendering it correct.

5.3 Limitations of Existing Models

5.3.1 Understanding Irrelevant Information

In some cases, irrelevant information in the problem statement results in gibberish and erroneous outputs. Two examples, with irrelevant information in the problem statement, are shown in table 5.2 and 5.3, where one yields the wrong expression and another yields the correct expression respectively.

Table 5.2: Understanding Irrelevant Information

Problem	Nafis has 4 burgers. Rifat takes 2 burgers from him. How many burgers does Nafis have?	**Nafis and Rifat are giving a presentation.** Nafis has 4 burgers. Rifat takes 2 burgers from him. How many burgers does Nafis have?
Expression	$x = 4 - 2$	$x = 2 - 44$
Verdict	Correct (✓)	Wrong (×)

Table 5.3: Understanding Irrelevant Information

Problem	Rifat has 4 books. Nafis gifts him 2 more books. How many books does Rifat have?	**Nafis and Rifat are giving a presentation.** Rifat has 4 books. Nafis gifts him 2 more books. How many books does Rifat have?
Expression	$x = 2 + 4$	$x = 2 + 4$
Verdict	Correct (✓)	Correct (✓)

5.3.2 Understanding Grammar

Some models perform poorly when there are issues pertaining to grammatical correctness/errors in problem text. In Table-5.4, the first problem text contains grammatical errors and the model correctly gives the expression. But when the grammar is corrected, it results in a wrong expression output.

Table 5.4: Understanding Grammar

Problem	Donald had some apples. Hillary took 20 apples from him. Now Donald has 100 apples. How many apples Donald **had** before?	Donald had some apples. Hillary took 20 apples from him. Now Donald has 100 apples. How many apples **did** Donald **have** before?
Expression	$x = 100 + 20$	$x = 100 - 20$
Verdict	Correct (✓)	Wrong (×)

5.3.3 Understanding Numbers

Since every digit in numbers was tokenized separately, our transformer model cannot properly understand the concept of numbers. That's why by changing a single digit in a number can result in a completely different expression as shown in table 5.5. However, splitting the numbers into digits and tokenizing them separately was necessary because without it our model would need to learn every possible number differently. Separating them ensures our model learns only 10 digits.

Table 5.5: Understanding Numbers

Problem	Jerry had 135 pens. John took **19** pens from him. How many pens Jerry have left?	Jerry had 135 pens. John took **39** pens from him. How many pens Jerry have left?
Expression	$x = 135 - 19$	$x = 135 - 399$
Verdict	Correct (✓)	Wrong (×)

5.3.4 Understanding Problems Formulated Differently

The models sometimes fail to generate the correct equations of problems that are paraphrased or reworded versions of problems they can already solve. In the following Table 5.6, the outputs of the RoBERTaGen model is shown.

Table 5.6: Understanding Problems Formulated Differently

Problem	2 times the sum of 4 and some number is 34. What is the number?	Let X be a number. If 2 multiplied by the sum of 4 and X is equal to 34, what is X?
Expression	$x = 34.0/2.0 - 4.0$	$x = (2.0 + 4.0)$
Verdict	Correct (✓)	Wrong ($\times$)

5.3.5 Understanding Problems Requiring Constant Values

The models seem to falter at solving problems that require some basic unit conversion knowledge to solve, *e.g.*, 12 units make a dozen etc. Some problems also require the derivation of a constant from the number of entities or objects mentioned in the problem statement. These types of problems also seem to cause a lot of trouble for the models.

Table 5.7: Understanding Problems that Require Constant Values

Problem	Mary, Sam, Keith, and Alyssa each have 6 marbles. How many marbles do they have in all?	Sam saw 1 dozen baseballs on a shelf. How many baseballs did Sam see?
Expression	$x = (6.0 + NUM_1)$	$x = (1.0 + NUM_1)$
Verdict	Wrong ($\times$)	Wrong ($\times$)

5.3.6 Understanding Irrelevant Quantities

The models seem to perform badly at some of the adversarial samples that contain quantities not required to solve the problem.

Table 5.8: Understanding Irrelevant Quantities

Problem	Sally has 6 blue balloons. Fred has 3 times more blue balloons than Sally. How many blue balloons does Fred have now?	Sally has 6 blue balloons and her favorite number is 8. Fred has 3 times more blue balloons than Sally. How many blue balloons does Fred have now?
Expression	$x = 6.0 * 3.0$	$x = (6.0 + 8.0)$
Verdict	Correct (✓)	Wrong ($\times$)

5.3.7 Understanding Inverse Variants of Problems

The models show incompetence in solving the inverse version of problems they can already solve. This indicates a lack of backward reasoning ability of the models.

Table 5.9: Understanding Inverse Versions of Problems

Problem	Recently, the value of Kate's retirement fund decreased by \$12. If her fund was worth \$1472 before, how much is it worth now?	The value of Kate's retirement fund is now worth \$1460. How much was the value of Kate's retirement fund worth before it decreased by \$12?
Expression	$x = 1472.0 - 12.0$	$x =< unk > +12.0$
Verdict	Correct (✓)	Wrong (×)

There are many more types and categories of problems that can prove to be challenging for the models. Some examples of these types of problems have been discussed in Section 7.1. The dataset we introduce, PARAMAWPS, aims to be a robust testbed for MWP solvers by addressing these limitations. By presenting problem samples with a diverse range of mathematical concepts and language, this dataset would challenge the capabilities of the model in a nuanced and sophisticated manner. As such, it would provide a thorough assessment of the model's ability to process and solve complex mathematical problems, making it an invaluable tool for determining the model's overall robustness. Additionally, the use of sophisticated vocabulary in the MWPs would further test the model's language processing capabilities, providing a more complete evaluation of its performance.

5.3.8 MWP Task Performance Analysis of Large Language Models

To test out the assertion made in other studies [71, 72] about the incompetence of LLMs in complex reasoning tasks compared to fine-tuned smaller models, we used 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 [1] along with our dataset PARAMAWPS for testing the mathematical reasoning of these models. One of the most capable models in the GPT-3.5 series of

Table 5.10: Value accuracy of the LLMs in a zero-shot setup testing. † denotes evaluation on the whole dataset.

Models	Mawps† (%)	ParaMawps† (%)
GPT-J (6B)	9.9	5.9
text-babbage-001 (3B)	2.76	3.21
text-curie-001 (13B)	4.09	4.20
gpt-3.5-turbo (154B)	80.3	73.0

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 154 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 were 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* could output in the given format relatively consistently, *text-babbage-001* with 3 billion parameters could occasionally produce the output in the correct format, but tried to generate full sentences more often than not, whereas the 350 million parameters *text-ada-001* could 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 could be mostly mitigated by using very specific instructions for the prompt. The results in Table-5.10 and Table-5.1 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.

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-5.11 shows the effect of increasing the number of generated prob-

Table 5.11: Value accuracy with different numbers of linguistic variants of the problem samples. † denotes 5-fold cross validation.

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

Table 5.12: Effect of Majority Voting on Value accuracy across all 5 folds. † 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

lem 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-5.12 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.

Chapter 6

Conclusion and Future Works

6.1 Avenues of Improvement and Limitations

The research work outlined in this paper still has a lot of limitations. 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. Even though almost all the samples in the dataset that we introduced, namely ParaMAWPS, have been manually checked for correctness by 3 undergraduate students, we can't yet guarantee a foolproof review of it. Another limitation of our work is the absence of a proper tie-breaking strategy in our Majority Voting module. Furthermore, we can introduce a system of weighted votes (*e.g.* semantic similarity scores such as BERTScore [112], as weights) so that the votes of wrongly predicted equations don't trump that of correctly generated predictions as another frontier of improvement. We also plan to incorporate and experiment with the Tree-based decoder [37] in our proposed pipeline.

6.2 Summary of Our Contributions

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. 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.

6.3 Future Endeavors

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.

6.4 Epilogue

As we pen down these final words, we are filled with a profound sense of gratitude and accomplishment. This thesis, a culmination of our efforts, is a testament to a momentous intellectual odyssey spanning over 2 years of our undergraduate lives. We hope our efforts are deemed a noteworthy and meaningful contribution to this domain of research.

Bibliography

[1] R. Koncel-Kedziorski, S. Roy, A. Amini, N. Kushman, and H. Hajishirzi, "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*, 2016, pp. 1152–1157.

[2] A. Patel, S. Bhattamishra, and N. Goyal, "Are nlp models really able to solve simple math word problems?" *arXiv preprint arXiv:2103.07191*, 2021.

[3] J. Piaget, *Child's Conception of Number: Selected Works vol 2*. Routledge, 2013.

[4] J. Peterson, R. Pihl, D. Higgins, J. Séguin, and R. Tremblay, "Neuropsychological performance, iq, personality, and grades in a longitudinal grade-school male sample," *Individual Differences Research*, vol. 1, pp. 159–172, 12 2003.

[5] S. Kingsdorf and J. Krawec, "A broad look at the literature on math word problem-solving interventions for third graders," *Cogent Education*, vol. 3, no. 1, p. 1135770, 2016. [Online]. Available: https://doi.org/10.1080/2331186X.2015.1135770

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[7] P. He, X. Liu, J. Gao, and W. Chen, "Deberta: Decoding-enhanced bert with disentangled attention," *arXiv preprint arXiv:2006.03654*, 2020.

[8] R. Penrose, *Shadows of the Mind.* Oxford University Press Oxford, 1994, vol. 4.

[9] L. d. M. Daniel Selsam, "Imo grand challenge," https://imo-grand-challenge.github.io/.

[10] E. A. Feigenbaum, J. Feldman *et al.*, *Computers and thought.* New York McGraw-Hill, 1963.

[11] D. G. Bobrow, "Natural language input for a computer problem solving system," 1964.

[12] C. R. Fletcher, "Understanding and solving arithmetic word problems: A computer simulation," *Behavior Research Methods, Instruments, & Computers*, vol. 17, no. 5, pp. 565–571, 1985.

[13] Y. Bakman, "Robust understanding of word problems with extraneous information," *arXiv preprint math/0701393*, 2007.

[14] M. Yuhui, Z. Ying, C. Guangzuo, R. Yun, and H. Ronghuai, "Frame-based calculus of solving arithmetic multi-step addition and subtraction word problems," in *2010 Second International Workshop on Education Technology and Computer Science*, vol. 2. IEEE, 2010, pp. 476–479.

[15] N. Kushman, Y. Artzi, L. Zettlemoyer, and R. Barzilay, "Learning to automatically solve algebra word problems," in *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2014, pp. 271–281.

[16] M. J. Hosseini, H. Hajishirzi, O. Etzioni, and N. Kushman, "Learning to solve arithmetic word problems with verb categorization." in *EMNLP*, vol. 523533. Citeseer, 2014.

[17] S. Roy, T. Vieira, and D. Roth, "Reasoning about quantities in natural language," *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 1–13, 2015.

[18] L. Zhou, S. Dai, and L. Chen, "Learn to solve algebra word problems using quadratic programming," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 817–822.

[19] A. Mitra and C. Baral, "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)*, 2016, pp. 2144–2153.

[20] C.-C. Liang, K.-Y. Hsu, C.-T. Huang, C.-M. Li, S.-Y. Miao, and K.-Y. Su, "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*, 2016, pp. 67–71.

[21] C.-C. Liang, S.-H. Tsai, T.-Y. Chang, Y.-C. Lin, and K.-Y. Su, "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.* Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 151–155. [Online]. Available: https://aclanthology.org/C16-2032

[22] R. Koncel-Kedziorski, H. Hajishirzi, A. Sabharwal, O. Etzioni, and S. D. Ang, "Parsing algebraic word problems into equations," *Transactions of the Association for Computational Linguistics*, vol. 3, pp. 585–597, 2015.

[23] S. Roy and D. Roth, "Solving general arithmetic word problems," *arXiv preprint arXiv:1608.01413*, 2016.

[24] S. Roy, S. Upadhyay, and D. Roth, "Equation parsing: Mapping sentences to grounded equations," *arXiv preprint arXiv:1609.08824*, 2016.

[25] S. Roy and D. Roth, "Unit dependency graph and its application to arithmetic word problem solving," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, no. 1, 2017.

[26] S. Shi, Y. Wang, C.-Y. Lin, X. Liu, and Y. Rui, "Automatically solving number word problems by semantic parsing and reasoning," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015, pp. 1132–1142.

[27] Y. Zou and W. Lu, "Text2math: End-to-end parsing text into math expressions," *arXiv preprint arXiv:1910.06571*, 2019.

[28] D. Huang, S. Shi, C.-Y. Lin, J. Yin, and W.-Y. Ma, "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)*, 2016, pp. 887–896.

[29] S. Upadhyay, M.-W. Chang, K.-W. Chang, and W.-t. Yih, "Learning from explicit and implicit supervision jointly for algebra word problems," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 2016, pp. 297–306.

[30] D. Huang, S. Shi, C.-Y. Lin, and J. Yin, "Learning fine-grained expressions to solve math word problems," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 805–814.

[31] Y. Wang, X. Liu, and S. Shi, "Deep neural solver for math word problems," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 845–854.

[32] L. Wang, D. Zhang, L. Gao, J. Song, L. Guo, and H. T. Shen, "Mathdqn: Solving arithmetic word problems via deep reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[33] D. Huang, J. Liu, C.-Y. Lin, and J. Yin, "Neural math word problem solver with reinforcement learning," in *Proceedings of the 27th International Conference on Computational Linguistics*, 2018, pp. 213–223.

[34] P. Mishra, L. J. Kurisinkel, D. M. Sharma, and V. Varma, "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*, 2018.

[35] L. Wang, D. Zhang, J. Zhang, X. Xu, L. Gao, B. T. Dai, and H. T. Shen, "Template-based math word problem solvers with recursive neural networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 7144–7151.

[36] J. Li, L. Wang, J. Zhang, Y. Wang, B. T. Dai, and D. Zhang, "Modeling intra-relation in math word problems with different functional multi-head attentions," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 6162–6167.

[37] Z. Xie and S. Sun, "A goal-driven tree-structured neural model for math word problems." in *IJCAI*, 2019, pp. 5299–5305.

[38] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[39] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[40] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pre-training approach," *arXiv preprint arXiv:1907.11692*, 2019.

[41] Q. Liu, W. Guan, S. Li, and D. Kawahara, "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)*, 2019, pp. 2370–2379.

[42] Y. Shen and C. Jin, "Solving math word problems with multi-encoders and multi-decoders," in *Proceedings of the 28th International Conference on Computational Linguistics*, 2020, pp. 2924–2934.

[43] Q. Wu, Q. Zhang, J. Fu, and X.-J. Huang, "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)*, 2020, pp. 7137–7146.

[44] X. Lin, Z. Huang, H. Zhao, E. Chen, Q. Liu, H. Wang, and S. Wang, "Hms: A hierarchical solver with dependency-enhanced understanding for math word problem," in *Thirty-Fifth AAAI Conference on Artificial 2021*, 2021, pp. 4232–4240.

[45] J. Shen, Y. Yin, L. Li, L. Shang, X. Jiang, M. Zhang, and Q. Liu, "Generate & rank: A multi-task framework for math word problems," *arXiv preprint arXiv:2109.03034*, 2021.

[46] Z. Liang, J. Zhang, J. Shao, and X. Zhang, "Mwp-bert: A strong baseline for math word problems," *arXiv preprint arXiv:2107.13435*, 2021.

[47] Z. Liang, J. Zhang, L. Wang, W. Qin, J. Shao, and X. Zhang, "Mwp-bert: A numeracy-augmented pre-trained encoder for math word problems."

[48] Z. Li, W. Zhang, C. Yan, Q. Zhou, C. Li, H. Liu, and Y. Cao, "Seeking patterns, not just memorizing procedures: Contrastive learning for solving math word problems," *arXiv preprint arXiv:2110.08464*, 2021.

[49] J. Xiong, Z. Wan, X. Hu, M. Yang, and C. Li, "Self-consistent reasoning for solving math word problems," *arXiv preprint arXiv:2210.15373*, 2022.

[50] Y. Cao, F. Hong, H. Li, and P. Luo, "A bottom-up dag structure extraction model for math word problems," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 1, 2021, pp. 39–46.

[51] J. Zhang, R. K.-W. Lee, E.-P. Lim, W. Qin, L. Wang, J. Shao, and Q. Sun, "Teacher-student networks with multiple decoders for solving math word problem." IJCAI, 2020.

[52] G. Hinton, O. Vinyals, J. Dean *et al.*, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, vol. 2, no. 7, 2015.

[53] W. Yu, Y. Wen, F. Zheng, and N. Xiao, "Improving math word problems with pre-trained knowledge and hierarchical reasoning," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 3384–3394.

[54] Y. Hong, Q. Li, D. Ciao, S. Huang, and S.-C. Zhu, "Learning by fixing: Solving math word problems with weak supervision," in *AAAI Conference on Artificial Intelligence*, 2021.

[55] J. Qin, X. Liang, Y. Hong, J. Tang, and L. Lin, "Neural-symbolic solver for math word problems with auxiliary tasks," *arXiv preprint arXiv:2107.01431*, 2021.

[56] S. Huang, J. Wang, J. Xu, D. Cao, and M. Yang, "Recall and learn: A memory-augmented solver for math word problems," *arXiv preprint arXiv:2109.13112*, 2021.

[57] J. Zhang, L. Wang, R. K.-W. Lee, Y. Bin, Y. Wang, J. Shao, and E.-P. Lim, "Graph-to-tree learning for solving math word problems." Association for Computational Linguistics, 2020.

[58] S. Li, L. Wu, S. Feng, F. Xu, F. Xu, and S. Zhong, "Graph-to-tree neural networks for learning structured input-output translation with applications to semantic parsing and math word problem," *arXiv preprint arXiv:2004.13781*, 2020.

[59] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," *Advances in neural information processing systems*, vol. 32, 2019.

[60] D. Cai and W. Lam, "Graph transformer for graph-to-sequence learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 05, 2020, pp. 7464–7471.

[61] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[62] O. Chatterjee, A. Waikar, V. Kumar, G. Ramakrishnan, and K. Arya, "A weakly supervised model for solving math word problems," *arXiv preprint arXiv:2104.06722*, 2021.

[63] Z. Jie, J. Li, and W. Lu, "Learning to reason deductively: Math word problem solving as complex relation extraction," *arXiv preprint arXiv:2203.10316*, 2022.

[64] Z. Shao, F. Huang, and M. Huang, "Chaining simultaneous thoughts for numerical reasoning," *arXiv preprint arXiv:2211.16482*, 2022.

[65] Y. Li, Z. Lin, S. Zhang, Q. Fu, B. Chen, J.-G. Lou, and W. Chen, "On the advance of making language models better reasoners," *arXiv preprint arXiv:2206.02336*, 2022.

[66] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, and D. Zhou, "Self-consistency improves chain of thought reasoning in language models," *arXiv preprint arXiv:2203.11171*, 2022.

[67] X. Pi, Q. Liu, B. Chen, M. Ziyadi, Z. Lin, Y. Gao, Q. Fu, J.-G. Lou, and W. Chen, "Reasoning like program executors," *arXiv preprint arXiv:2201.11473*, 2022.

[68] W. Chen, X. Ma, X. Wang, and W. W. Cohen, "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks," *arXiv preprint arXiv:2211.12588*, 2022.

[69] K. Cobbe, V. Kosaraju, M. Bavarian, J. Hilton, R. Nakano, C. Hesse, and J. Schulman, "Training verifiers to solve math word problems," *arXiv preprint arXiv:2110.14168*, 2021.

[70] J. Yang, H. Jin, R. Tang, X. Han, Q. Feng, H. Jiang, B. Yin, and X. Hu, "Harnessing the power of llms in practice: A survey on chatgpt and beyond," *arXiv preprint arXiv:2304.13712*, 2023.

[71] J. Huang and K. C.-C. Chang, "Towards reasoning in large language models: A survey," *arXiv preprint arXiv:2212.10403*, 2022.

[72] N. Ho, L. Schmid, and S.-Y. Yun, "Large language models are reasoning teachers," *arXiv preprint arXiv:2212.10071*, 2022.

[73] J. Wei, X. Wang, D. Schuurmans, M. Bosma, E. Chi, Q. Le, and D. Zhou, "Chain of thought prompting elicits reasoning in large language models," *arXiv preprint arXiv:2201.11903*, 2022.

[74] S. Roy and D. Roth, "Illinois math solver: Math reasoning on the web," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, 2016, pp. 52–56.

[75] L. Wang, Y. Wang, D. Cai, D. Zhang, and X. Liu, "Translating a math word problem to an expression tree," *arXiv preprint arXiv:1811.05632*, 2018.

[76] T.-R. Chiang and Y.-N. Chen, "Semantically-aligned equation generation for solving and reasoning math word problems," *arXiv preprint arXiv:1811.00720*, 2018.

[77] Y. Meng and A. Rumshisky, "Solving math word problems with double-decoder transformer," *arXiv preprint arXiv:1908.10924*, 2019.

[78] Q. Liu, W. Guan, S. Li, F. Cheng, D. Kawahara, and S. Kurohashi, "Reverse operation based data augmentation for solving math word problems," *arXiv preprint arXiv:2010.01556*, 2020.

[79] Y. Hong, Q. Li, R. Gong, D. Ciao, S. Huang, and S.-C. Zhu, "Smart: A situation model for algebra story problems via attributed grammar," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 14, 2021, pp. 13 009–13 017.

[80] B. Kim, K. S. Ki, D. Lee, and G. Gweon, "Point to the expression: Solving algebraic word problems using the expression-pointer transformer model," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2020, pp. 3768–3779.

[81] A. Mukherjee and U. Garain, "A review of methods for automatic understanding of natural language mathematical problems," *Artificial Intelligence Review*, vol. 29, no. 2, pp. 93–122, 2008.

[82] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[83] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin, "Convolutional sequence to sequence learning," in *International Conference on Machine Learning.* PMLR, 2017, pp. 1243–1252.

[84] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[85] N. Chomsky, "Three models for the description of language," *IRE Transactions on information theory*, vol. 2, no. 3, pp. 113–124, 1956.

[86] D. Gildea and D. Jurafsky, "Automatic labeling of semantic roles," *Computational linguistics*, vol. 28, no. 3, pp. 245–288, 2002.

[87] M. Collins, "Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms," in *Proceedings of*

the 2002 conference on empirical methods in natural language processing (EMNLP 2002), 2002, pp. 1–8.

[88] R. Herbrich, "Large margin rank boundaries for ordinal regression," *Advances in large margin classifiers*, pp. 115–132, 2000.

[89] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[90] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[91] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[92] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[93] W. Kintsch and J. G. Greeno, "Understanding and solving word arithmetic problems." *Psychological review*, vol. 92, no. 1, p. 109, 1985.

[94] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[95] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in neural information processing systems*, vol. 30, 2017.

[96] R. Geirhos, J.-H. Jacobsen, C. Michaelis, R. Zemel, W. Brendel, M. Bethge, and F. A. Wichmann, "Shortcut learning in deep neural networks," *Nature Machine Intelligence*, vol. 2, no. 11, pp. 665–673, 2020.

[97] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018.

[98] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[99] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu *et al.*, "Exploring the limits of transfer learning with a unified text-to-text transformer." *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.

[100] W. Ling, D. Yogatama, C. Dyer, and P. Blunsom, "Program induction by rationale generation: Learning to solve and explain algebraic word problems," *arXiv preprint arXiv:1705.04146*, 2017.

[101] A. Amini, S. Gabriel, P. Lin, R. Koncel-Kedziorski, Y. Choi, and H. Hajishirzi, "Mathqa: Towards interpretable math word problem solving with operation-based formalisms," *arXiv preprint arXiv:1905.13319*, 2019.

[102] J. Qin, L. Lin, X. Liang, R. Zhang, and L. Lin, "Semantically-aligned universal tree-structured solver for math word problems," *arXiv preprint arXiv:2010.06823*, 2020.

[103] S.-Y. Miao, C.-C. Liang, and K.-Y. Su, "A diverse corpus for evaluating and developing english math word problem solvers," *arXiv preprint arXiv:2106.15772*, 2021.

[104] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini, "Deduplicating training data makes language models better," *arXiv preprint arXiv:2107.06499*, 2021.

[105] V. Kumar, R. Maheshwary, and V. Pudi, "Adversarial examples for evaluating math word problem solvers," *arXiv preprint arXiv:2109.05925*, 2021.

[106] Q. Liu, W. Guan, S. Li, F. Cheng, D. Kawahara, and S. Kurohashi, "Roda: reverse operation based data augmentation for solving math word problems," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 30, pp. 1–11, 2021.

[107] T. Schick and H. Schütze, "Generating datasets with pretrained language models," *arXiv preprint arXiv:2104.07540*, 2021.

[108] V. Kumar, R. Maheshwary, and V. Pudi, "Practice makes a solver perfect: Data augmentation for math word problem solvers," *arXiv preprint arXiv:2205.00177*, 2022.

[109] Y. Lan, L. Wang, Q. Zhang, Y. Lan, B. T. Dai, Y. Wang, D. Zhang, and E.-P. Lim, "Mwptoolkit: An open-source framework for deep learning-based math word problem solvers," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 11, 2022, pp. 13 188–13 190.

[110] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *Advances in neural information processing systems*, vol. 27, 2014.

[111] J. Su, S. Wu, D. Xiong, Y. Lu, X. Han, and B. Zhang, "Variational recurrent neural machine translation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[112] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.