

https://dspace.org/

MultiI-View Ranking: Tasking Transformers to Generate and Validate Solutions to Math Word Problems

Mzimba, Rifumo

2023-11
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/41971
Mzimba, Rifumo. (2023). MultiI-View Ranking: Tasking Transformers to Generate and Validate Solutions to Math Word Problems. [Master's dissertation, University of the Witwatersrand, Johannesburg]. https://hdl.handle.net/10539/41971

Downloaded from a DSpace repository. https://dspace.org/

Multi-View Ranking: Tasking Transformers to Generate and Validate Solutions to Math Word Problems

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Rifumo Mzimba
1619542

Supervised by
Prof. Richard Klein
Prof. Benjamin Rosman

October 29, 2023



A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, South Africa, in fulfillment of the requirements for the degree of Master of Science.

Abstract

The recent developments and success of the Transformer model have resulted in the creation of massive language models that have led to significant improvements in the comprehension of natural language. When fine-tuned for downstream natural language processing tasks with limited data, they achieve state-of-the-art performance. However, these robust models lack the ability to reason mathematically. It has been demonstrated that, when fine-tuned on the small-scale Math Word Problems (MWPs) benchmark datasets, these models are not able to generalize.

Therefore, to overcome this limitation, this study proposes to augment the generative objective used in the MWP task with complementary objectives that can assist the model in reasoning more deeply about the MWP task. Specifically, we propose a multi-view generation objective that allows the model to understand the generative task as an abstract syntax tree traversal beyond the sequential generation task.

In addition, we propose a complementary verification objective to enable the model to develop heuristics that can distinguish between correct and incorrect solutions. These two goals comprise our multi-view ranking (MVR) framework, in which the model is tasked to generate the prefix, infix, and postfix traversals for a given MWP, and then use the verification task to rank the generated expressions.

Our experiments show that the verification objective is more effective at choosing the best expression than the widely used beam search. We further show that when our two objectives are used in conjunction, they can effectively guide our model to learn robust heuristics for the MWP task. In particular, we achieve an absolute percentage improvement of 9.7% and 5.3% over our baseline and the state-of-the-art models on the SVAMP datasets. Our source code can be found on https://github.com/ProxJ/msc-final.

Declaration

I, Rifumo Mzimba, hereby declare the contents of this research proposal to be my own work. This proposal is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university or for any other degree.



Signature

30/10/2023

Date

Acknowledgements

Completing this dissertation has been one of the most challenging and rewarding experiences of my academic career. To everyone who helped me along the way, I want to express my sincere gratitude and appreciation.

To begin, I'd like to express my gratitude to my advisors, Profs. Richard Klein and Benjamin Rosman, for their expertise, patience, and direction throughout my research. Their suggestions and criticisms helped me shape and improve my ideas, and their unflinching encouragement kept me going.

I would also like to thank my colleagues and friends who were there for me when I really needed their encouragement. A particular shout out to my professional colleagues who took on extra responsibilities to allow me to focus on writing this dissertation.

I owe a great debt of gratitude to my family for their unending love, encouragement, and patience throughout this process. I couldn't have gotten through this period without their unwavering love, encouragement, and support.

I would also like to extend my gratitude to the rest of the academic staff who probed interesting questions and made interesting suggestions, the MSL and team for providing computational resources, and the sponsors who funded my studies.

I would like to acknowledge the difficulties I faced during the research process. With ambitious plans for radical advances, many ideas were tried but proved fruitless. This experience has given me a newfound respect for academic research and taught me the value of tenacity, persistence, and asking for assistance when one is stuck.

As a final note, I'd like to express my appreciation to the academic community at large for the work they've done in the field. I have been able to shape my research and hone my arguments thanks to the plethora of literature and research I have had at my disposal.

In conclusion, I am immensely grateful to everyone who helped me along the way. Your support, encouragement, and insights have been invaluable, and I am truly humbled by your generosity and kindness.

To God be the glory for giving me the fortitude, patience, and perseverance to finish this study.

Contents

Preface
Abstract i
Declaration ii
Acknowledgements iii
Table of Contents iv
List of Figures vii
List of Tables viii

1 Introduction **1**

2 Objective **7**
2.1 Research Objective 7
2.2 Research Hypothesis 7
2.3 Research Questions 8
2.3.1 Research Questions for the Multi-View Generation Task 8
2.3.2 Research Questions for the Verification Task 8

3 Background and Related Work **9**
3.1 Introduction 9
3.2 Transformers 9
3.2.1 Background (Recurrent Neural Networks) 10
3.2.2 Architecture Overview 12
3.2.3 Input Embedding 12
3.2.4 Positional Encoding 14
3.2.5 Attention Mechanism 14
3.2.6 Position-wise Feedforward Neural Network (FNN) 17
3.2.7 LayerNorm and Residual Connections 17
3.2.8 Final layer 18
3.3 Metric Learning 18
3.3.1 Distance Metric 19
3.3.2 Losses 20
3.3.3 Miners 23
3.4 Pooling 27
3.4.1 Single Token pooling 27
3.4.2 Max-pooling 28

3.4.3 Average Pooling . . . 28
3.4.4 Attentive Pooling . . . 28
3.4.5 Mixed Pooling . . . 29
3.5 Natural Language Processing . . . 29
3.5.1 Tokenization . . . 30
3.5.2 Sequential Generation Task . . . 32
3.6 Learning Paradigms . . . 34
3.6.1 Transfer Learning . . . 35
3.6.2 Multi-task Learning . . . 35
3.6.3 Multi-view Learning . . . 36
3.6.4 Multi-Model Learning . . . 36
3.7 Math Word Problems . . . 38
3.7.1 Mathematics Laws . . . 38
3.7.2 Math Notations . . . 39
3.8 Related Work . . . 40
3.8.1 Encoder Architectures . . . 41
3.8.2 Decoder Architectures . . . 42
3.9 Conclusion . . . 44

4 Data Representation and Multi-view Learning 45
4.1 Introduction . . . 45
4.2 Related Work . . . 47
4.2.1 Mathematics Expression Notation . . . 49
4.2.2 Conditional Generation . . . 50
4.2.3 Quantity Masking . . . 50
4.2.4 Chain of Thought . . . 51
4.2.5 Generation Consistency . . . 51
4.3 Methodology . . . 51
4.3.1 Problem Statement . . . 51
4.3.2 Model Architecture . . . 52
4.3.3 Quantity Masking . . . 53
4.3.4 Solution Notation . . . 54
4.3.5 Generation Consistency . . . 56
4.3.6 Training . . . 56
4.4 Experiments . . . 58
4.4.1 Datasets . . . 58
4.4.2 Baseline . . . 58
4.4.3 Transformers . . . 58
4.4.4 Training Details . . . 59
4.4.5 Evaluation Metrics . . . 59
4.4.6 Overall Results . . . 60
4.4.7 Expression Notation . . . 63
4.4.8 Multi-view Notation . . . 65
4.5 Conclusion and Future Work . . . 65
4.5.1 Future Direction . . . 66

5 Multi-tasking Multi-view Decoders **69**
5.1 Introduction . . . 69
5.2 Related Work . . . 70
5.2.1 Multi-tasking Models . . . 70
5.2.2 Multi-Decoder Models . . . 71
5.2.3 Verifiers . . . 71
5.3 Methodology . . . 72
5.3.1 Multi-view Generation . . . 73
5.3.2 Verification . . . 75
5.4 Experiments . . . 81
5.4.1 Datasets . . . 81
5.4.2 Baseline . . . 81
5.4.3 Experimental Setup . . . 83
5.4.4 Overall Results . . . 83
5.4.5 Ablation . . . 85
5.4.6 Multi-view and Verification . . . 87
5.4.7 Qualitative Analysis . . . 88
5.5 Conclusion . . . 90
5.5.1 Future Work . . . 90

6 Conclusion **92**

References **119**

List of Figures

1.1 Many-to-many mapping between questions and solutions. 2
1.2 Impact of minor changes on questions. 2

3.1 Transformer architecture. 13
3.2 Transformer - multi-head attention architecture 17
3.3 An example of expression trees 40

4.1 An example of the proposed masking technique. 47
4.2 Examples of different quantity masking techniques. 54
4.3 The proposed token-level view-consistency strategy. 57
4.4 The proposed multi-view BART model. 57
4.5 Divergence of context representations during training 64

5.1 The proposed multi-view ranking (MVR) model. 72
5.2 Examples of expression tree templates 78

List of Tables

4.1 Notations and quantity masks used to solve math word problems. 48
4.2 Dataset statistics. 58
4.3 Overall results of the multi-view model compared to baseline models. . . 63
4.4 Performance of each view in the multi-view setup. 64
4.5 Impact of using a different number of views and view combinations. . . 65

5.1 Overall results of our Multi-view Ranker (MVR). 84
5.2 Ablation results for different model setups. 87
5.3 Model performance under different learning objectives. 88
5.4 Qualitative analysis of the models 89

Chapter 1

Introduction

Solving mathematical word problems is one of the fundamental math concepts taught in elementary school. Math Word Problems (MWP) are mathematical questions that are primarily expressed in everyday language rather than using mathematical symbols. They frequently contain a scenario and a query related to the mathematical relationship between the quantities or objects in the scenario. One needs cognition, language comprehension, working memory, and attention to solve this [Lin 2021; Verschaffel *et al.* 2020; Kula 2007].

MWPs are an interesting benchmark for testing the reasoning ability of language models, similar to how humans use them [Scheibling-Sève *et al.* 2020]. In particular, MWP can prove useful in testing pre-trained language models that have demonstrated similar language comprehension as humans [Scheibling-Sève *et al.* 2020; Cobbe *et al.* 2021; Wei *et al.* 2022; He *et al.* 2020b].

The resolution of MWPs is a significant step towards the development of general intelligence [Fletcher 1985; Wei *et al.* 2022]. Furthermore, these models can be incorporated into virtual assistants, search engines, and Computer Algebra Systems to make these systems more robust, and to help students develop problem-solving skills [Scheibling-Sève *et al.* 2020; Morton and Qu 2013; Jeschke and Richter 2007; Kapralov *et al.* 2020; Zhang *et al.* 2020a].

The ability of language models to solve symbolic math problems has been shown by numerous researchers, demonstrating their aptitude for mathematics [Lample and Charton 2019; Saxton *et al.* 2019; Noorbakhsh *et al.* 2021]. Symbolic math problems involve manipulating mathematical expressions using symbolic notation to find a solution. These are often used in advanced mathematics such as calculus, algebra, and differential equations. While MWPs require simpler arithmetic operations, they remain an open problem [Patel *et al.* 2021; Zhang *et al.* 2020a]. This indicates that the challenge MWPs present is textual comprehension rather than purely mathematical [Morton and Qu 2013].

Most NLP tasks require semantic comprehension, however, MWPs also need mathematical logic to arrive at the final solution. The mapping between the question and solution spaces is many-to-many and unrestricted. Questions that have different semantic mean-

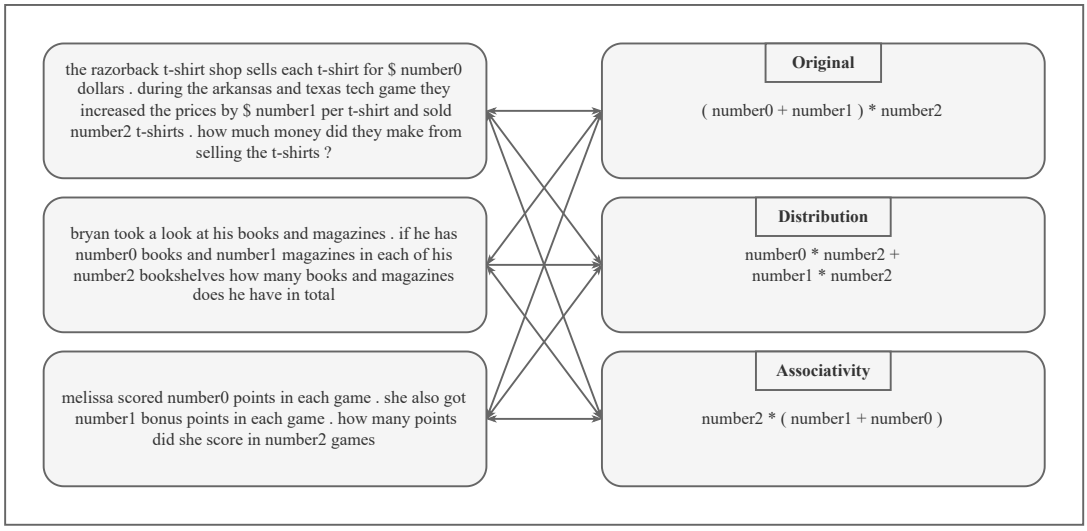


Figure 1.1: Many-to-many mapping between questions and solutions. Questions have different contexts and semantics but the same ground truth solution. Questions extracted from SVAMP [Patel *et al.* 2021].

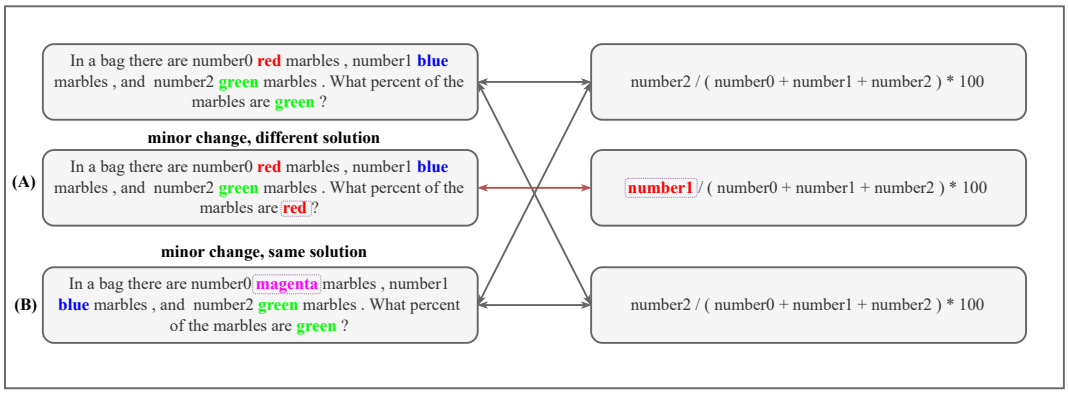


Figure 1.2: A minor change in the question can lead to a completely different solution. Questions extracted from SVAMP [Patel *et al.* 2021].

ings can have the same solution. Due to the laws of distribution and associativity in mathematical expressions, a single question can have multiple correct solutions.

Figure 1.1 shows examples with significant semantic and structural differences and yet have the same solution. The model must learn a feature space robust enough to be invariant of these semantic distinctions. It is also possible that a minor change in the problem statement could result in a completely different solution. For example, in Figure 1.2 (A), we can see that changing only one word, the "green" in the question with "red" leads to a different solution. However, substituting the word "red" with any color other than "green" will lead to the same solution. Similarly, a minor change in the solution may or may not lead to a different solution. For example, swapping values is permissible during addition and multiplication; however, results in errors during subtraction and division [Sundaram *et al.* 2022].

Given the challenges and utility of MWPs, efforts to automatically solve MWPs date back to the 1960s [Bobrow 1964; Charniak 1969; Feigenbaum and Feldman 1963]. Manually crafted rules were developed to match the MWPs with the corresponding mathematical expression that resolves to the answer [Bakman 2007; Mukherjee and Garain 2008; Yuhui *et al.* 2010]. For these strategies to be generalizable, the crafted rules must be exhaustive to cover every possible MWP in an unrestricted language. This is not feasible in the real world, limiting the number of applications for such a system [Li *et al.* 2022a; Sundaram *et al.* 2022].

The subsequent research adapted the use of statistical and machine learning techniques to automatically learn the rules relating the MWP to its solution [Kwiatkowski *et al.* 2013; Goldwasser and Roth 2014]. In this setup, an MWP dataset is used to teach a model the relationships between the query values and the background information of the math problems. The model generates the relations as mathematical equations that answer the query [Zhang *et al.* 2020a]. Although this configuration is an improvement over the initial strategy, the models can only solve a small number of problems that are highly similar to the training data.

The limitation of the statistical methods has led to the development of deep learning methods to solve MWPs [Wang *et al.* 2017]. Deep learning is capable of solving several mathematical reasoning challenges [Lample and Charton 2019; Saxton *et al.* 2019; Henighan *et al.* 2020; Charton *et al.* 2021]. However, the datasets used are relatively large in comparison to the datasets in the MWP domain. The cost of quality annotated MWPs data is greater than that of synthetically generated symbolic mathematics [Upadhyay *et al.* 2016; Lample and Charton 2019; Saxton *et al.* 2019; Hendrycks *et al.* 2021; Patel *et al.* 2021].

To deal with limited data that have complex relations to be learned, transfer learning has been successfully applied in natural language processing (NLP) to achieve state-of-the-art (SOTA) performance on small datasets [Dai and Le 2015; Devlin *et al.* 2018; Lewis *et al.* 2019; Malte and Ratadiya 2019]. This learning paradigm has been shown to outperform models that are only trained directly on the MWP datasets [Sundaram *et al.* 2022; Lan *et al.* 2022]. While using these models or parts thereof increases performance, the models learn shallow heuristics that are sensitive to minor changes in the math problems [Patel *et al.* 2021; Shen *et al.* 2021].

A big leap forward in the application of transfer learning in NLP came with the Transformer architecture [Vaswani *et al.* 2017]. This allowed researchers to pre-train language models on available large datasets. These models can further be fine-tuned on smaller datasets that exhibit similar properties to the pre-training objective.

Particularly in the MWP domain, pre-trained language models (PLMs) have been shown to outperform models that are only trained directly on the MWP datasets [Sundaram *et al.* 2022; Lan *et al.* 2022; Patel *et al.* 2021]. Although PLMs elicit better performance, these models learn shallow heuristics that are sensitive to small changes in math problems [Patel *et al.* 2021].

Several strategies have been proposed to make deep learning models learn MWPs more effectively. Particularly, the sequential encoding and decoding in pre-trained Transform-

ers struggle to capture the structural information that MWPs exhibit. Graph encoders [Li *et al.* 2020; Zhang *et al.* 2020c], along with hierarchical tree decoders [Xie and Sun 2019] and bottom-up Directed Acyclic Graphs (DAGs) [Cao *et al.* 2021; Jie *et al.* 2022; Li *et al.* 2022a] introduce architectural modifications to incorporate structural information originating from MWPs through of.

The changes made to the encoder/decoder architecture of the PLMs enforce that the model learns the mathematical relationships between the quantities in a structural manner. This is something that traditional sequential architecture does not explicitly require. The architectural modifications, however, are slower to train and too specialized for mathematical problems. The goal of this research is to see if the sequence-to-sequence Transformer can learn deeper heuristics and structural information by using learning objectives rather than specialized architectural changes.

Specifically, we propose a multi-view generation objective to learn structural information and a verification objective to learn deeper heuristics. Our multi-view generation objective is to learn the prefix, infix, and postfix notations of the ground truth expression tree of a given MWP. We show that a model trained to generate multiple views of the abstract syntax tree (AST) demonstrates deeper heuristics than a model that is trained to generate only one view. Furthermore, the ability to generate multiple views that evaluate to the same solution shows that the model has some understanding of the structural properties of mathematical expressions. The multi-view objective has been proposed in the literature [Zhang *et al.* 2022a; Shen and Jin 2020]. These proposals use two views and different architectures for the decoders. Different to these, we use three views and sequential pre-trained decoders.

Our verification objective, inspired by Shen *et al.* [2021], trains our model to mark MWP solutions as correct or incorrect. We show that a model that can generate and validate expressions has a deeper understanding of the MWP task and does not use shallow heuristics that generate tokens that could potentially yield the correct solutions without fully understanding the MWP.

With our multi-view generation objective and a verification objective, we propose a new multi-view ranking (MVR) framework, where we not only treat these two objectives as auxiliary tasks but further consider their output to choose the final solution. In particular, given an MWP, our multi-view generation task makes the prefix, infix, and postfix views of the AST solution, and we use the correctness score from the verification task to choose the best view.

We conducted several experiments throughout to show the effectiveness of our strategies. We implement a baseline BART model which achieves an accuracy of 40.6% on the SVAMP [Patel *et al.* 2021] dataset, while the SOTA accuracy is at 45.0%. Our implementation of the multi-view objective improves the baseline's generation accuracy by 2.3% while the verification task improves it by 6.0%. Furthermore, we run experiments that show that using these two objectives to generate the final solution results in an overall accuracy of 50.3%, an accuracy improvement of 9.7% from our baseline, and 5.3% on the current SOTA results for SVAMP.

In summary, the contributions of this work are as follows:

- We propose a multiview generation objective, where our model has to generate the prefix, infix, and postfix notations of the solution to MWPs. This objective exposes the model's misunderstanding of the MWP, which would otherwise be concealed by a single-generation objective.
- We also propose refinements to Shen *et al.* [2021]'s proposed verification objective, rephrasing it as a metric learning task rather than a classification task. This objective determines whether the model can distinguish between correct and incorrect solutions by embedding the correct solutions closer to the MWP than the incorrect solutions.
- We also propose a multi-view ranking framework that uses the proposed multi-view generation task to generate multiple candidate solutions and the verification task to choose the best candidate solution. This framework enables the model to produce a variety of potential solutions, which the verification task will evaluate before choosing the best one.
- We conduct experiments that show that our proposed model is capable of correctly answering a greater number of MWPs than previous models that were considered to be SOTA.

Having outlined our main contributions, we give a brief overview of the structure of this work. Chapter 2 is dedicated to an in-depth discussion of the research objectives. This chapter unfolds as follows: Section 2.1 offers a comprehensive exploration of the research objectives, elucidating the core goals and intentions of this study. Following this, Section 2.2 formulates and presents the research hypothesis, laying the groundwork for the central thesis that this research endeavors to test and validate. Subsequently, in Section 2.3, we systematically outline the research questions that serve as the investigative pillars of this work. These questions provide a strategic framework for our inquiry, guiding us in our quest to unravel the complexities and nuances of our chosen subject matter.

Chapter 3 lays the theoretical foundations of our work. This chapter starts with a brief introduction in Section 3.1. We then discuss the Transformer architecture used for all of our experiments in Section 3.2. Sections 3.3 and 3.4 discuss metric learning and the pooling mechanism, respectively. These concepts are useful for our verification objective. Thereafter, we detail NLP concepts and learning paradigms applied in our research in Sections 3.5 and 3.6 respectively. Section 3.7 briefly discusses MWPs, expression notations, and expression properties. Thereafter, Section 3.8 relates the Transformer architecture to other architectures proposed in the literature. The background information is then summarized in Section 3.9.

We present our proposed multi-view and verification objectives in Chapters 4 and 5, respectively. In particular, Chapters 4 and 5 are structured as independently and self-contained as possible. These two chapters follow a similar structure: they start with an introduction and motivation section (Sections 4.1 and 5.1); which is followed by their respective related work (Sections 4.2 and 5.2); the methodology of the chapters then follows (Sections 4.3 and 5.3); then the experiments and results are discussed (Sections 4.4 and 5.4); and a conclusion section summarizes the chapter and discusses

future work (Sections 4.4 and 5.4).

In particular Chapter 4 aims to explore how data representation impacts the performance of Transformers to solve MWPs. It focuses on quantity masking and the expression notation used in the MWP domain. We propose two major changes: a random indicator mask and generating three notations/views instead of one. We find that the multi-view notation setup requires a verifier to realize its full potential. Chapter 5 sets out to investigate the verification objective and integrate it with the multi-view objective which results in our multi-view ranker (MVR) model.

Having presented ablation studies on the data representations, and our proposed model along with its benchmark performance, we conclude our work in chapter 6. Furthermore, we outline the direction for future work based on ours.

Chapter 2

Objective

2.1 Research Objective

The primary aim of this research is to enrich our comprehension of the strategies employed to bolster the performance of pre-trained Transformers in the realm of logical tasks, with a specific focus on solving Math Word Problems (MWPs). The overarching objective is to augment the model's problem-solving abilities by introducing auxiliary learning tasks, thereby enhancing the heuristics acquired, while refraining from the necessity of devising specialized neural architectures. In pursuit of this objective, this study introduces two essential learning tasks: the multi-view generation task and the verification task.

The multi-view generation task is designed to enable the model to comprehend the abstract syntax tree (AST) underlying MWPs. This objective necessitates the model to perceive the solution not through a single sequential lens, but rather through three distinct views, which mirror the essential components of an AST. The model's ability to generate these three views not only indicates its understanding of the structural relationships within the problem but also its capacity to tackle MWPs with a more structural and semantic approach.

Simultaneously, the verification task is introduced to help the model discern correct solutions from incorrect ones. This auxiliary task equips the model with the ability to identify multiple correct solutions in MWPs, which are rooted in the underlying mathematical principles governing the expressions.

2.2 Research Hypothesis

Our research is underpinned by the following hypothesis: The performance of the pre-trained BART model in solving MWPs can be significantly improved by integrating a multi-view generation and verification objective. Moreover, these auxiliary tasks can be instrumental in generating the final solution, where the verification task serves as a mechanism for selecting the most suitable solution from the multiple generated views.

2.3 Research Questions

2.3.1 Research Questions for the Multi-View Generation Task

The core research questions addressed in Chapter 4 revolve around the multi-view generation task, these include:

- Which of the quantity masking techniques available in the existing literature proves most effective when applied to the pre-trained BART model?
- To what extent can the BART model autonomously generate the prefix, infix, and postfix views, both individually and collectively, and how does its performance compare in these scenarios?
- Is it more advantageous for the BART model to generate all three views in unison, or is it more beneficial to condition the model to generate a specific view based on a given prompt?
- Does training the BART model to generate consistent state vectors for terms in mathematical expressions that correspond to the same nodes in the AST result in improved performance?

2.3.2 Research Questions for the Verification Task

In Chapter 5, the research focus shifts to the verification task, and the following questions are explored:

- How does the verification objective perform as an auxiliary task when compared to the baseline BART model and the multi-view objective? Additionally, what is the combined performance of these two objectives?
- Can the verification task be harnessed to re-rank expressions generated through beam-search by the BART model based on their correctness, thereby enhancing overall accuracy? How does this strategy compare to a voting mechanism that selects the most consistent solution from the beams?
- Does the treatment of the verification objective under a metric learning paradigm yield superior results compared to a classification paradigm in terms of performance and accuracy?

By addressing these research questions, this dissertation aims to advance the field's understanding of leveraging auxiliary tasks to improve the performance of pre-trained Transformers on logical tasks, with a specific focus on Math Word Problems.

Chapter 3

Background and Related Work

3.1 Introduction

In this chapter, we introduce the theoretical foundation for this thesis. We start by discussing the Transformer architecture as an advancement over the traditional sequence-to-sequence architecture in Section 3.2. This forms the basis of our model. Section 3.3 discusses metric learning and how it can be used to rank and learn better representations. This research uses metric learning for verification and to enforce consistency in the multi-view generation objective. Before we can pass the representations to the verifier, we need to pool from the model representations. Strategies that can be used are discussed in Section 3.4.

Section 3.5 discusses key concepts of Natural Language Processing (NLP), including tokenization, greedy search, and beam search. Section 3.6 briefly discusses learning paradigms and how they can improve the model's ability to learn the task at hand. These tasks include transfer learning, multitask learning, multi-view learning, and multi-model learning.

Section 3.8 gives an overview of the various architectures found in the literature that solve MWPs. We discuss these architectures and learning objectives in relation to the Transformer. In section 3.9 we briefly summarize the theoretical background presented in this chapter.

3.2 Transformers

The Transformer is an attention-based neural architecture designed to address sequence-to-sequence modeling tasks while also dealing with long-range dependencies. It is the current state-of-the-art technique in the field of Natural Language Processing and Computer Vision [Lin *et al.* 2022; Khan *et al.* 2022; Dosovitskiy *et al.* 2020; Carion *et al.* 2020].

This section starts with a discussion of the recurrent structures that were used to process sequential data before the Transformer architecture introduced in Section 3.2.1. After

that, we provide an architectural summary of the Transformer in Section 3.2.2. A detailed architectural breakdown of the critical parts related to our work is given in the subsections that follow. In particular, Section 3.2.3 and Section 3.2.4 discuss the input and positional embeddings, respectively. Thereafter, Section 3.2.5 discusses the attention mechanism used in Transformers.

3.2.1 Background (Recurrent Neural Networks)

Before the introduction of the Transformers, most state-of-the-art NLP solutions were based on gated recurrent neural networks (RNNs), such as Long short-term memory (LSTMs) [Gers *et al.* 2000; Wen *et al.* 2015; Chen *et al.* 2016; Wang and Jiang 2015] and gated recurrent units (GRUs) [Cho *et al.* 2014; Chung *et al.* 2014; Kumar *et al.* 2016; Zhang *et al.* 2018]. The vanilla RNNs for sequence-to-sequence modeling encode the entire input into one context vector that the decoder uses to generate the desired output. This restricts the amount of information the decoder can use and, as a result, the performance of the RNNs. An attention mechanism was introduced to allow the decoder to create a context vector by choosing which encoder states to attend to at every time stamp [Wang *et al.* 2016b; Liu *et al.* 2017; Irie *et al.* 2016]. This improved the RNN's accuracy, however, not its speed during training.

Transformers were introduced to improve the speed of sequential data modelling. They replace all the recurrent structures in RNNs with attention mechanisms. Vaswani *et al.* [2017] showed that provided with enough training data, attention mechanisms alone can match the performance of RNNs with attention. This eliminated the need to sequentially process data during training and allowed researchers to efficiently use parallel computing architectures. As such, larger models can be trained on large available datasets at a lower time complexity.

Sequential Processing

The Feedforward Neural Networks (FNN) are the first and simplest type of layers of artificial neurons devised [Haykin 1998; Zell 1994; Schmidhuber 2015] where information flows in one direction only. However, natural language presents the challenge of modeling sequential data of uneven lengths. To resolve this, recurrent neural networks (RNNs) were introduced. These architectures use a feedback loop to enable data to be recirculated into the input before being sent again for additional processing and final output.

The feedback loop allows RNNs to maintain a context vector that contains information on the data seen before the current step. At step n, the model combines the context state representing the sequence up to step $n-1$ with the input at n to create a new context state that represents the sentence up to the current step n.

Challenges with RNNs

Theoretically, the information from each step can propagate arbitrarily far down the sequence if at every point the state continues to encode contextual information about

the step. In practice, this mechanism is flawed: the vanishing gradient problem leaves the model's state at the end of a long sentence without precise, learnable information about the preceding sequence. The influence of the earlier steps on the later steps decreases as the sequence length increases. This causes vanilla RNNs to suffer from short-term memory.

Gated RNNs

A common way to deal with gradient instabilities is through gates. The gates control how the context state is updated at each step. Each step can either add, delete, or leave the context vector unchanged. This reduces the repeated multiplications caused by the feedback loop. As a result, gated RNNs can model longer-term dependencies. Two commonly used gated RNNs are Long Short-Term Memory (LSTM) [Hochreiter and Schmidhuber 1997; Gers *et al.* 2000] and Gated Recurrent Unit (GRU) [Cho *et al.* 2014; Chung *et al.* 2014].

Unresolved challenges by gated RNNs

Gated RNNs achieved state-of-the-art results before the introduction of Transformers [Salehinejad *et al.* 2017]. However, these have several drawbacks. The computation of each step depends on the result of the computation of the previous step. This makes training RNNs inefficient, as they cannot be parallelized on modern deep-learning hardware. The other challenge with RNNs is that future input information cannot be reached from the current state. To resolve this, two RNNs are chained in opposite directions to form a bi-directional RNN [Bahdanau *et al.* 2014]. However, these RNNs are independent, which restricts the contextual information encoded for each step. The Transformer architecture addresses these challenges by replacing the recurrences in these networks with attention.

Attention

The attention mechanisms allow a model to create a context vector by drawing from all states at any point along the sequence. The attention layer has access to all previous states and can weigh them using a learned relevance metric to provide pertinent data about distant tokens. Attention has been used with Recurrent Neural Networks (RNNs) and has been shown to improve performance [Wang *et al.* 2016b; Luong *et al.* 2015]. The development of the Transformer architecture showed that attention mechanisms could function effectively on their own, eliminating the need for sequential recurrent data processing to produce the performance seen in RNNs with attention. Transformers employ an attention mechanism without an RNN, processing all tokens simultaneously and allocating attention weights among them in layers. Since the attention mechanism only uses information about other tokens from lower layers, it can be computed for all tokens in parallel at each layer, which leads to improved training speed.

Encoder-Decoder Model

The RNN and the Transformer output the hidden state corresponding to each input token. This means that the output length is equal to the input length. In most sequence-to-sequence problems, the desired output length often differs from the input length. To handle this, the model can use a prompt token or end-of-string token to denote that the input sequence ends, and the model should start generating the corresponding target sequence. During training, the loss is computed for the tokens after the prompt. Similarly, during inference, the outputs before the prompt are disregarded. This architecture utilizes the same network to encode the input and generate the output sequence. This approach is adapted by models such as GPT [Radford *et al.* 2018 2019].

Alternatively, two networks can be used to encode and generate the target sequence. This resembles the encoder-decoder architecture. Firstly, the encoder learns the embedding of each token in the context of the whole sequence. Thereafter, the decoder takes this sequence of embedding or a summary thereof to generate the desired target sequence of tokens. The decoded sequence is conditioned on all the input tokens and their previously generated tokens. This variant is widely used for sequence-to-sequence problems [Mohamed *et al.* 2021; Tan *et al.* 2020].

3.2.2 Architecture Overview

The Transformer follows an encoder-decoder architecture that consists of embeddings, multi-headed attentions, and FNNs. This is depicted in Figure 3.1. Given a sequence of tokens, the Transformer creates embedding vectors for each token and its position using an input layer and a positional embedding layer. The embeddings are then passed through a series of encoder and decoder blocks on the encoder and decoder, respectively. In contrast to RNNs, all inputs are fed into the network simultaneously rather than sequentially.

Each encoder and decoder block contains attention components that allow each processed token to be aware of relevant information from other tokens in the sequence. The decoder output is passed through a fully connected layer and a softmax function that converts these to the probability of each token in the vocabulary. Teacher forcing is used to train the decoder [Raffel *et al.* 2020], allowing training to occur in parallel. Since the decoder will not have access to future tokens during inference, an attention mask is used to hide these tokens so that the model doesn't attend to them during training.

3.2.3 Input Embedding

The first layer in the Transformer is the input, or word embedding layer. First proposed in Bengio *et al.* [2000], the input embedding layer takes in a tokenized sequence of numbers in a fixed-size vocab Σ and maps these to vectors of continuous numbers. These vectors represent the learned meaning of each token, so tokens that are closer in the embedding space are expected to have similar meanings. The context, semantics, and syntactic properties, as well as the linear relationship between tokens, can be cap-

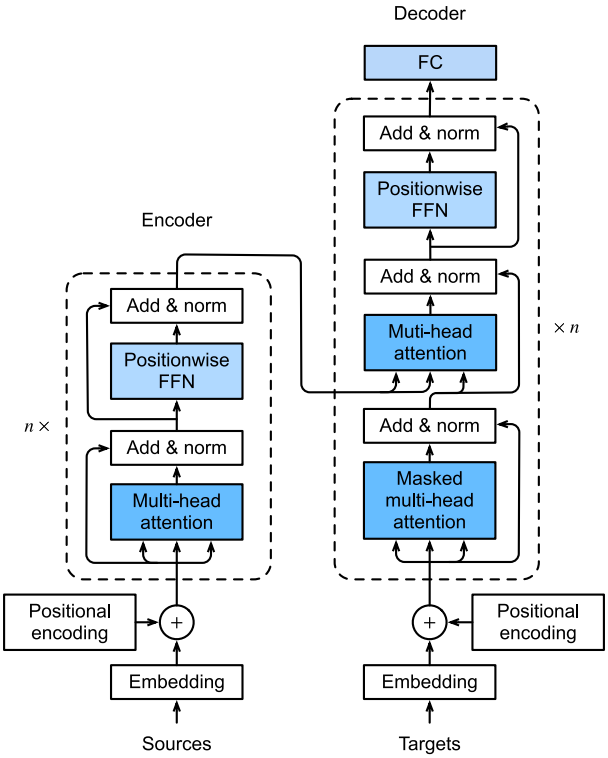


Figure 3.1: Transformer architecture[1]

tured by vectors more efficiently than by the discrete entity in the vocabulary [Collobert and Weston 2008; Mikolov *et al.* 2013].

More formally, given a vocabulary Σ, and a sequence of discrete tokens $X = \{x_1, x_2, ..., x_N\}$, where $x_i \in \Sigma$, the input embedding (IE) layer learns a mapping of each token to a d-dimensional real-valued vector $IE : \mathbb{N} \rightarrow \mathbb{R}^d$. The dimension of the embedding d, which represents the width of the Transformer, is a hyper-parameter that is normally chosen to be much smaller than the vocabulary size ($d << |\Sigma|$) [Collobert and Weston 2008; Stahlberg 2020]. For instance, the Transformer vocabulary for language modeling typically has up to 50000 tokens although the embedding dimension is within the range of $512 - 1024$.

A higher-dimensional embedding can capture fine-grained relationships between words, however, requires more data to learn. The study carried out by Wies *et al.* [2021] shows that the ratio between the dimension and vocabulary of the data and the depth of the Transformer can have an impact on performance. Furthermore, the dimension d should be divisible by the number of attention heads.

Press and Wolf [2016] showed that there is a performance gain when the input embedding layer is shared between the decoder and encoder where the input and output vocab is the same. As a result, it's been adopted in the design of the Transformer

[Vaswani *et al.* 2017; Lewis *et al.* 2019; Raffel *et al.* 2020].

3.2.4 Positional Encoding

The input embedding gives the embedding of each token, however, does not account for the order or position of each token. However, the meaning can change depending on the order of the tokens. For example, "a mosquito bit John" and "John bit a mosquito" have the same tokens, yet, they mean different things due to the order of the tokens. RNNs can implicitly learn positional information from their sequential processing. Differing from RNNs, Transformers process data in parallel using an attention mechanism that does not have a notion of order. As such, the transformer explicitly adds positional information to the input embeddings through this module. Positional encodings PE map the position of each token to a d-dimensional real-valued vector $PE : \mathbb{R} \rightarrow \mathbb{R}^d$. This is then summed with the input embedding $E = IE + PE$ to produce embeddings with both positional and contextual information. Alternatively, these two vectors can be concatenated; however, this will increase the memory footprint and will slow down training, as the optimization process needs to adjust more parameters due to the extra dimensions.

The original implementation of the Transformer model used sine and cosine functions of different frequencies to add positional information denoted with the following equation:

$$PE_{(pos,i)} = \begin{cases} \sin(pos/10000^{2i/d}) & \text{if } i = 2k \\ \cos(pos/10000^{2i/d}) & \text{if } i = 2k+1 \end{cases}$$

where pos denotes the position of the token in the input sequence, i is the index of each dimension in the output vector, and k is the offset. This positional encoding can capture absolute and relative positional information. Furthermore, it can scale to unseen lengths of sequences. Vaswani *et al.* [2017] also experimented with learned positional embeddings [Gehring *et al.* 2017a] and found that the results are on par with the sinusoidal encoding proposed. Subsequent work [Devlin *et al.* 2018; Radford *et al.* 2018; Lewis *et al.* 2019] adopted the learned positional embedding. Wang and Chen [2020] do an analysis of positional embeddings under different training objectives. The positional encoder is not shared between the encoder and the decoder, in contrast to the input embedding layer. The fully encoded input tokens E are then passed to the multi-headed self-attention layer.

3.2.5 Attention Mechanism

The attention mechanism is a central component of the Transformer architecture, and its operation varies between the encoder, decoder, and the encoder-decoder model. The

[1]Adapted from https://d2l.ai/chapter_attention-mechanisms-and-transformers/transformer.html

Transformer treats the input representations E as a set of key-value pairs. For each position j in E, the attention mechanism queries its relationship to all representations in E. This is known as the self-attention mechanism since queries and key-value pairs are of the same sequence. This is used in the encoder and decoder to learn the relationship between the inputs. Furthermore, the decoder has a cross-attention mechanism where queries come from its inputs and key-value pairs come from the encoder.

More formally, the attention mechanism maps a query vector Q and a set of key-value vector pairs (K, V) to an output vector A. The output is generated by computing a weighted sum of the values. The weight assigned to each value is determined by a compatibility function of the query with the corresponding key. These weights are called attention scores [Vaswani *et al.* 2017]. Since Q, K and V are d-dimensional, they can be organized into matrices $Q \in \mathbb{R}^{M \times d}$, $K \in \mathbb{R}^{N \times d}$ and $V \in \mathbb{R}^{N \times d}$, where M is the query sequence length, and $M = N$ for self-attention.

Scaled Dot-Product Attention

The Transformer adopts a scaled version of the dot product attention [Luong *et al.* 2015] to compute the attention scores. Figure 3.2b depicts this, and can be denoted with the following equation:

$$A = Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$$

The similarity between Q and K is computed as their dot-product scaled by $1/\sqrt{d}$. This is then passed through a softmax function to produce the attention scores. The dot-product of the attention scores and V gives the output matrix $A \in \mathbb{R}^{M \times d}$. The scaling prevents the softmax function from giving values close to 1 for highly correlated vectors and values close to 0 for non-correlated vectors, making gradients more reasonable for back-propagation. The softmax function normalizes over the columns of the scaled dot product of Q and K so that the weights for each query vector sum up to one.

Multi-Head Attention

Instead of performing a single attention function with Q, V, and K, the Transformer performs the scaled dot-product attention h times using separate identical attention layers. A single layer is called a head and h is the number of heads in multi-head attention (MHA), represented by Figure 3.2a.

For each head, learned linear projects are used to transform Q, K, and V to a d/h-dimensional space, then passed through the scaled dot-product attention in parallel, yielding h attention heads with d/h dimensions. These are concatenated and projected again to a d-dimension matrix. The total computational complexity is comparable to that of single-head attention with full dimensionality because of the reduced dimension of each head.

Formally, MHA can be defined as

$$MHA = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^A$$

where $W^A \in \mathbb{R}^{d \times d}$ and $\text{head}_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$, $W_i^Q, W_i^K, W_i^V \in \mathbb{R}^{d \times d/h}$ for $i \in [1, h]$. With MHA, the model can look at information from different representation subspaces at the same time. Voita *et al.* [2019] and Clark *et al.* [2019] provide more information on the different semantics captured by each head.

Masked Multi-Head Attention

The encoder and decoder have different requirements for attention due to their distinct roles in the Transformer model. While the encoder operates on an input sequence, the decoder generates an output sequence. To address this, the model uses masked multi-head attention.

In the encoder, there is no need for masking since all positions in the input sequence can be considered during the self-attention process. However, in the decoder, the self-attention mechanism must prevent information flow from future tokens to the current token, to maintain causality. This is achieved by applying an attention mask. When generating token $\hat{y}_t$ at time step t, the model should only attend to the tokens preceding it, i.e., $\hat{y}_0, y_1, ..., \hat{y}_{t-1}$. To enforce this constraint, the attention mask sets all future attentions (i.e., those for $t+1, t+2, ..., T$) to negative infinity in the input of the softmax. As a result, these future attentions receive zero attention scores and do not influence the current token's generation.

Beyond temporal masking, attention masks are also useful for cases where sequences in a batch have varying lengths. To handle this, special padding tokens are often added to sequences, and the model should be prevented from attending to these padding tokens. The attention mask serves this purpose as well.

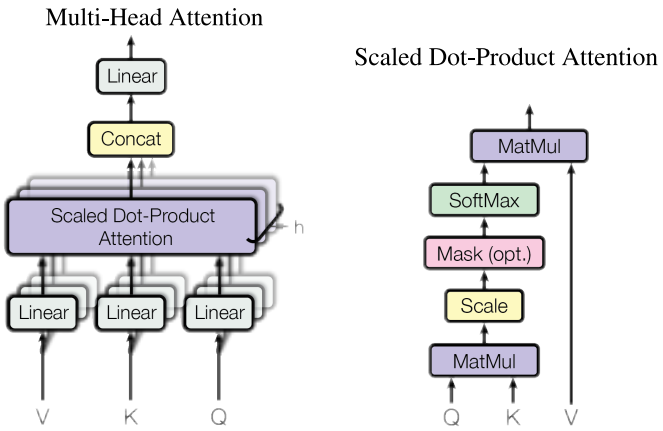
Self-Attention

The Transformer employs self-attention [Lin *et al.* 2017b; Cheng *et al.* 2016; Parikh *et al.* 2016] in the encoder and a masked version thereof in the decoder. The representation of each position in the sequence is computed by relating it to every other unmasked position in the sequence. The Transformer entirely depends on the attention mechanism as opposed to RNNs and Convolutional Neural Networks (CNNs) [Gehring *et al.* 2017b; Kalchbrenner *et al.* 2016]. Self-attention is free from the assumptions made by RNNs and CNNs that the embedding of one token depends more on its neighboring tokens [Dou 2022].

Cross-Attention

The second MHA layer in the decoder applies cross-attention, where the queries are derived from the decoder and the key-value pairs from the encoder's final outputs. This module is the only connection between the encoder and the decoder and serves to mix the two embedding sequences. As such, the decoder has access to its previously generated tokens through self-attention and the input tokens via this module.

In summary, self-attention is used in the encoder to model relationships within the input sequence, while masked self-attention is applied in the decoder to ensure causal



(a) Multi-Head Attention (b) Scaled Dot-Product Attention

Figure 3.2: Transformer - multi-head attention architecture. Adapted from Vaswani *et al.* [2017].

generation of output tokens. Cross-attention, on the other hand, bridges the gap between the encoder and the decoder, enabling the decoder to incorporate information from both its own generated tokens and the encoder's output sequence. This distinction in how attention is used between the encoder, decoder, and encoder-decoder model is a key feature of the Transformer architecture.

3.2.6 Position-wise Feedforward Neural Network (FNN)

At the end of each encoder/decoder layer, the Transformer inserts a fully connected feedforward network that is applied to each position separately and identically. This consists of two linear transformations with a non-linear activation in between, usually a ReLU [Nair and Hinton 2010] or GeLU [Hendrycks and Gimpel 2016]. This incorporates non-linearity into the model explicitly and strengthens the model capacity [Kim 2022].

3.2.7 LayerNorm and Residual Connections

To improve the stability of training, the "Add & norm" applies a residual connection [He *et al.* 2016] and layer normalization [Ba *et al.* 2016] in the output of the MHA module and the feedforward module. Residual connections add the input of a module to its output, improving gradient propagation. This addresses the problem of deterioration of the training accuracy as more layers are added to the neural architecture [He *et al.* 2016]. Layer normalization adjusts the mean and standard deviation of the output of a layer, which can accelerate the training. The output of the "Add & Norm" can thus be defined as:

$$LayerNorm(Z_R) = \frac{Z_R - \mathrm{E}[Z_R]}{\sqrt{Var[Z_R] + \epsilon}} \cdot \gamma + \beta$$

where γ, β are learnable parameters, $Z_R = Z + Z_O$ is the residual output with Z being the input to each sublayer (MHA or feedforward), and Z_O is the output from these sublayers. A small value ϵ is added in the denominator for numerical stability. The mean $\mathrm{E}[Z_R]$ and variance $Var[Z_R]$ are respectively denoted by the following:

$$\mathrm{E}[Z_R] = \frac{1}{N}\sum_{i=1}^{N} Z_R^{(i)} \qquad Var[Z_R] = \frac{1}{N}\sum_{i=1}^{N}\left(Z_R^{(i)} - \mathrm{E}[Z_R]\right)^2.$$

3.2.8 Final layer

The output of the final decoder layer passes through a final layer, which acts as a classifier/output embedding layer. This maps the final hidden states of the decoder from d-dimension back to the vocabulary size. The classifier output is then fed into a softmax layer, which will produce probability scores between 0 and 1. The index of the highest probability score is taken as the predicted token. Following the works of Press and Wolf [2016], the output embedding layer is tied to the input embedding which reduces the number of parameters and also improves performance.

3.3 Metric Learning

In Chapter 5 we propose to solve Math Word Problems as a verification task. Verification can be addressed as a metric learning task, a classification task, or more recently, a disentangled representation learning task [Ridgeway and Mozer 2018; Lee *et al.* 2020]. This section provides insights into the fundamentals of deep metric learning methods related to the verification setup in Chapter 5. The goal of the metric learning algorithm is to learn a metric that assigns a small distance to similar points and a relatively large distance to dissimilar points.

The training can follow a weakly supervised or supervised setting. In the weakly supervised setting, the goal is to learn a distance metric that puts positive pairs close together and negative pairs far away, while in the supervised setting, the goal is to learn a distance metric that puts points with the same label close together while pushing away points with different labels. This research focuses on using a supervised setting.

More specifically, for a set of data points $X = \{x_1, x_2, ..., x_N\}$, where $x_i \in \mathbb{R}^d$ and their corresponding discrete labels $Y = y_1, y_2, ..., y_N \in \mathbb{Z}^+$, the goal is to train a model $F_\theta(\cdot) : X \to \mathbb{R}^d$ together with a distance $D : \mathbb{R} \to \mathbb{R}$, such that for any pair of points $x_i, x_j \in X$, $D(x_i, x_j)$ is smaller if the corresponding labels are the same, that is, $y_i = y_j$, $y_i, y_j \in Y$, otherwise it should produce a larger value. Here θ represents the learned weights, d is the embedding dimensions, and N is the number of data points. Thus, we need to choose the right distance function D, and the loss function $\mathcal{L}$ to train our model F_θ [Kha Vu 2021]. The distance metric D is discussed in Section 3.3.1, followed by loss functions in Section 3.3.2 and how to choose training samples in Section 3.3.3.

3.3.1 Distance Metric

A Metric is a non-negative function D between two points x_i and x_j that describes the notion of 'distance' between these two points. Let X be a nonempty set. A distance or metric over X is a map $d : X \times X \to \mathbb{R}$ that satisfies the following properties:

- Non-negativity: The distance between two distinct points is always positive, $D(x_i, x_j) > 0$ and $D(x_i, x_i) = 0$.
- Symmetry: The distance from x_i to x_j is always the same as the distance from x_i to x_j, i.e., $D(x_i, x_j) = D(x_j, x_i)$.
- Triangular inequality: The sum of distances between two lines drawn from three points cannot be greater than the third line, $D(x_i, x_j) \leq D(x_i, x_h) + D(x_h, x_j)$, $\forall x_i, x_j, x_h \in X$.

One other property of distance metrics not mentioned above is the identity of indiscernibles, which states if $D(x_i, x_j) = 0$, then $x_i = x_j$. This property does not need to hold for metric learning; as such, we consider *pseudo-distances* [Collatz 2014]. Furthermore, for the purpose of training our models, it is usually sufficient to calculate some transformation of the distance metric that still gives us a similarity measure between x_i and x_j.

Euclidean Distance

Euclidean distance is one of the most common distance measures. Also known as the L_2 metric, this can be described as a measure of the length of a line segment between two points in Euclidean space. Using the Pythagorean theorem [Weisstein 2006], the square of the euclidean distance can be calculated from the Cartesian coordinates of the points:

$$D(x_i, x_j) = ||x_i - x_j||^2$$

The square of the euclidean distance is often used since it avoids computing the square root, making it more efficient from a computation perspective. Although it is not really a distance, it retains the most useful properties of a distance metric from a learning perspective. The Euclidean distance is not scale invariant, which can result in skewed distances. As such, it is beneficial to standardize/normalize before use. Moreover, as the dimensionality d increases, the less useful Euclidean distance becomes less meaningful. It fails to capture non-linearity in data since the distance is isotropic, i.e., the same in every direction.

The Mahalanobis distance is a more general case of the euclidean distance that is non-isotropic, given by $D(x_i, x_j) = ||Wx_i - Wx_j||^2$, where W is a linear transformation matrix found by decomposing the inverse of the positive-definite covariance matrix $M = W^T W$ of a given probability distribution. For the euclidean distance, W is the identity matrix. This distance is prone to overfitting due to its dependence on the mean, and the linear transformation W is not always able to capture the inherent non-linearity of the data.

Cosine Distance

The cosine distance uses cosine similarity to determine the distance between data points. Cosine similarity can be defined as the cosine of the angle between two vectors in an inner product space, which can be computed as the normalized Euclidean dot product of two vectors:

$$sim(x_i, x_j) = \frac{x_i \cdot x_j}{||x_i||||x_j||}$$

Cosine similarity ranges between -1 and 1, where 1 means the angle between x_i and x_j is 0, i.e., they are similar. To ensure that cosine similarity is non-negative, we can define cosine distance as:

$$D(x_i, x_j) = 1 - sim(x_i, x_j)$$

The cosine distance measures the dissimilarity between x_i and x_j. Since the cosine similarity is computed in the Euclidean space, we can derive the equivalent Euclidean distance to the cosine distance. When x_i, and x_j are normalized to unit vectors such that $||x_i|| = ||x_j|| = 1$ we have:

$$||x_i - x_j||^2 = ||x_i||^2 + ||x_j||^2 - 2x_i \cdot x_j = 2(1 - sim(x_i, x_j))$$

As we can see, the cosine distance is given by $\frac{||x_i - x_j||^2}{2}$. While the cosine distance is not a metric itself, we can easily see that it is a square of the Euclidean distance, and the square root thereof produces a distance metric [Schubert 2021]. This is suffices for metric learning.

From a computation perspective, a dot product is faster because it saves on one vector subtraction and does not suffer from "catastrophic cancellation" for small distances [Schubert and Gertz 2018; Lang and Schubert 2020]. As such, it is more efficient to compute the Euclidean distance as a dot product.

The unnormalized Euclidean distance is commonly used on dense, continuous variables where dimensional magnitudes matter, whereas the normalized Euclidean distance is commonly used on sparse, discrete domains such as text since it was more robust to variations in occurrence counts between semantically related terms.

Similar to the Euclidean distance, cosine distance suffers from the curse of dimensionality, though with a lesser effect. As the dimensionality increases, the number of data points required for good performance of any machine learning algorithm increases exponentially. Chen *et al.* [2020b] found that projecting the embedding vectors $F_\theta(\cdot)$ to a lower dimension before applying the distance measures produces better results. To handle this, data mining methods, together with loss functions are implemented to be more robust to this.

3.3.2 Losses

The loss function is an objective function that our model learns to minimize or maximize. This function produces a scalar value that indicates how the model fits the data;

therefore, improvements in this number correspond to a better model [Goodfellow *et al.* 2016; Reed and MarksII 1999]. The metric learning objective function predicts relative distances between inputs, which differs from other objective functions for classification and regression problems, whose objective is to learn to predict a label, a value, or a set of values given an input.

Standard regression and classification algorithms can be categorized as point-wise loss functions, where a single data point is considered at a time through the loss function. On the other hand, metric learning considers a pair or a list of data points at a time. That is, the metric learning objective trains the model F_θ to produce similar representations by minimizing the embedding distance when the data points have the same label, otherwise, the model should produce distant representations, i.e., maximize the embedding distance.

Pairwise Loss

The pairwise loss function dates back to one of the first metric learning losses proposed [Chopra *et al.* 2005; Hadsell *et al.* 2006]. It follows a simple and intuitive objective function to directly minimize the pairwise distance between two samples, x_1 and x_2, with the following formulation:

$$\mathcal{L} = \mathbb{I}_{y1=y2} D(x_1, x_2) + \mathbb{I}_{y1\neq y2} \max\left(0, m - D(x_1, x_2)\right)$$

where $m > 0$ is a set hyper-parameter margin, defining the lower bound distance between samples of different labels. When the representations produced for a negative pair are sufficiently distant, no effort is wasted in enlarging that distance, so further training can focus on more difficult pairs.

Triplet Loss

The pairwise loss uses two samples, however, more samples can be used. For instance, Wang *et al.* [2014], Schroff *et al.* [2015], Cui *et al.* [2016] and Hermans *et al.* [2017] use triplets instead of pairs and reported improvements in performance. Triplet-based losses consist of an anchor x, a positive sample x^+, and a negative sample x^-. The triplet loss learns a distance metric by which the anchor is further from the negative samples than it is from the similar point by a margin $D(x, x^-) > D(x, x^+) + m$. The loss can be formulated as:

$$\mathcal{L} = \max(0, m + D(x, x^+) - D(x, x^-))$$

where m is the margin. The margin prevents a trivial solution to the loss function which is found when both of the distances are equal to zero $D(x, x^+) = D(x, x^-) = 0$ Since the network is updated with both similar and dissimilar samples, the triplet loss requires fewer samples for convergence. However, the performance of the triplet loss is highly dependent on the sampling process of x^+ and x^-.

If the positive is already closer to the anchor than the negative, the model will learn almost nothing since the requirement on the relative distance of the embeddings is

already satisfied; however, if the sampled positive is far away from the anchor or a negative very close to it, the model improves much more during each training step [Kaya and Bilge 2019].

The triplet loss only compares a sample to one negative sample during one update, ignoring other negative samples. The embedding vector for a sample is only guaranteed to be far from the selected negative sample, not from the all other negatives. Thus, an example can be distinguished from a few negative samples while remaining close to many other negatives. After looping over enough randomly sampled triplets, the final distance metric should be balanced; however, individual updates may be unstable and converge slowly [Sohn 2016; Weng 2021].

N-pair Loss

Sohn [2016] proposed to extend the triplet loss to include a comparison with multiple negative samples. The proposed N-pair loss, uses one anchor x, one positive sample x^+ and $N-1$ negative samples $\{x_i^-\}_{i=1}^{N-1}$. If $N=2$, the N-pair loss reduces to triplet loss. Mathematically, this can be denoted as follows:

$$\begin{aligned}\mathcal{L} &= \log\left(1+\sum_{i=1}^{N-1}\exp(D(x,x_i^-)-D(x,x^+))\right)\\ &= -\log\frac{\exp(D(x,x^+))}{\exp(D(x,x^+))+\sum_{i=1}^{N-1}\exp(D(x,x_i^-))}\end{aligned}$$

The equation produced by N-pair loss is similar to the softmax loss. Although the softmax loss may assist the model being trained to converge to the optima, it does not seek to keep positive pairs closer and negative pairs farther similar to what metric losses do. Softmax loss only learns indistinct features. As a result, the methods use metric learning with softmax. Several proposals [Goldberger *et al.* 2004; He *et al.* 2020a; Oord *et al.* 2018; Gutmann and Hyvärinen 2010 2012] follow the same structured loss function, with different distance measures D. Increasing the number of negative samples N has been found to improve the contrastive power of the model [Khosla *et al.* 2020; He *et al.* 2020a; Chen *et al.* 2020b; Tian *et al.* 2020; Henaff 2020].

Supervised Contrastive Learning

A more generalized form of N-pair loss was proposed by Khosla *et al.* [2020], which can cater to multiple positive samples in addition to multiple negatives. The proposed SupCon loss aims to improve the weaknesses of the N-pair loss and the softmax function. The heavy dependence on hard mining from the N-pair loss is minimized by using many positives and many negatives for each anchor. Moreover, the softmax loss function was shown to outperform the N-pair loss for classification tasks [Horiguchi *et al.* 2019]. However, the softmax loss is prone to produce poor margins. The SupCon loss is designed so that it can effectively use the ground truth targets to embed samples with the same label closer than samples with different labels. The mathematical derivation

is given by

$$\mathcal{L} = \sum_{i=1}^{2N} \frac{1}{2|N_i| - 1} \sum_{j \in N_i, j \neq i} \log \frac{\exp(D(x_i, x_j^+))}{\sum_{k \in \mathbb{I}, k \neq i} \exp(D(x_i, x_k^-))}$$

where $N_i = \{j \in \mathbb{I} : \tilde{y}_i = \tilde{y}_j\}$ contains a set of labeled sample indices and $|N_i|$ is its cardinality. Khosla *et al.* [2020] found that including more positive samples into the set N_i leads to better results. Furthermore, SupCon was found to be less sensitive to hyper-parameter changes and noise [Khosla *et al.* 2020].

Multi-Similarity Loss

The drawbacks of conventional loss functions such as triplet loss and contrastive loss, which sometimes lack discriminative power when applied to large datasets, are addressed by the multi-similarity (MS) loss [Wang *et al.* 2019b]. Traditional loss functions, which are limited in their ability to capture complex similarity relationships in high-dimensional feature spaces, only consider pairwise similarity relationships between samples.

The idea behind MS loss is to measure similarity in more than one direction, as opposed to just one or two, as is the case with conventional loss functions. The loss function then encourages the model to learn a representation of features that maximizes the similarity between positive examples and minimizes the similarity between negative examples.

$$\mathcal{L} = \frac{1}{|X^+ \cup X^-|} \left[\frac{1}{\alpha} \log[1 + \sum_{(x, x^+) \in X^+} \exp(-\alpha(D(x_i, x_j^+) - m))] \right.$$
$$\left. + \frac{1}{\beta} \log[1 + \sum_{(x, x^-) \in X^-} \exp(\beta(D(x_i, x^-)) - m))] \right]$$

where α and β are hyper-parameters.

3.3.3 Miners

Deep metric learning relies on the ability to learn meaningful representations of data, where the similarity or dissimilarity between data points is crucial. Unlike conventional classification or regression tasks, in which random sampling of data points often suffices, deep metric learning requires a more nuanced approach to sample selection. The reason lies in the fundamental objective of the model: instead of predicting labels, it aims to understand the relationships between data points, learning to push similar items closer and dissimilar ones apart [Bucher *et al.* 2016; Kaya and Bilge 2019; Wang *et al.* 2019b].

The effectiveness of the deep metric learning network depends not only on the quality of its mathematical models and architectures but also on the discriminative power of the samples it encounters during training. To facilitate the network's learning and enhance the quality of the learned representations, it is imperative to provide it with training instances that challenge its ability to distinguish between data points.

In this context, sample selection plays a pivotal role as a preprocessing step before applying the deep metric learning model. The process of selecting the right training examples to feed the network is akin to setting the stage for the model's success [Schroff *et al.* 2015].

To train the model effectively to learn a meaningful distance metric, it is essential to sample pairs of positive and negative data points from the datasets. This process of sample selection and pair creation is often referred to as "mining." The importance of mining in the context of deep metric learning has garnered considerable attention in the research community, with various techniques and strategies proposed to optimize the learning process [Wang *et al.* 2018a 2014; Wu *et al.* 2017; Hermans *et al.* 2017; Schroff *et al.* 2015; Sikaroudi *et al.* 2020].

Anchor, Positive and Negative Samples

Let X be the set of examples and Y be the set of corresponding labels, denoted as $\{(x_i, y_i)\}$. Within this context, we define the sets of pairs as follows: The positive pair set, denoted as X^+, consists of pairs where the labels match, that is, $X^+ = \{(x, x^+) \in X \times X \,|\, y = y^+\}$. The negative pair set, denoted as X^-, comprises pairs with differing labels, which can be represented as $X^- = \{(x, x^-) \in X \times X \,|\, y \neq y^-\}$.

In this context, the samples x, x^+, and x^- are typically referred to as the anchor, positive, and negative samples. The primary objective is to train the model, represented as f_θ, such that for every anchor, the positive sample embeddings, as measured by some distance function D, are closer to the anchor than the negative sample embeddings by a margin m. Formally, this can be expressed as:

$$D(x, x^+) + m < D(x, x^-), \quad \forall x \in X, (x, x^+) \in X^+, (x, x^-) \in X^- \tag{3.1}$$

Easy, Semi-hard and Hard Samples

Given an anchor, positive, and negative sample combination (x, x^+, x^-), the inequality in Equation (3.1) from the previous section may or may not be violated. When the inequality is not violated, the positive and negative samples (x^+, x^-) are considered easy. Since the model already gets them right, easy samples do not improve the network when trained.

Semi-hard samples occur when the inequality in Equation (3.1) is only violated by the margin m, that is, $D(x, x^+) < D(x, x^-) < D(x, x^+) + m$. In other words, the negative sample is within a marginal distance away from the positive.

When the negative sample is closer to the anchor than the positive sample, (x^+, x^-) is considered a hard sample. This is denoted as $D(x, x^-) < D(x, x^+)$. The semi-hard and hard samples represent samples that the model has not yet mastered. Therefore, we need to mine these samples so that we can train the model on them.

Offline Mining

Mining for hard samples can be done offline before the batch is created. The whole training set is passed through the model in order to obtain the model embeddings. Subsequently, the distances are computed, where the sets of anchor, positive and negative (x, x^+, x^-) are dropped if they are easy. Different implementations keep only the hardest, hard, semi-hard, or a combination of these. The remaining valid sets are divided into K batches. In this way, offline mining creates batches that have the most informative samples.

This technique computes the embeddings of the full training set to generate batch samples, which have some memory and computation overhead. Valid samples can change after the batch gradient updates. This means that the offline mined sets need to be updated frequently, after several steps, or at the end of every epoch to avoid overfitting. This renders offline mining inefficient. Offline mining can be adopted when the datasets are small or if the hardness of the samples can be computed independently of the model embeddings where these drawbacks are less dire [Prétet *et al.* 2020].

Online Mining

Alternatively, mining can be done online, within each batch. First introduced in Schroff *et al.* [2015], online mining aims to reduce the computation and memory burden of offline mining by only considering the positive and negative samples in the sampled batch. Furthermore, this method uses up-to-date weights of the model to compute the embeddings. For each class label, n samples are selected for every batch. This provides the anchors and positive samples. Every other sample in the batch with a different label is considered a negative sample. This will generate easy, semi-hard, and hard samples. All valid triplets or only the hardest samples are used in the loss function to update the weights, where the latter often performs best [Schroff *et al.* 2015; Hermans *et al.* 2017].

The online option is the preferable choice for larger datasets since it can be computationally prohibitive to compute and sort the pair-wise distances between all samples [Schroff *et al.* 2015; Musgrave *et al.* 2020]. Online mining has been well adopted in the literature [Wang *et al.* 2019c; Liu *et al.* 2019a; Lee *et al.* 2018; Chen *et al.* 2020a]. However, online mining requires big batch sizes to include enough distinct negative data to push the model to acquire meaningful representation to distinguish various examples [Oh Song *et al.* 2017; Kaya and Bilge 2019].

Online Batch Hard Mining

In the online batch hard mining, we seek to find the hardest positive and hardest negative for each anchor. A positive sample x_h^+ and a negative sample x_h^- are said to be the

hardest if the following holds:

$$x_h^+ = \underset{(x,x^+)\in X_B^+,(x,x^-)\in X_B^-}{\text{argmax}} D(x,x^+) - D(x,x^-)$$
$$x_h^- = \underset{(x,x^+)\in X_B^+,(x,x^-)\in X_B^-}{\text{argmin}} D(x,x^+) - D(x,x^-)$$

where X_B^+ and X_B^- are subsets of the positive and negative sets, $X_B^+ \subseteq X^+$ and $X_B^- \subseteq X^-$ respectively, within the current batch. Mining for the hardest samples is computationally taxing, however, relatively less compared to doing it offline. Furthermore, the hardest sample for online mining is relative to the batch, which makes it a moderate sample and less prone to causing the model to overfit.

Hard sampling provides the biggest gradient updates, which can speed up convergence. However, mining very hard samples early on during training can cause the model to get stuck on local minima, and consequently, lead to model collapse [Hermans *et al.* 2017; Harwood *et al.* 2017; Wu *et al.* 2017]. Semi-hard sampling avoids the model overfitting on outliers and improves its robustness. However, only a few semi-hard samples can be generated, requiring large batch sizes. A curriculum where harder samples are gradually increased was found to perform better than semi-hard mining [Harwood *et al.* 2017].

Online Batch All Mining

Rather than mining for the hardest samples, all the valid combinations can be considered for the loss computation. This strategy is known as batch-all mining. For B samples in a batch X_B and n samples for each label, all the positive samples $X_B^+ = \{(x,x^+) \in X^+ | (x,x^+) \in X_B\}$ and negative samples $X_B^- = \{(x,x^-) \in X^- | (x,x^-) \in X_B\}$ we have a total of $B(B-n)(n-1)$ triplets that contribute to the loss.

The batch-all approach results in less computational overhead in comparison to batch-hard online mining. However, this approach was found to perform consistently worse than batch hard online mining [Ding *et al.* 2015; Wang *et al.* 2016a]. Hermans *et al.* [2017] found that the inclusion of easy triplets was the major cause and only considering combinations that violate the loss function improves performance. Khosla *et al.* [2020] and Wang *et al.* [2019b] propose loss functions that intrinsically perform hard mining by downweighing the contribution of easy samples.

Multi-Similarity Miner

Wang *et al.* [2019b] proposed a mining technique in which negatives are selected if they are closer to an anchor than its most difficult positive, and positives are selected if they are further away from an anchor than its most difficult negative. More formally, for positive samples $X_B^+ \subseteq X^+$ and negative samples, $X_B^- \subseteq X^-$ in a batch, and a

margin m, (x, x^+) and (x, x^-) are mined for the loss if the following holds:

$$D(x, x^-) > \min_{(x,x^+)\in X_B^+} D(x, x^+) - m$$
$$D(x, x^+) < \max_{(x,x^-)\in X_B^-} D(x, x^-) + m$$

3.4 Pooling

Processing sequential data yields the same number of embedding vectors as there are tokens in the input sequence. That is, the embeddings are at a token level with a variable length. It is often desired to distill the information encoded by each of the token embedding vectors into one fixed-length vector that represents the context of the sequence, that is, a sequence embedding. The context vector can then be used for classification and metric learning tasks such as sentiment analysis, natural language inference, and text similarity [Farouk 2019; Kowsari *et al.* 2019; Williams *et al.* 2017; Reimers and Gurevych 2019].

Pooling is a procedure that can be used to produce a representative context vector. Several pooling methods can be used, such as simply taking the average of all the tokens or taking the max value at each dimension across all tokens. Reimers and Gurevych [2019] found that the significance of the pooling procedure depends on the objective function and the datasets used. As a result, it is worth experimenting with different pooling strategies.

In the following subsections, we discuss several pooling methods that can be used to create a context vector. In particular, we discuss single token pooling (Section 3.4.1), max-pooling (Section 3.4.2), average pooling (Section 3.4.3), attentive pooling (Section 3.4.4), and mixed pooling (Section 3.4.5).

3.4.1 Single Token pooling

Early sequential processing techniques, such as the RNNs produced a context vector at the end of a sequence that stored the information of all the tokens it saw. As such, for a sequence that is concatenated by two special tokens, a beginning-of-string token bos at the beginning and an end-of-string token eos at the end, the context vector in RNNs is the eos token. Transformers on the other hand do not have this leverage.

Devlin *et al.* [2018] train their proposed Transformer architectures to produce a context vector through the bos token. The bos token, also called the CLS token, is more consistent than the eos token since it occurs at the same position every time. performs well for the classification objective under which it was trained, however, it produces poor results for metric learning tasks [Reimers and Gurevych 2019]. Furthermore, pooling the bos token models that were not pre-trained to produce a context vector through this token can lead to poor performance.

3.4.2 Max-pooling

Another common strategy to produce a compressed representation is to pool the maximum value over time for every dimension [Kalchbrenner *et al.* 2014; Joulin *et al.* 2016]. Formally known as max-pooling [LeCun *et al.* 1998], this method extracts the most prominent features across all dimensions. Furthermore, max-pooling is invariant to sentence length, as opposed to average pooling and attentive pooling [Suárez-Paniagua and Segura-Bedmar 2018]. Max-pooling is particularly well suited to the separation of features that are very sparse [Boureau *et al.* 2010ba].

With very long sequences, the information from most token embeddings will be dropped. To preserve more information, the more general k-max-pooling method can be employed [Kalchbrenner *et al.* 2014]. For a sequence with N d-dimensional token embeddings $E^N \in \mathbb{R}^d$ and $k \leq N$, the k-max-pooling operation pools the k most active features in E, producing k fixed vectors $E^k_{max} \in \mathbb{R}^d$. The hyper-parameter k can be set statically or dynamically as a function of N to allow for smooth extraction of features [Kalchbrenner *et al.* 2014; Shu *et al.* 2018]. To produce one context vector, Zhao *et al.* [2018] sums E^k_{max} and scales this by a factor s. If $s = k$, the context becomes the mean of E^k_{max}, and if $k = N$, then the scaled k-max-pooling is equivalent to average pooling.

3.4.3 Average Pooling

The average pooling [LeCun *et al.* 1998 1989; Reimers and Gurevych 2019] computes the context vector by taking the element-wise arithmetic mean of the token embeddings. Different from max-pooling, average pooling down-weighs the activated features by combining the non-maximal features. This makes the average pooling procedure more suited for preserving background information.

3.4.4 Attentive Pooling

Another option to produce the context vector is to use attention [Luong *et al.* 2015; Bahdanau *et al.* 2014] to get a weighted sum of the inputs instead of the average pool. Not all tokens in the sequence contribute equally to the meaning of the sentence. The attentive pooling mechanism learns to weigh the relevant tokens more than the irrelevant ones to produce a context embedding. The relevance weights are learned alongside the model. Given a sequence of N embeddings $E = \{e_1, e_2, ..., e_N\}$, the context vector is produced by the following:

$$
\begin{aligned}
E_{seq} &= \Sigma_{i=1}^{N} \alpha_i e_i \\
\alpha_i &= \frac{\exp(u_i^\top u_s)}{\Sigma \exp(u_i^\top u_s)} \\
u_i &= \tanh(W_s e_i + b_s)
\end{aligned}
$$

where E_{seq} is the sentence embedding, α_i is the attention weight corresponding to the i^{th} token embedding, and W_s, b_s, and u_s are trainable parameters. The sentence-level context vector u_s can be randomly initialized and jointly learned during the training process [Yang *et al.* 2016; Wu *et al.* 2020a]. Alternatively, u_s can be set to be the unit

matrix [Lin *et al.* 2017b; Chen *et al.* 2018]. The attention mechanism can be extended to multi-head attention [Chen *et al.* 2018; Wu *et al.* 2020c].

Different from the mean and max pools, the attentive pool contains trainable parameters. This allows the model to choose which tokens it finds the most useful for the sequence context at the expense of additional parameters and possibly training time. Furthermore, under low data settings, the attentive pool can lead to over-fitting [Li *et al.* 2018].

3.4.5 Mixed Pooling

It is also possible to use more than one pooling technique. For instance, Yu *et al.* [2014], Lee *et al.* [2016], Zhao *et al.* [2018] and Zhou *et al.* [2021] saw performance gains when they combined average pooling and max-pooling. This combines the advantages of each strategy. The max pool or average pool can be used as the query vector for an attentive pooling procedure [Maini *et al.* 2020; Liu *et al.* 2016]. Sahu and Anand [2018] and Suárez-Paniagua and Segura-Bedmar [2018] concatenates the results of multiple pooling strategies. Concatenating these methods, however, introduces more parameters that take more time to learn or tune and increases computational overhead. Gholamalinezhad and Khosravi [2020] and Zafar *et al.* [2022] review several other pooling techniques used in the literature.

3.5 Natural Language Processing

The challenge to solving Math Word Problems (MWPs) is found in the domain of Natural Language Processing (NLP). More specifically, MWPs are presented as textual data, and then fed through an NLP model, which is expected to generate a mathematical expression that solves the problem. NLP models process text in its token form, which is produced through a process known as tokenization.

The objective to solve MWPs can be presented to the model as a sequence generation task, where the model needs to generate the tokens corresponding to the correct solution, or as a ranking task, where the model generates scores for several hypotheses as its choice to the final solution. In either case, the model first encodes each of the tokens into an embedding space and then uses these embeddings to generate the desired outcome. The point-wise ranking task can be viewed as a metric learning task, with the input MWP as an anchor, the model learns to score positive solutions higher than negative solutions [McFee and Lanckriet 2010; Cakir *et al.* 2019].

The remainder of this section is organized as follows: In Section 3.5.1, we delve into the tokenization process, encompassing word-level, character-level, subword-level, and byte-level tokenizers. Subsequently, in Section 3.5.2, we explore two generative objective techniques, namely, greedy search and beam search. We then delve into the utilization of label smoothing and teacher forcing, which can effectively improve the generation objective during the training process.

3.5.1 Tokenization

Tokenization is an integral part of virtually every NLP task and is arguably the first step in preprocessing that is linguistically motivated because it identifies the basic units on which all other processing is based [Michelbacher 2013]. Tokenization breaks raw text into discrete pieces representing either words, phrases, symbols, or other meaningful elements called tokens [Gupta and Malhotra 2015; Verma *et al.* 2014]. The tokens can be treated as atomic components of the text, and in turn, form the building blocks of NLP [Lyon 2022]. Having tokenized our dataset, we can create our vocabulary, which is a set of unique tokens used in the text corpus.

Text can be tokenized using a variety of methods, including word-level, character-level, and subword tokenization, respectively. Inadequate tokenization of any portion of the text can result in misconceptions later on in the natural language processing process.

Word-level Tokenization

A relatively simple and straightforward approach to split the text into words or tokens is to use a delimiter, such as white spaces or punctuation. Advanced rules can be incorporated to handle hyphens, suffixes, and prefixes [Honnibal and Montani 2017; Speer 2019; Koehn *et al.* 2007]. Word tokenization collectively refers to approaches that split text on a word level.

An issue with word tokens is that taking every unique word in a large text corpus can lead to a very huge vocabulary sizes. For instance, Dai *et al.* [2019] use a word-level tokenizer that generates a vocabulary that is four times larger than single language pre-trained Transformers utilizing other granular tokenization techniques. A big vocabulary size necessitates an enormous input and output embedding layer, which increases the model's memory and time complexity.

Dealing with words that are "out of vocabulary" (OOV) is another significant challenge associated with word tokens. The majority of the time, this problem can be fixed by mapping all unknown words to a single token that represents 'unknown' tokens. This presents a number of difficulties, one of which is that the meaning of many of the tokens that are OOV is lost.

Character-level Tokenization

Rather than splitting text into words, it could be split into characters, this is exactly what character-level tokenization does. This resolves the above-mentioned issues with the word-level tokenization technique. The vocabulary is relatively small, which can have a size of 26 for a text-only case-insensitive English corpus, or 256 to incorporate all the ASCII[2] codes.

Character-level Tokenization can also handles OOV words coherently by preserving the

[2] https://www.asciitable.com/

information of the word. It breaks down the OOV word into characters and represents the word in terms of these characters. This method is particularly useful for languages where individual characters carry important information, such as Chinese or Japanese. This strategy has been adopted by Xue *et al.* [2022] to design a token-free Transformer that can be used in unilingual and multilingual settings.

Character-level tokenizer requires no training to learn a vocabulary. Several drawbacks arise with this tokenization technique, one being the length of the sequences of tokens it produces. Because each world is now comprised of all characters, the length of the tokenized sequence has the potential to easily surpass that of the original text. In addition, the model has to spend more capacity to reach a higher-level representation compared to other tokenization methods since the individual tokens on their own do not have any semantic meaning.

Subword Tokenization

The primary goal of subword-based tokenization techniques is to solve the large vocabulary size and a significant number of OOV tokens in word-based tokenization, and the challenges presented by character-based tokenization, which results in extremely long sequences and individual tokens with little significance. To do this, the subword-based tokenization algorithms do not split the frequently used words into smaller subwords.

Some of the popular subword tokenization algorithms are WordPiece [Devlin *et al.* 2018], Byte-Pair Encoding (BPE) [Sennrich *et al.* 2015], Unigram, and SentencePiece [Kudo and Richardson 2018]. Our research adopts the BPE tokenizer, in particular, the Byte-level BPE tokenizer used in our pre-trained model [Lewis *et al.* 2019; Radford *et al.* 2019]. This is crucial because a pre-trained model can only work successfully if it is fed input that has been tokenized using the same rules as were used to tokenize its training data.

Byte-level BPE Tokenization

Byte-level BPE (BBPE) tokenizer a variant of the BPE tokenizer [Gage 1994; Sennrich *et al.* 2015], which was initially developed as an algorithm to compress texts and has been widely adopted for pre-trained language models [Liu *et al.* 2019c; Radford *et al.* 2018 2019; Lewis *et al.* 2019; Lample and Conneau 2019].

While standard BPE uses characters as the base tokens, BBPE utilizes bytes, such as those found in Unicode encoding, as the fundamental units of tokenization. This allows the tokenization to operate at a finer granularity, similar to character tokenization. The tokenization process can either remain at the byte level or use Byte Pair Encoding to merge bytes into larger units, depending on the specific requirements of the task.

In practice, the Byte-level BPE Tokenization process begins by creating a base vocabulary using bytes. Using the byte-level base tokens, the BBPE algorithm identifies the most frequent ”byte-pairs” and replaces them with a new, combined token that represents both of the original tokens. This process continues until a predetermined number of merge operations have been completed or the desired vocabulary size has been

achieved. The number of merge operations and the size of the target vocabulary are configurable hyper-parameters [Wang *et al.* 2020; Radford *et al.* 2019].

The benefit of BPE is that it can capture both common and uncommon words in a corpus. BPE ensures that the most common words are represented as a single token in the vocabulary, while the rare and OOV words are broken down into smaller sub-word units that are shared with other words, allowing them to be more easily represented in the model. This makes BPE especially useful for languages with complex morphology, where words can take on a variety of inflections and conjugations. Moreover, the length of tokenized texts is shorter than character-level tokenization, and the vocabulary size is smaller than word-level tokenization [Mielke *et al.* 2021; Webster and Kit 1992].

3.5.2 Sequential Generation Task

The sequential generation task in NLP involves generating text one word or token at a time, in sequential order. The output generated at each time step is used as input for generating the next output. The model learns to predict the probability distribution of the next word given the previous words generated, and on some input context. Greedy search and beam search are two commonly used algorithms for generating sequences in NLP. Sequential generation is widely used in various NLP tasks such as language translation, text summarization, and dialogue systems [Lin *et al.* 2022; Kalyan *et al.* 2021; Allahyari *et al.* 2017].

Greedy Search

In NLP, greedy search is a simple algorithm for generating text sequences. It is a type of search algorithm that generates the next word in a sequence by selecting the word with the highest probability at each time step. The initial seed for a greedy search can be a single word, phrase, or sentence. The model then generates, at each time step, a probability distribution over the vocabulary of possible subsequent words, based on the context of the preceding words. The model then selects the most probable word from this distribution and inserts it into the generated sequence. This process continues until a stopping criterion is met, such as a maximum sequence length or an end-of-sequence token. The resultant sequence is then output as the generated text's final output.

One of the primary benefits of greedy search is its simplicity and efficiency. It is a straightforward algorithm that requires minimal computational resources, making it fast and easy to implement. There are, however, some disadvantages to using greedy search. Due to the fact that it only considers the word with the highest probability at each time step, it can lead to suboptimal results, especially when the model makes an error early in the generation process, which can cause a cascade of errors [Meister *et al.* 2020a; Freitag and Al-Onaizan 2017; Yoo *et al.* 2020].

Beam Search

Beam search is an alternative algorithm to greedy search that keeps track of the top k most probable sequences at each time step, where k is a predefined beam size. The

algorithm is comparable to the greedy search algorithm, with the exception that it chooses the top k words from the generated probability distribution based on their probabilities. The algorithm then generates k new candidate sequences by appending one of the k words to each of the k existing sequences.

The k new candidate sequences are then ranked according to their probabilities, and the top k sequences are retained for the subsequent time step. This process continues until a stopping criterion is met, such as a maximum sequence length or an end-of-sequence token. The resulting sequence with the highest probability is then used to generate the final text.

Beam search has the advantage of exploring multiple options at each time step, which can result in higher output quality than greedy search. Beam search can handle situations where the most likely word at each time step might not lead to the best overall sequence. It does this by keeping track of multiple candidate sequences, which gives it a wider range of possible outcomes. Beam search is computationally more expensive than greedy search due to the need to keep track of multiple candidate sequences at each time step.

Beam search has the potential to result in repetitive or redundant output, as it prefers taking up paths that are highly probable. Various techniques have been proposed to modify the beam search algorithm to address this issue, such as length normalization, which adjusts the probabilities based on the length of the sequence, and diverse beam search, which encourages diversity in the output by penalizing redundant sequences [Meister *et al.* 2020a; Freitag and Al-Onaizan 2017; Yoo *et al.* 2020; Meister *et al.* 2020b].

Label Smoothing

Label smoothing is a regularization technique that is used to prevent models from becoming overconfident and to improve their generalizability. Smoothing labels replaces hard 0/1 targets in training data with soft targets that distribute probability mass to labels with comparable attributes. A smoothed distribution that gives more weight to similar labels is used in place of presenting a label as a one-hot vector with a single 1 and all other values as 0. This enables the model to identify representations that are less susceptible to noise or small input changes.

Smoothing labels requires a modification of the training cross-entropy loss function. In place of the conventional cross-entropy loss, which only penalizes the model for bad predictions, a penalty term is added that rewards predictions that are closer to the smoothed targets. Generally, this penalty term represents the Kullback-Leibler [Kullback 1997] (KL) divergence between model predictions and smoothed targets.

A set hyperparameter controls smoothing by moving the probability mass to other labels. Smoothing is controlled by a set of hyperparameters that move probability mass to other labels. Too much smoothing can cause the model to underfit, making it inflexible and unable to distinguish between labels. When the model becomes overconfident and performs poorly on new data, overfitting occurs. Label smoothing is a simple regularization technique that enhances the generalizability of deep learning models. This is

particularly true for NLP tasks with large label spaces and noisy or absent training data [Yuan *et al.* 2020; Lukasik *et al.* 2020; Müller *et al.* 2019].

Teacher Forcing

The idea behind teacher forcing is to use ground-truth tokens from the training data as inputs to the decoder during training, as opposed to the model's own predictions. This is performed to stabilize the training procedure and assist the model in learning to generate high-quality sequences.

During training, the model takes the source sequence as input and generates the target sequence, one token at a time. The decoder is fed the true target tokens at each time step rather than the model's own predictions. This means that at time step t, the decoder gets the correct token y_t instead of the token that the model predicted. The application of teacher forcing ensures that the model learns to produce high-quality sequences that closely match the target sequences in the training data [Lamb *et al.* 2016; Williams and Zipser 1989].

3.6 Learning Paradigms

In machine learning (ML), there are multiple learning paradigms, each of which addresses different types of problems and requires distinct algorithms and models. Supervised learning, unsupervised learning, and reinforcement learning are the three main learning paradigms that makeup ML. This classification is based on the learning objectives and data use types [Grira *et al.* 2004; Sathya *et al.* 2013; Morales and Escalante 2022].

Other aspects of the training process can be considered in addition to the learning objective, such as the number of models trained, the variants of the data, the number of tasks the model needs to learn, and whether the model is transferring knowledge from another task. These learning paradigms are discussed briefly in this section.

Supervised Learning The most prevalent and widely used learning paradigm in ML is supervised learning. In supervised learning, an algorithm is given a labeled dataset containing the correct outputs for each input. The objective of the algorithm is to discover a function that maps inputs to correct outputs so that it can make accurate predictions on new, unobserved data [Grira *et al.* 2004; Morales and Escalante 2022].

Unsupervised Learning In the unsupervised learning paradigm, the algorithm is provided with an unlabeled data set, and the objective is to identify patterns and structures in the data without explicit guidance. Clustering, anomaly detection, and dimensionality reduction are common unsupervised learning tasks [Grira *et al.* 2004; Morales and Escalante 2022].

Reinforcement Learning In reinforcement learning, the model or agent acquires knowledge through interaction with its environment. The objective of reinforcement

learning is to discover a set of actions or a policy that maximizes a reward signal. The agent receives feedback in the form of rewards or punishments and adjusts its behavior accordingly. Reinforcement learning is frequently used in complex environments for tasks such as game playing, robotics, and decision making [Grira *et al.* 2004; Sathya *et al.* 2013; Li 2017].

The rest of the section discusses transfer learning in Section 3.6.1, followed by multi-task learning paradigms in Section 3.6.2. Thereafter, we discuss multi-view learning and multi-model learning in Section 3.6.3 and Section 3.6.4 respectively.

3.6.1 Transfer Learning

Transfer learning is a technique in which a model that has been trained on a large dataset for a particular task is used as a starting point for a new, related task. The idea is that the pre-trained model has already learned features from the original dataset that can be transferred to the new task. Transfer learning makes it possible to train a new model faster and with less data than if it had to be trained from scratch [Zhuang *et al.* 2020].

Pre-training

Pre-training refers to training a model on a large, unsupervised dataset before fine-tuning it for a specific task on a smaller, labeled dataset. Pre-training is based on the the basis that by training a model on a large amount of data, it can learn general features and patterns that are useful for a variety of tasks without being explicitly guided on those tasks.

Pre-training is especially beneficial when there are few labeled data for a specific task. In such situations, pre-training a model on a large unlabeled dataset can assist the model in learning more general features that can be fine-tuned for the specific task using the smaller labeled dataset. This can improve the quality of the task while reducing the amount of data that must be labeled [Kamath *et al.* 2020; Church *et al.* 2021].

Fine-tuning

A model is fine-tuned for a specific task by training it on a smaller, labeled data set. The concept behind fine-tuning is that the pre-trained model has already learned general features and patterns from a larger dataset, and the process of fine-tuning is used to adapt these features to the specific task at hand [Yang *et al.* 2020; Zhuang *et al.* 2020; Torrey and Shavlik 2010; Weiss *et al.* 2016].

3.6.2 Multi-task Learning

Multi-task learning involves teaching a single model to perform several tasks. To make each task better, multi-task learning shares information between tasks. Multi-task learning assumes that some tasks have comparable characteristics or patterns and that if a

model learns to perform several tasks simultaneously, it can use these shared characteristics to enhance its performance on each task individually.

When there is insufficient labeled data for each task, the model can learn to perform multiple tasks using the same labeled data. It is common practice to train distinct output layers for each task while using the same representation of features for all tasks. Multi-task learning can enhance performance, reduce training time, and assist models in adapting to new tasks more effectively than training separate models. However, it can be challenging to design a multi-task learning model that balances shared and task-specific features and can learn multiple tasks independently [Zhang and Yang 2021; Ruder 2017; Zhang and Yang 2018].

3.6.3 Multi-view Learning

Multi-view learning is a machine learning (ML) technique that involves learning from multiple representations or "views" of the same data. Each view provides a unique perspective on the data, and the goal is to improve overall learning performance by combining information from multiple views.

In multi-view learning, data is typically represented using multiple feature sets or modalities, including images, text, audio, and graphs. Each feature set represents a unique perspective on the data, and the objective is to learn a model that can combine these perspectives to make accurate predictions or decisions.

Multi-view learning is similar to multi-task learning in that it involves simultaneously learning multiple related tasks. In multi-view learning, related tasks are associated with each perspective. The model can use the complementary information offered by each view to improve its performance on each task by learning to combine the different views in an efficient manner.

Multi-view learning has several advantages over single-view learning, including enhanced performance, enhanced generalization to new data, and enhanced robustness to noisy or incomplete data. Designing effective models that can successfully combine data from multiple views, especially when the views are heterogeneous or contain redundant information, can be difficult [Sun 2013; Zhao *et al.* 2017; Xu *et al.* 2013].

3.6.4 Multi-Model Learning

In multi-model learning, multiple models are trained on the same dataset to perform the same or different tasks. The objective is to combine the results of these models to enhance the overall precision or performance. Each model in multi-model learning may be trained on various subsets of the data or using various hyper-parameters to represent various viewpoints of the same data. The models may be optimized to perform a particular task or generate a particular type of prediction. There are various ways to combine the outputs of the models. Ensemble learning and stacking are two common methods [Tommasi *et al.* 2013].

Ensemble Learning

Multiple models are combined in ensemble learning to increase generalization and prediction accuracy. The predictions can collectively generalize well to new data. Techniques such as bagging, boosting, and random forests can be used to implement ensemble learning. Each technique has its strengths and weaknesses and can be chosen based on the nature of the data and the task [Sagi and Rokach 2018; Dong *et al.* 2020].

The bagging technique involves training multiple models independently, each on a random subset of the training data. These models utilize different data subsets, resulting in varying weights and feature sets. The final prediction is determined by averaging the predictions from each model or selecting the majority vote [Sagi and Rokach 2018; Zhou and Zhou 2021].

Bootstrapping trains weak models sequentially using weighted training data. After each round, the training data is weighed to emphasize misclassified samples. The average weighted accuracy of each model's predictions serves as the final prediction [Zhou and Zhou 2021; Dietterich and others 2002].

Random forests combine bagging and decision tree concepts. Multiple decision trees are trained on distinct subsets of data using random subsets of features in a random forest. Each decision tree in the forest is trained on a random subset of the data and a random subset of the features, and the final prediction is determined by averaging or voting on the predictions of each tree. Random forests can aid in reducing overfitting and improving the model's accuracy, particularly for high-dimensional or noisy datasets [Ho 1995; Fawagreh *et al.* 2014; Zhou and Zhou 2021].

Ensemble learning is helpful when several weak models work well on different subsets of data or features. Combining the predictions of multiple models has the potential to improve the accuracy and stability of the final result, particularly for datasets that are complex or noisy. Nevertheless, ensemble learning can be computationally expensive and may require careful hyperparameter tuning [Dietterich and others 2002; Sagi and Rokach 2018].

Staking

To make a prediction, stacking trains a meta-model using the outputs of multiple base models. Multiple models are stacked to improve accuracy or performance. In stacking, the base models and meta-models are trained independently. Different algorithms and hyperparameters are used to train multiple base models. Base models can be trained on various subsets of data or features. After training the base models, the second portion of the training data is predicted. Using these predictions, the meta-model is trained to make predictions [Xiao *et al.* 2018; Zhou and Zhou 2021; Ting and Witten 1997].

In stacking, a meta-model uses the predictions of more than one base model to improve the final result. Moreover, it may help us account for both types of relationships and make more accurate predictions. The stacking meta-model is determined by the problem and the base models. Meta-model inputs should be processed and a prediction made. Additionally, it should generalize to new data without being overfitted.

It could necessitate hyperparameter tuning and be computationally costly [Ting and Witten 1997; Xiao *et al.* 2018].

3.7 Math Word Problems

Math word problems are presented in natural language and frequently involve real-world scenarios that require mathematical analysis. In order to solve these problems, the reader must convert the given information into mathematical expressions or equations, perform mathematical operations on the problem, and then interpret the solution in the context of the original problem.

As an illustration, consider the following math word problem: "Lisa flew 256 miles at 32 miles per hour. How long did Lisa fly?"[3] The given information must first be converted into mathematical expressions or equations before this problem can be solved. In this case, the time Alisa flew can be denoted as x and the speed as $256/x$ (using the common knowledge that speed is the distance over time). The time Alisa flew is then determined by setting up the equation $256/x = 32$ and solving for x to find that the time Alisa flew is $8h$.

The remainder of this section delves into the mathematical laws or properties that oversee mathematical expressions, as addressed in Section 3.7.1. Following that, Section 3.7.2 explores the various notations used to represent mathematical expressions.

3.7.1 Mathematics Laws

In mathematics, there are several laws or properties that govern the behavior of numbers and operations. Some of the most important laws include the distribution law, associative law, commutative law, and identity law. These principles lead to the existence of multiple potential solutions for each mathematical word problem (MWP). Consequently, when assessing solutions generated for MWPs, it is crucial to account for this multiplicity and refrain from simply comparing them with the ground-truth solution.

- The distribution law states that multiplication is distributed over addition, which means that $a(b+c) = ab+ac$. For example, $2(3+4) = 2(7) = 14$, and $2(3)+2(4) = 6+8 = 14$.
- The associative law states that the way terms are grouped does not affect the result of the operation, which means that $(a+b)+c = a+(b+c)$. For example, $(2+3)+4 = 5+4 = 9$, and $2+(3+4) = 2+7 = 9$.
- The commutative law states that the order in which terms are added or multiplied does not affect the result of the operation, which means that $a+b = b+a$ and $ab = ba$. For example, $2+3 = 3+2$ and $2 \cdot 3 = 3 \cdot 2$.

[3] Question adapted from SVAMP dataset [Patel *et al.* 2021].

- The identity law states that there exist unique elements, known as the additive identity (0) and the multiplicative identity (1), such that adding 0 to any number leaves it unchanged, and multiplying any number by 1 leaves it unchanged. For example, $2 + 0 = 2$ and $2 \cdot 1 = 2$.
- The inverse law states that every element has an inverse such that when added or multiplied, they equal the identity element, and the distributive law of exponents, which states that $a^{(b+c)} = a^b \cdot a^c$.

3.7.2 Math Notations

The arithmetic expression can be naturally represented as a binary tree structure such that the operators with higher priority are placed in the lower level and the root of the tree contains the operator with the lowest priority. There are three main notations used to write arithmetic and logical expressions: prefix, postfix, and infix. Figure 3.3 shows an illustration of these.

Infix Notation

Infix notation is the notation that is most commonly used in mathematics. In infix notation, the operators are placed between the operands. For example, the expression "2 + 3" is written in infix notation as "2 + 3". Infix notation often requires the use of parentheses to indicate the order of operations, especially when dealing with complex expressions.

Prefix Notation

Prefix notation, also known as Polish notation, is a way to write arithmetic and logical expressions in which the operator comes before its operands. In prefix notation, the operators are written first, followed by the operands on which it operates. For example, the expression "2 + 3" can be written in prefix notation as "+ 2 3". The prefix notation eliminates the need for parentheses to indicate the order of operations and makes it easy to evaluate expressions using a stack-based algorithm.

Postfix Notation

Postfix notation, also known as reverse Polish notation, is a way of writing arithmetic and logical expressions in which the operator comes after its operands. In postfix notation, the operands are written first, followed by the operator that operates on them. For example, the expression "2 + 3" can be written in postfix notation as "2 3 +". Postfix notation eliminates the need for parentheses to indicate the order of operations and also makes it easy to evaluate expressions using a stack-based algorithm.

The prefix and postfix notations have several advantages over the infix notation. One of the main advantages is that they eliminate the need for parentheses to indicate the order of operations. The order is unambiguous in these representations because the

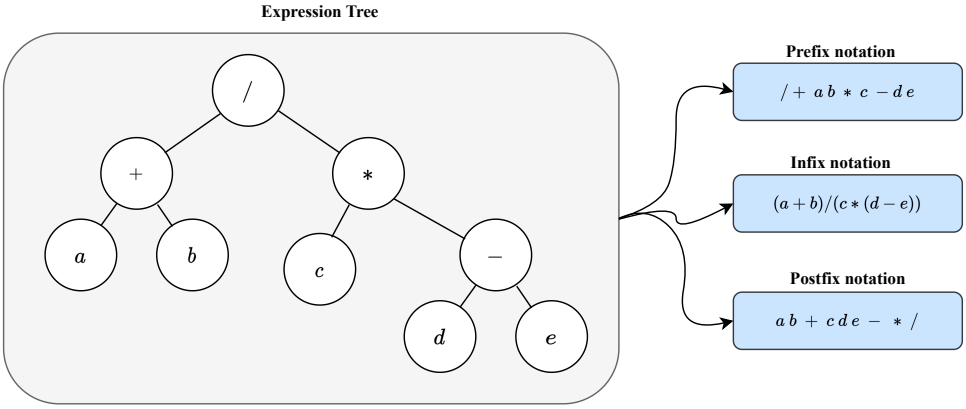


Figure 3.3: An example of an expression tree with the prefix, postfix, and infix traversals.

operands are written in the correct order, and the operator always precedes or follows them in the prefix and postfix notations, respectively.

Another advantage of the prefix and postfix notation is that they also make it easy to evaluate expressions using a stack-based algorithm. Postfix notation expressions are evaluated by reading them from left to right and stacking the operands. When the algorithm encounters an operator, it pops the required number of operands off the stack, applies the operator, and pushes the result back on. In both notations, the stack value after all operators is the result.

3.8 Related Work

The goal of automatically solving Math Word Problems MWPs is actively being investigated [Li *et al.* 2021c]. Several methods have been proposed to solve MWPs. The differences primarily lie in the architectures of the models. Earlier methods relied heavily on rule-based semantic parsing to solve MWPs, reviewed in Zhang *et al.* [2020a]. Huang *et al.* [2016] showed that these methods are limited in their potential to solve MWPs in their proposed diversified dataset. This led the researchers to develop deep learning techniques in efforts to eliminate the burden of rule-based data [Wang *et al.* 2018d 2017]. Despite the great results that have been achieved by scaling deep learning models, mathematical reasoning is an exception [Henighan *et al.* 2020].

Research has developed specialized neural network architectures to explicitly capture the properties that mathematical expressions possess [Xie and Sun 2019; Jie *et al.* 2022]. Typically, the MWP solving task is phrased as a sequence-to-sequence challenge, which requires an encoder to embed the math word query and a decoder to generate a mathematical expression that solves the query. In this section, we discuss several architectures that have been designed to encode (Section 3.8.1) and decode (Section 3.8.2) math word problems and math expressions. We compare these to the Transformer used for our research. The results of these approaches are depicted in Table 4.3 and Table 5.1

3.8.1 Encoder Architectures

The encoder is responsible for encoding the context of the MWP into an embedding space which the decoder can use to generate the answer. The quality of these embeddings is critical for the decoder to generate the correct solutions. Good embeddings should minimize the non-predictive information from the MWPs, and maximize the predictive information that determines whether any two given MWPs have the same solution or different solutions. The proposed deep-learning solutions that aim to learn an embedding space include RNNs, Graph Networks, and Transformers.

Sequential (RNNs) Encoders

Intuitively, an MWP is a sequence of words that with context can be captured using a sequential encoder. Before the proposal of Transformers, LSTMs and GRUs achieved SOTA results for sequential data, which made them the best choice for the MWP task. Wang *et al.* [2017] is one of the earliest works to adapt RNNs to solve MWPs. Compared to Transformers, RNNs require relatively fewer data to learn embeddings, which makes them attractive to use on MWP benchmark datasets. RNNs are still the dominant architectures used in literature [Robaidek *et al.* 2018; Xie and Sun 2019; Hong *et al.* 2021; Wang *et al.* 2018c; Chiang and Chen 2018; Li *et al.* 2019; Su *et al.* 2018].

Transformers make it easier to apply transfer learning compared to RNNs, where pre-trained models can be applied to downstream tasks [Lin *et al.* 2022]. There are not as many pre-trained RNNs that have the notion of language to be used for MWP solving. These models are then trained from scratch, which lead them to overfit since there is only limited data to learn from [Patel *et al.* 2021]. This further motivates the adoption of GRUs over LSTMs, since GRUs have fewer parameters, they are less likely to overfit. However, training from scratch makes it impossible to learn the open MWP space from the setup.

This has led recent research to use pre-trained Transformer embeddings in combination with the RNNs. This is done mainly in two ways: rather than initializing the RNNs with random weights, the embedding layer of a pre-trained Transformer such as Roberta [Liu *et al.* 2019c] or BERT [Devlin *et al.* 2018] is used [Patel *et al.* 2021]. The other approach is to freeze the weights of a pre-trained Transformer, then attach a trainable RNN [Li *et al.* 2021c]. This strategy has been shown to perform significantly better than training from scratch [Patel *et al.* 2021; Wang *et al.* 2022a].

Graph Encoders

Math Word Problems express complex relationships between quantities, objects, and actions that can introduce new quantities, or change current quantities associated with certain objects. These relationships can be modeled using graphs. Sequential encoding from RNNs might struggle to learn these from the limited training datasets. This has led researchers to design architectures that explicitly model these using graphs.

Early implementations created graphs using heuristics [Roy and Roth 2016 2017]. Li *et al.* [2020], and Zhang *et al.* [2020c] independently proposed Graph Neural Net-

works [Hamilton *et al.* 2017; Li *et al.* 2015] to model relations in the MWPs. Li *et al.* [2020] uses a dependency parse tree and constituency tree of text descriptions. The dependency parse tree represents various grammatical relationships between pairs of text words, and also plays a role in transforming texts into logical forms, while the constituency tree contains the phrase structure information which further describes the word relationships.

Zhang *et al.* [2020c] constructs a quantity cell graph to associate descriptive words to quantities, and a quantity comparison graph to retain the numerical qualities of the quantities. Patel *et al.* [2021] showed that initializing these with pre-trained Transformer embeddings has performance gains.

Transformer Encoders

Pre-trained Transformers have been widely adopted for various downstream tasks that tend to have limited data, and MWPs are no exception. Other than the work discussed above that uses the embeddings of the transformer alongside another model, several works use the Transformer encoder as a stand-alone model [Huang *et al.* 2021; Liang *et al.* 2021; Li *et al.* 2021c; Lan *et al.* 2022]. Shen *et al.* [2021] is the first to adopt an encoder-decoder pre-trained Transformer [Lewis *et al.* 2019] to solve MWPs, while most work adopts BERT and RoBERTA models. Transformer encoders already have a notion of language. During fine-tuning, they implicitly learns relations between words and quantities for the MWP task, which lessens the need to explicitly encode these using graphs.

Multiple Encoders

Shen and Jin [2020] proposes using a sequence-based encoder to obtain the context representation of text descriptions alongside a graph-based encoder that integrates the dependency parse tree and numerical comparison information. Yu *et al.* [2021] combines two encoders with a tree-structured decoder. Their first encoder is a pre-trained language model, a source of external knowledge. As for the second encoder, its responsibility is to model the hierarchical context among words and sentences.

3.8.2 Decoder Architectures

The decoder is responsible for generating the math word solution. Traditionally, this is treated as a sequential prediction problem, where the next token is generated as a conditional probability of every token before it and the encoder's hidden states. Although this approach works, it might not necessarily learn the mathematical properties of expressions. As such, invalid mathematical expressions can potentially be produced. Research has proposed neural architectures that traverse mathematical expression syntax trees, either from a top-down or bottom-up approach. We discuss these approaches, alongside sequential decoders in relation to pre-trained decoders that have been used in literature.

Sequential Decoders

Solutions to MWP can be generated sequentially using the prefix, postfix, or infix notation. Early work adopted the sequence-to-sequence architecture to generate mathematical expressions [Chiang and Chen 2018; Wang *et al.* 2017; Li *et al.* 2019]. Attention mechanisms are used in conjunction with the sequential decoder, which usually is a GRU, LSTM, or convolutional network. These networks are simple, usually one or two layers deep to avoid over-fitting and to ease training. Furthermore, these are randomly initialized.

Top-down Tree Decoders

While the early sequence-to-sequence neural networks improved the robustness of automatic MWP solvers, their performance was not satisfying. It is difficult to model the tree-structured relationship of mathematical expressions through sequence during decoding [Xie and Sun 2019]. Liu *et al.* [2019b] and Xie and Sun [2019] proposed tree-structured decoders to generate mathematical expressions. Different from sequential generation, the tree-decoder generates an expression tree in a goal-driven manner based on the parent node and the left sibling tree of each node. It uses top-down goal decomposition and bottom-up subtree embedding to directly predict the expression tree.

The explicit tree-based design rapidly dominated the MWPs community [Zhang *et al.* 2020c; Li *et al.* 2021c; Zhang *et al.* 2020b; Chatterjee *et al.* 2021; Liu *et al.* 2021; Liang *et al.* 2021]. Zhang *et al.* [2020c] replaced the sequential encoder in the architecture proposed by Xie and Sun [2019] with a graph encoder and showed that this led to a significant boost in performance. Similarly, Li *et al.* [2021c] and Liang *et al.* [2021] replace the encoder with a pretrained BERT model and reported performance gains.

Bottom-up Relational Decoders

Different from the tree-based generation models, relational decoders aim to obtain step-by-step expressions using deductive systems [Shieber *et al.* 1995; Nederhof 2003]. Kim *et al.* [2020] designed an expression pointer transformer model to predict expression fragmentation. Cao *et al.* [2021] introduced a DAG model to extract the numerical token relation in a bottom to top manner. Jie *et al.* [2022] treats MWP solving as a complex relation extraction task, where they repeatedly identify the basic relations between different quantities. These yield bottom-up explainable deductive reasoning steps. However, this is computationally extensive, since the relation of all quantities needs to be accounted for.

Transformers

Transformers treat MWP as a sequential decoding task. Different from the sequential models discussed above, Transformers are pre-trained. Noorbakhsh *et al.* [2021] showed that pre-trained language models can be applied to solve integrals and ordinary differential equations trained with limited data. However, training Transformers from

scratch tends to perform poorly with limited data [Patel *et al.* 2021; Chiang and Chen 2018; Nogueira *et al.* 2021; Noorbakhsh *et al.* 2021].

Shen *et al.* [2021] uses the full BART [Lewis *et al.* 2019] encoder-decoder pretrained model. One of the reasons for proposing the tree-decoder was to ensure that valid expressions are generated. Shen *et al.* [2021] found that invalid expressions were produced in less than 1% of the generated expressions. Lan *et al.* [2022] treats the masked language modeling pre-trained models, BERT [Devlin *et al.* 2018] and RoBERTA [Liu *et al.* 2019c], as decoders. Furthermore, they also implement GPT2 [Radford *et al.* 2019], an auto-regressive Transformer to solve MWPs.

Multi Decoders

Similar to the utilization of multiple encoders has found prominence in the literature, the adoption of multiple decoders has also been a subject of investigation and discussion in prior research. Wang *et al.* [2018c] ensembles a Transformer, with two other sequential decoders, one LSTM and the other convolutional. Shen and Jin [2020] uses a sequential decoder alongside a sequential decoder. Zhang *et al.* [2020b] proposed a knowledge distillation approach to create soft labels for two student networks. Meng and Rumshisky [2019] uses two transformer decoders, one does left-to-right decoding while the other does a right-to-left decoding. Zhang *et al.* [2022a] uses a tree decoder and a relational decoder and reports significant gains over using a single model. The goal to automatically solve math word problems is still far from reach, with rule-based and template-based approaches still in use to cover the shortfalls of neural-based approaches [Çelik *et al.* 2022; Lee *et al.* 2021].

3.9 Conclusion

This chapter introduced the fundamental concepts used in this research. Particularly, Section 3.2 explained the Transformer [Vaswani *et al.* 2017] architecture which is the base model used throughout this research. Thereafter, Section 3.3 discusses metric learning, which is used to enhance our multi-view generation task and verification task in Chapters 4 and 5 respectively. Section 3.4 details the pooling mechanism, which allows us to summarize the context of generated sequences that might be of different lengths. The summarized context can then be used to compare and validate math expressions.

We then discussed NLP concepts that need to be considered when setting up our training environment, followed by different learning methods that can be used to improve training. Section 3.7 gives a brief overview of the notations that can be used to write math expressions and the laws that govern them. Section 3.8 relates the Transformer model used in this research to other models in the literature.

In Chapter 4 that follows, we explore how the Transformer performs under different data settings that have been proposed in the literature. This chapter is preliminary to Chapter 5 which proposes a multiview data setting, solved using multiple Transformer decoders, under a multitasking learning paradigm.

Chapter 4

Data Representation and Multi-view Learning

4.1 Introduction

We discussed the Transformer architecture that Vaswani *et al.* [2017] introduced in Section 3.2, which eliminated recurrences in the network and allowed the effective use of parallel computing. This stimulated research to train these models on the vast amount of text data available on the Internet. The results of Devlin *et al.* [2018], Lewis *et al.* [2019], Radford *et al.* [2019], Liu *et al.* [2019c] and Raffel *et al.* [2020] showed that pre-training these models on this large corpus of data in an unsupervised paradigm can improve performance in downstream tasks with limited data.

Pre-trained variants of the Transformer architecture have been used to solve MWPs [Li *et al.* 2021c; Shen *et al.* 2021; Liang *et al.* 2021; Lan *et al.* 2022; Jie *et al.* 2022; Xu *et al.* 2022]. In MWPs, the model is given a math problem and is expected to generate the solution. The task is simplified to the generation of a mathematical expression that evaluates to the numeric value that the MWP queries, rather than directly generating the numeric value. Directly generating the numeric answer frequently performs worse, since it requires the model to perform arithmetic operations in addition to understanding natural language and relational reasoning [Hong *et al.* 2021; Wei *et al.* 2022]. The MWP formulation poses it as a sequence-to-sequence modeling task, motivating the use of pre-trained encoder-decoder Transformers such as BART [Lewis *et al.* 2019] over auto-encoding Transformers such as RoBERTa [Liu *et al.* 2019c] and BERT [Lan *et al.* 2019].

Shen *et al.* [2021] showed that fine-tuning the BART model achieves SOTA performance on the Math23K [Wang *et al.* 2017] benchmark dataset, outperforming other specialized architectures such as Graph2Tree [Zhang *et al.* 2020c], GTS [Xie and Sun 2019], and Multi-E/D [Shen and Jin 2020]. Furthermore, it outperformed architectures that incorporated pre-trained embeddings from BERT and RoBERTa. As a result, we have decided to use BART for our research.

This chapter explores the optimal training setup of the BART model. One of the primary

focuses is on the impact of data representation on the performance of the model. In most proposals, the MWPs are presented to the model with the quantities masked with either a NUM token [Shen and Jin 2020; Xie and Sun 2019] or a NUM_i token [Lan *et al.* 2022; Chiang and Chen 2018], where i is the index of the quantity in the MWP.

Since the value of the quantity does not change the solution template, this abstraction allows the model to reason about the relation of the quantities rather than the values. This also decreases the diversity of expressions, which can lead to faster convergence [Wang *et al.* 2017]. The NUM and NUM_i both have advantages and disadvantages. Eberts and Ulges [2019] and Raffel *et al.* [2020] found that the impact of the indicator in the quantity mask is dependent on the task and dataset. The impact of these masking techniques on MWPs has not been extensively studied.

The recent direction using large language models does not mask the quantities [Wei *et al.* 2022; Wang *et al.* 2022c]. Table 4.1 shows several quantity masks that have been used in previous work. This chapter seeks to empirically determine the impact of how quantities are handled when resolving MWPs. In addition to what has been done in the literature, we propose that the quantities in the MWP are randomly assigned NUM_i, where i is repurposed as an identifier rather than an index of the quantity. See Figure 4.1 for an example. This prevents the model from learning shallow heuristics based on the index. Given that i varies, the model will have to learn to determine the quantity relations in the context to use them appropriately to generate the solution.

Shen and Jin [2020], Zhang *et al.* [2022a] and Meng and Rumshisky [2019] found that generating multiple views of the mathematical expression trees yields better performance than generating a single view. Inspired by this, we investigate whether a Transformer can generate all three views; prefix, infix, and postfix. The sequential decoder in the Transformer is often replaced by a tree decoder [Xie and Sun 2019] or a relational decoder [Cao *et al.* 2021]. These decoders explicitly model the tree structure of expressions. We aim to train the Transformer to learn this structure from the training objective, and its ability to generate all equivalent expressions can better evaluate this.

Furthermore, we propose to align the three multi-view expressions at the token level. This motivates the model to apply the same logic when generating each node of the expression tree, as opposed to merely considering the task as a sequential generation of independent tokens. This also forces the model to learn deeper heuristics to solve the problem, as at each step it must reason about how the token is viewed by the other traversals.

In summary, this chapter explores the impact of data representation on the BART model, focusing on quantity masking and expression notation. The contributions in this chapter are as follows.

- We provide a study that compares different quantity masking techniques used in prior works. This includes the use of no mask, the NUM mask, and the NUM_i mask. In addition to this, we propose a $RAND_i$ mask that randomly assigns the indicator i. To the best of our knowledge, the impact of these approaches has not been studied.

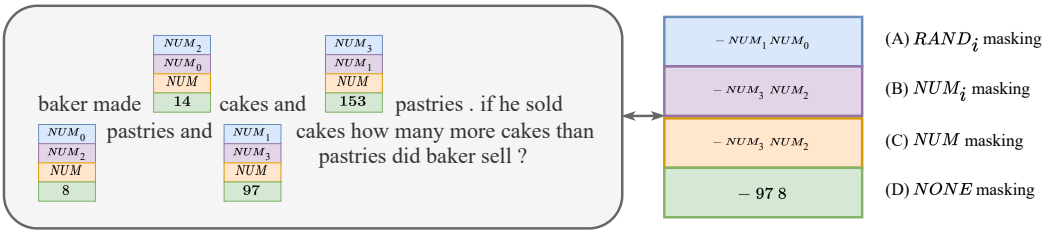


Figure 4.1: Examples of different quantity masking techniques. Examples adapted from SVAMP [Patel *et al.* 2021].

- We propose a multi-view generation objective, where the seq2seq Transformer has to generate the prefix, infix, and postfix views of the expression tree instead of one. We propose a concatenated multi-view generation task, where the Transformer has to generate the prefix, infix, and postfix traversals of the expression tree.
- We show that the multi-view generation task can be improved by adding a view generation consistency objective that aligns tokens that represent the same node in the expression tree across the different views.
- Experiments on the SVAMP [Patel *et al.* 2021] and MAWPS [Koncel-Kedziorski *et al.* 2016] benchmark datasets show that the indicator i is critical for the model's ability to learn MWPs. Furthermore, we show that randomizing the assignment of indicators improves the learned heuristics of the model. In addition, training BART to generate multiple notations improves its understanding of the MWP task compared to training it to generate a single notation.

The rest of the chapter is structured as follows: Section 4.2 gives a brief survey on literature that explored data representations for MWPs. This is followed by the methodology for the chapter in Section 4.3. We discuss experimental results and ablation studies in Section 4.4. Section 4.5 concludes the chapter, with a summary and direction for future work.

4.2 Related Work

Data representation plays a critical role in how models perform. For example, Nogueira *et al.* [2021] found that representing numbers as characters, substrings, or simply adding underscores between them greatly impacts the ability of the Transformer to extrapolate. Similar results have been found in other works [Muffo *et al.* 2022; Wallace *et al.* 2019]. The prefix and postfix notations have the same tokens in different orders; however, Nogueira *et al.* [2021] and Kubota *et al.* [2022] found the postfix notation to perform better than the prefix notation for solving MWP and symbolic mathematics, respectively. This inspires us to explore how data representation influences the BART model. In this section, we discuss previous work that relates to ours. The results of these methods can be found in Table 4.3 and Table 5.1.

Model	Notation	Q.Mask	Type
DNS [Wang *et al.* 2017]	infix	NUM_i	seq2seq
Transformer [Lan *et al.* 2022]	infix	NUM_i	seq2seq
BERT-BERT [Lan *et al.* 2022]	infix	NUM_i	plm
RoBERTa-RoBERTa [Lan *et al.* 2022]	infix	NUM_i	plm
mBART [Shen *et al.* 2021]	infix	NUM_i	plm
Gen&Rank [Shen *et al.* 2021]	infix	NUM_i	plm
Math-EN [Wang *et al.* 2018c]	postfix	NUM_i	seq2seq
S-Aligned [Chiang and Chen 2018]	postfix	NUM_i	seq2seq
T-RNN [Wang *et al.* 2019a]	postfix	NUM_i	seq2tree
Group-ATT [Li *et al.* 2019]	postfix	NUM_i	seq2tree
RE-Deduction [Jie *et al.* 2022]	DAGs	NUM_i	seq2tree
GTS [Xie and Sun 2019]	prefix	NUM	seq2tree
SAU-Solver [Qin *et al.* 2020]	prefix	NUM	seq2tree
GSGSF-6L [He and Xiao 2022]	prefix	NUM	seq2tree
Graph2Tree [Zhang *et al.* 2020c]	prefix	NUM	graph2tree
TeacherSup [Zhang *et al.* 2020b]	prefix	NUM	graph2tree
MultiE&D [Shen and Jin 2020]	prefix	NUM	graph2tree
EEH-G2T [Wu *et al.* 2021b]	prefix	$NONE$	graph2tree
MultiView [Zhang *et al.* 2022a]	prefix	NUM_i	seq2tree
RoBERTa-GTS [Patel *et al.* 2021]	prefix	NUM_i	seq2tree
CLRSolver [Wu *et al.* 2022]	prefix	NUM_i	graph2tree
CogSolver [Liu *et al.* 2022]	prefix	NUM_i	graph2tree
TeacherSup [Liang and Zhang 2021]	prefix	NUM_i	graph2tree
RPKHS [Yu *et al.* 2021]	prefix	NUM_i	graph2tree
RoBERTa-Graph2Tree [Patel *et al.* 2021]	prefix	NUM_i	graph2tree

Table 4.1: Different notations and quantity masks used to solve math word problems. By enabling the quantity mask to randomly shuffle, the model becomes less sensitive to the mask and focuses more on the context the mask appears in.

4.2.1 Mathematics Expression Notation

Prefix Notation

Mathematical expressions can be represented using three main notations: prefix, infix, and prefix (discussed in Section 3.7.2). The prefix notation is the most widely used in the literature [Lample and Charton 2019; Xie and Sun 2019; Zhang *et al.* 2020c; Wang *et al.* 2017; Xiong *et al.* 2022ba]. Lample and Charton [2019] proposed the first neural symbolic solver based on Transformers. They solve integrals and ordinary differential equations using the prefix notation, outperforming MATLAB [The MathWorks Inc. 2019] and Mathematica [Wolfram Research Inc. 2019].

Wang *et al.* [2017] proposed the first neural math word problem solver, also using the prefix notation. Work that followed on this uses the same notations [Noorbakhsh *et al.* 2021; Wang *et al.* 2018c]. The prefix notation is often preferred since it eliminates brackets, which makes the solutions shorter and spares the model from potential mistakes of misaligned brackets.

Infix Notation

Though the infix notation tends not to be preferred when solving math problems, it is the notation we are more used to. Saxton *et al.* [2019] proposed an infix notation mathematics dataset consisting of many different types of mathematics problems, covering arithmetic, algebra, probability, and calculus. Hendrycks *et al.* [2021] proposed the MATH dataset, also presented in infix notation. Lan *et al.* [2022] and Shen *et al.* [2021] use the infix notation to fine-tune pre-trained Transformers.

Postfix Notation

Although postfix notation and prefix notation have the same advantages over infix notation, the postfix notation is frequently overshadowed by prefix notation. Kubota *et al.* [2022] argues that the prefix notation makes it difficult to learn the syntactic structure of mathematical expressions because the operators and the operands are far from each other. Their experiments showed that the postfix notation performs better than the prefix notation for symbolic reasoning. Griffith and Kalita [2020] found that the postfix representation outperforms the prefix representation for MWPs. Wang *et al.* [2018c] adopted the postfix notation to solve MWPs.

Directed Acyclic Graphs (DAGs)

Jie *et al.* [2022], Li *et al.* [2022a], Cao *et al.* [2021] proposed to use a Directed Acyclic Graph (DAG) representation of the expressions which is fundamental to the bottom-up decoder. Kubota *et al.* [2022] solve symbolic mathematics using the DAG representation through a sequential decoder, where they introduce a token that demarcates the end of the sub-tree. DAGs allow the model to reuse the results from the intermediate sub-trees.

Multiple Notations

Shen and Jin [2020] solve MWPs using two decoders, a postfix notation sequential decoder, and a prefix notation tree-decoder. Zhang *et al.* [2022a] also uses two decoders to solve MWPs, one relational decoder using a DAG representation and one tree-decoder with a prefix representation. Meng and Rumshisky [2019] uses two sequential decoders that both generate the prefix notation, with one decoder generating it from left to right and the other decoder generating it from right to left. The multi-notation setup has been shown to perform better than a single notation setup. We propose to generate the prefix, infix, and postfix notations in our multi-view setup. All of these are generated using a single sequential decoder.

4.2.2 Conditional Generation

Research in Natural Language Processing (NLP) has successfully used one model to perform several tasks. Raffel *et al.* [2020] and Tang *et al.* [2022] use a single decoder to learn multiple NLP tasks, including machine translation, text summarization, question answering, and classification. They prepend a soft prompt to the inputs that indicate the task the model should perform. Similarly, Radford *et al.* [2018], Radford *et al.* [2019] and Dai *et al.* [2019] use the last token in the input as a prompt as to what to generate.

Liu *et al.* [2020b] and Fan *et al.* [2021] do a many-to-many machine translation by enforcing the last token in the inputs to the encoder to represent the source language and the first generated token from the decoder to represent the target language. Similarly, we enforce that the first token in our shared decoder architecture represents the target expression notation that the model should generate.

4.2.3 Quantity Masking

The quantities in MWPs are often masked, which simplifies the MWP task to a task of mapping an MWP to an expression template. This shortens the expressions generated by the decoder when the quantities are large. The masks for the quantities can either have an indicator for the index [Lan *et al.* 2022; Chiang and Chen 2018; Wang *et al.* 2018d; Li *et al.* 2019; Wang *et al.* 2018c], or they might not [Shen and Jin 2020; Xie and Sun 2019; Qin *et al.* 2020; Zhang *et al.* 2020bc; Li *et al.* 2021c; Liang *et al.* 2021].

In contrast to the tree and relational decoders, sequential decoders need not be masked. The proposed non-sequential decoders expect the quantity as a single token, which without the mask would require every quantity to be in the vocab, which is impractical. Wu *et al.* [2021c] argues that masking ignores the prominent role of numerical values when solving MWPs, and proposes not to mask the quantities. The recent direction taken by research that uses large pre-trained Transformers does not mask the quantities [Wang *et al.* 2022c; Wei *et al.* 2022; Cobbe *et al.* 2021; Li *et al.* 2022b; Magister *et al.* 2022].

4.2.4 Chain of Thought

Wei *et al.* [2022] proposed a chain of thought prompting, where a language model is prompted to generate a series of short sentences that mimic the reasoning process a person might employ to solve a reasoning task. They showed that chain of thought prompting significantly improves language model performance across a variety of multi-step reasoning tasks, including MWPs. However, they also found that chain-of-thought prompting does not positively impact performance for small models, and only yields performance gains when used with models of more than 100B parameters. We use a relatively small model with less than 1B parameters.

4.2.5 Generation Consistency

Li *et al.* [2021c] applies a contrastive consistency objective to the samples in a batch, where they treat trees or sub-trees that have the same operators as positive examples, and trees with different operators or sizes as negative examples. However, this only uses a single view. Similar to Li *et al.* [2021c], Zhang *et al.* [2022a] applies a contrastive consistency objective at a sub-tree level, however, between two views from two different decoders. Different from these, our alignment is at a token level across three views that are generated from a sequential decoder. The tree and relational decoders proposed above are hierarchical in nature, with each node containing information about its children. Our sequential decoder is flat, enabling us to apply token-level alignment.

4.3 Methodology

In this section, the methodology utilized in this chapter is described. We formalize the definition of the MWP task before providing a brief overview of the BART model. The various quantity masks used are then discussed and motivated. The setup of the various notations we use is then discussed, with particular emphasis on the multi-view setup. Thereafter, we discuss the proposed generation objective for view consistency. Finally, the overall training objective is presented. The proposed multi-view model is depicted by Figure 4.4.

4.3.1 Problem Statement

An MWP P consists of a sequence of tokens that represent the background, quantities, and query. The background establishes a relationship between the quantities, and the query seeks to know a relationship between the quantities that can be represented by a mathematical expression. Let the words in P forming the background and query be denoted by $W_P = \{w_1, w_2, ..., w_n\}$ and quantities found in P be denoted as $Q_P = \{q_1, q_2, ..., q_k\}$. The MWP queries about an unknown quantity Q_u, which is given by evaluating a target mathematical expression $Y = \{y_1, y_2, ..y_m\}$.

The target Y is made up of the quantity set Q_P, the operator set $O = \{+, -, *, /\}$, and constant quantities Q_C that may not necessarily appear in the MWP (such as $\pi = 3.14$,

$half = 0.5$, $week = 7$, etc.). The MWP task is to develop a model $F : P \mapsto \hat{Y}$ which produces an expression $\hat{Y}$ that generates the correct solution Q_u for the problem P.

4.3.2 Model Architecture

All of our experiments are carried out with BART [Lewis *et al.* 2019], a bidirectional and auto-regressive sequence-to-sequence model based on the Transformer [Vaswani *et al.* 2017] architecture. The encoder of BART is bidirectional [Devlin *et al.* 2018] with a completely visible attention mask. The decoder follows the standard left-to-right language modeling task with a causal attention mask [Radford *et al.* 2018]. BART is pre-trained to reconstruct the original text that was corrupted by an arbitrary noise function. This model has been shown to be effective for text generation, comprehension, and classification tasks.

Encoder

The structure of BART can be divided into two major functions: encoding and decoding. The encoder takes in a sequence of discrete tokens $P = (p_0, p_2, ..., p_N, p_{N+1})$ where $p_i \in \mathbb{Z}^+$, N is the number of tokens in the MWP, and p_0 and p_N denote the beginning-of-string token $[bos]$ and end-of-string token $[eos]$ automatically added to MWPs. The encoder generates a sequence of continuous representations $E = (e_0, e_1, ..., e_{N+1})$ where $e_i \in \mathbb{R}$. This contextualized representation is then passed to the decoder. We denote the mapping function of the BART encoder as follows:

$$E = (e_0, e_1, ..., e_{N+1}) = F_E(p_0, p_1, ..., p_{N+1}) \tag{4.1}$$

Decoder

In BART, the decoder takes the contextualized representation E of the encoded input text and generates an output sequence $D = (d_0, d_1, d_2, ..., d_M, d_{M+1})$, where $d_i \in \mathbb{R}$, M is the number of tokens in Y, d_0 is the encoded representation of the input start-of-decoding token, and d_{M+1} is the encoded representation of the generated end-of-decoding token. The decoder outputs pass through a linear layer f attached to a softmax function to generate the prediction of the target expression $\hat{Y} = (y_0, \hat{y}_1, \hat{y}_2, ..., \hat{y}_M, \hat{y}_{M+1})$. The decoder generates each output one at a time conditioned on the encoder outputs, the start-of-decoding token, and the tokens the decoder has already generated.

The decoding process can be denoted as follows:

$$\begin{aligned} \mathcal{P}(Y|P) &= \prod_{i=1}^{M+1} \mathcal{P}(\hat{y}_i|P, \hat{y}_{i-1}) \\ \mathcal{P}(\hat{y}_i|P, \hat{y}_{0:i-1}) &= \text{softmax}(f(d_i)) \\ d_i &= F_D(E; (y_0, \hat{y}_1, \hat{y}_2, ..., \hat{y}_{i-1})) \end{aligned}$$

where y_0 and y_{M+1} are special tokens ($[bos]$ and $[eos]$) indicating the start-of-decoding and end-of-decoding, respectively; $\hat{y}_{0:i-1} = (y_0, \hat{y}_1, \hat{y}_2, ..., \hat{y}_{i-1})$; E is the output of the encoder given by Equation (4.1); f is a linear layer that maps the dimension of each

decoder state d_i to the dimension of the output vocabulary. The output from the linear layer f represents the unnormalized predictions (logits), which are then normalized using the softmax function.

During training, the BART model is optimized to minimize the cross-entropy loss between the predicted sequence $\mathcal{P}(\hat{Y}|P)$ and the ground truth sequence Y. This can be denoted by the following equation:

$$\mathcal{L}_{GEN} = \sum_{P} -\log \mathcal{P}(\hat{Y}|P) \tag{4.2}$$

4.3.3 Quantity Masking

We experiment with different ways to represent the quantities Q_P. The quantities can be masked or unmasked. Figure 4.2 illustrates the examples of these. When the quantities in P are unmasked, the model has to generate an expression using these explicit quantities. This does not require any pre-processing or post-processing in order to be evaluated. Using the explicit quantities can also be beneficial when the quantities have some semantic value, for example, finding the next value in a sequence.

Alternatively, the quantities in P can be masked and the model would need to generate a template expression that evaluates to Q_u when the quantities in P are substituted. Masking the quantities reduces the number of tokens that need to be generated, and also helps the model learn to map P to more similar equation templates. This simplifies the problem, leads to faster convergence, and has reduced time complexity.

Unmasked Quantities

The initial approach is to have the set of quantities in P not be masked. Several studies have shown that character-level tokenization is more efficient than sub-word tokenization used by the BART tokenizer [Henighan *et al.* 2020; Nogueira *et al.* 2021]. Given that we are not performing any arithmetic operations, we use the pre-trained BPE tokenizer that BART used during pre-training. The sub-word significantly increases the number of potential tokens while also lowering the number of tokens that would result from character-level tokenization. Since the decoder only needs to learn to copy the correct quantities from the encoder, the search space for the decoder does not significantly expand. A challenge when quantities are not masked is that it can be hard to distinguish numbers, as can be seen in Figure 4.2 (I). The model has no way of telling which of the 4s in the MWP is the ground truth referring to.

NUM Quantity Mask

Masking can be done by replacing each quantity in P with a NUM token during pre-processing. The model is requested to generate Y, where each quantity has a quantity identifier i that indicates its index in Q_P. Since the model is not presented with i, it must learn a mapping of i in the ground truth to the correct quantity index in Q_P. This adds an extra complexity that might not benefit learning.

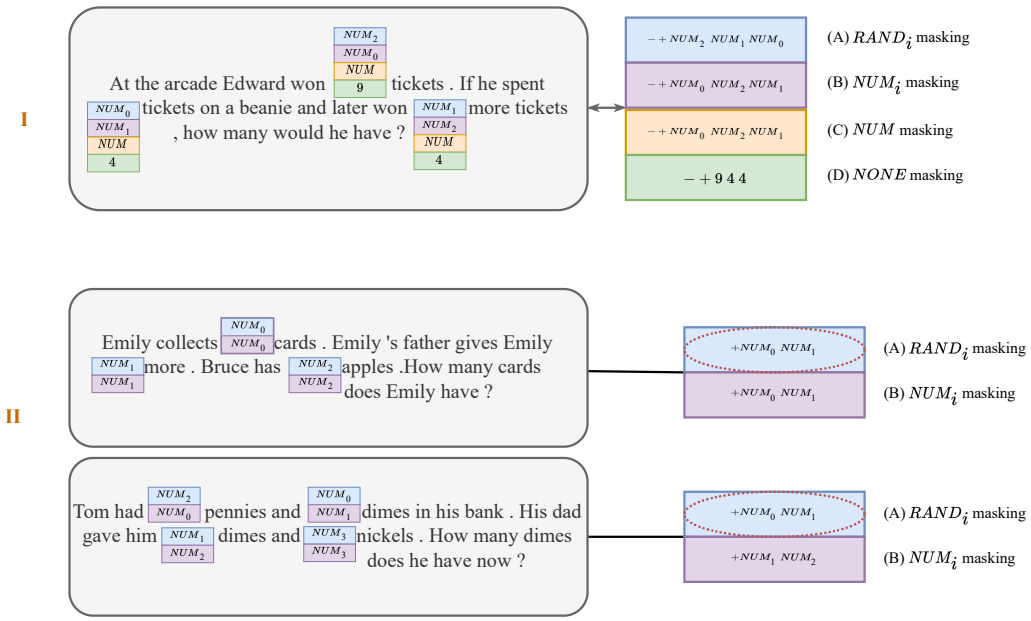


Figure 4.2: Examples of different quantity masking techniques. Examples adapted from SVAMP [Patel *et al.* 2021].

NUM_i Index Quantity Mask

Another approach to quantity masking is to all the quantities with NUM_i, where i represents the index of each quantity in Q_P. This removes the burden of counting the NUM tokens in P to find the index i. The counting task has been shown to be hard for Transformers [Nogueira *et al.* 2021].

NUM_i Random Identifier Quantity Mask

We propose an improvement over the NUM_i masking technique, where we randomly assign the mask identifier i, rather than having it tied to the i^{th} index. This prevents the model from learning shallow heuristics associated with the identifier i and challenges it to reason more about the context in which the i^{th} mask appears. This helps the model consolidate similar expressions regardless of the indices and also helps prevent an index bias from developing. Figure 4.2 (II) depicts an example of this. If the solution to several MWPs is the addition of two quantities, this strategy can help the model learn that the MWPs are the addition of two quantities and not of quantity index i_a and index i_c. During inference, the quantities are indexed in order, which is equivalent to NUM_i masking.

4.3.4 Solution Notation

The solution to MWP Y is represented by an abstract syntax tree (AST). This tree can be represented using the prefix, infix, and postfix notations. This section describes how a single decoder is used to generate the three views. Specifically, we investigate two methods for inducing our model to generate multiple views. The first method uses a

view target prompt, and the second method concatenates the views.

Prefixed Multi-view Notation

Given an MWP P and the expression tree Y that solves P, we generate three views $V = \{\textit{prefix, infix, postfix}\}$. For each MWP P the model is tasked with predicting all three views $\hat{Y}^{(V)}$ that evaluate to the same solution as the ground truth expression $Y^{(V)}$. We can inform the model which views to generate by adding a view prompt to either the encoder or the decoder. Raffel *et al.* [2020] prepends a prompt to the inputs to the encoder which conditions their multi-tasking model to a particular task. This allows the encoder to produce different encodings based on the task. However, in our case, we want the model to encode the MWP to be such that it represents an expression tree that the decoder can decode in three views. To accomplish this, the prompt is added to the decoder.

We adopt the strategy proposed in Liu *et al.* [2020b], where we force the target view prompt to be the first token generated by the decoder, instead of the start of sentence token $[bos]$ token. Since the decoder is left to right, every generated token will be conditioned on this token. In theory, any token that does not appear in the ground truth expression can be reserved for this task. We can either add new tokens to the BART vocabulary that we will train for this task or repurpose existing tokens. Since BART already has a large vocabulary, we decide to use the second option. Our specific prompts for the prefix, infix, and postfix views are pre, inf, and $post$, respectively. The standard special token representation $[\cdot]$ would produce three tokens, whereas each of these is one token.

During training, the decoder receives three view prompts for each MWP that the model encodes. That is, the model needs to generate $\hat{Y}^{(v)} \in \hat{Y}^{(V)}$ for each P. This produces three losses per MWP. We take the average of these. As a result, the final loss becomes:

$$\mathcal{L}_{GEN}^{(v)} = \sum_{P} -\log \mathcal{P}(\hat{Y}^{(v)}|P)$$
$$\mathcal{L}_{GEN}^{(V)} = \frac{1}{|V|} \sum_{v \in V} L_{GEN}^{(v)}$$

where V is the set of all views and L_{GEN}^{v} is the loss of view v, and $\hat{Y}_0^{(V)} = \{pre, inf, pos\}$ is used to condition the generation of $\hat{Y}^{(V)}$. The target view token is not included during the loss computation. During inference, we prompt the model to generate the postfix view, based on empirical studies and literature showing that this outperforms the other views [Griffith and Kalita 2020; Kubota *et al.* 2022]. The other views are used as auxiliary tasks, enhancing the model's generative capabilities for the postfix view.

Concatenated Multi-view Notation

Similar to the prefixed multi-view generation objective, concatenated multi-view notation requires the decoder to generate three views; however, in this case, the decoder must generate all three views as one sequence. We introduce a separator token # that

separates the different views. This is a similar concept to the $[sep]$ token in BERT [Devlin *et al.* 2018], except that it is applied to the decoder. The views are ordered based on the difficulty demonstrated in previous work: postfix, prefix, and infix [Griffith and Kalita 2020]. During inference, we use the separator token to split the generated expression into distinct views. The loss function can be denoted similarly to Equation (4.2), where $Y = \text{concat}(Y^{(V)})$ divided by the separator token.

4.3.5 Generation Consistency

The decoder is trained to traverse the same expression tree in different notations. The operands and operators that are generated should be consistent regardless of the generation view since they represent the same node in the expression tree. Hence, we apply a contrastive objective between the generated states D_t of the views to encourage the decoder to learn from the views simultaneously. Figure 4.3 shows our token alignment strategy.

To align the different views, we keep track of the node index t of each of the operands and operators on the target expression tree T while traversing to generate multiple views. We take the decoder states D generated by the model and add a token-level contrastive learning objective that maximizes the cosine similarity as follows:

$$\mathcal{L}_C(D^{(v)}, D^{(w)}) = \frac{1}{|T|} \sum_{t \in T} 1 - \frac{D_t^{(v)} D_t^{(w)}}{||D_t^{(v)}|| \cdot ||D_t^{(w)}||}$$

$$\mathcal{L}_{CON}^{(V)} = \frac{1}{|V| - 1} \sum_{i=2}^{|V|} \mathcal{L}_C(D^{(V_1)}, D^{(V_i)}) \tag{4.3}$$

where D is the output state of the decoder before the linear transformation, T is the expression tree for the problem P, and t is a node index in the expression tree, which could have different indices depending on the view. The brackets are not aligned for the infix view. As a result of the symmetry in the cosine similarity, we minimize the pairwise cosine distance between the first view and each subsequent view. This can account for all other potential pairwise similarities, as a result, they do not add any new information or constraints to the loss function.

4.3.6 Training

Given the training dataset $A = \{(P_1, T_1), (P_2, T_2), ..., (P_N, T_N)\}$, where T_i is the desired expression tree of problem P_i, and T_i is presented to the model through multiple views V, we minimize the generative loss and the view-consistency loss simultaneously:

$$\mathcal{L} = \sum_{(P,T) \in A} \mathcal{L}_{GEN}^{(V)} + \alpha_{CON} \mathcal{L}_{CON}^{(V)} \tag{4.4}$$

where α_{CON} is a hyper-parameter that weighs the contribution of the view-consistency generation loss.

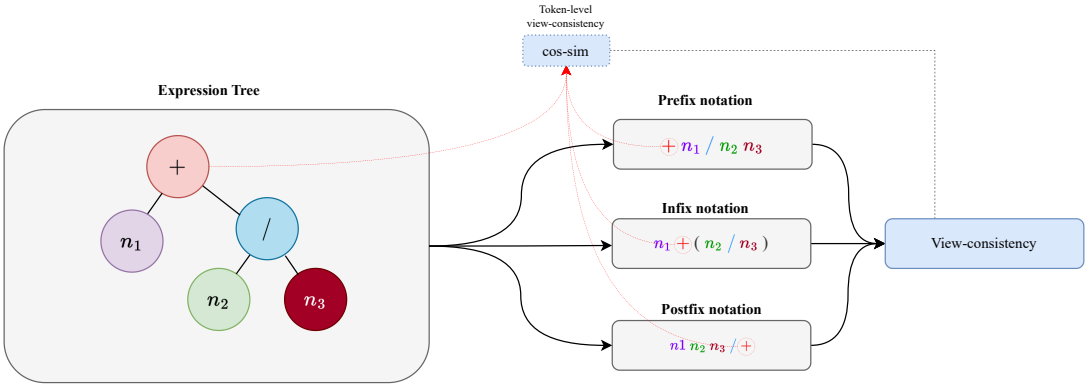


Figure 4.3: The token-level view-consistency strategy proposed. We take the decoder hidden states $D^{(V)}$, and we maximize the cosine-similarity between $d_i^{(v)}$ that represents the same node in the expression tree. The $+$ token appears in different indices i per view v, however, it is indexed accordingly per view maximizing the cosine-similarity during training.

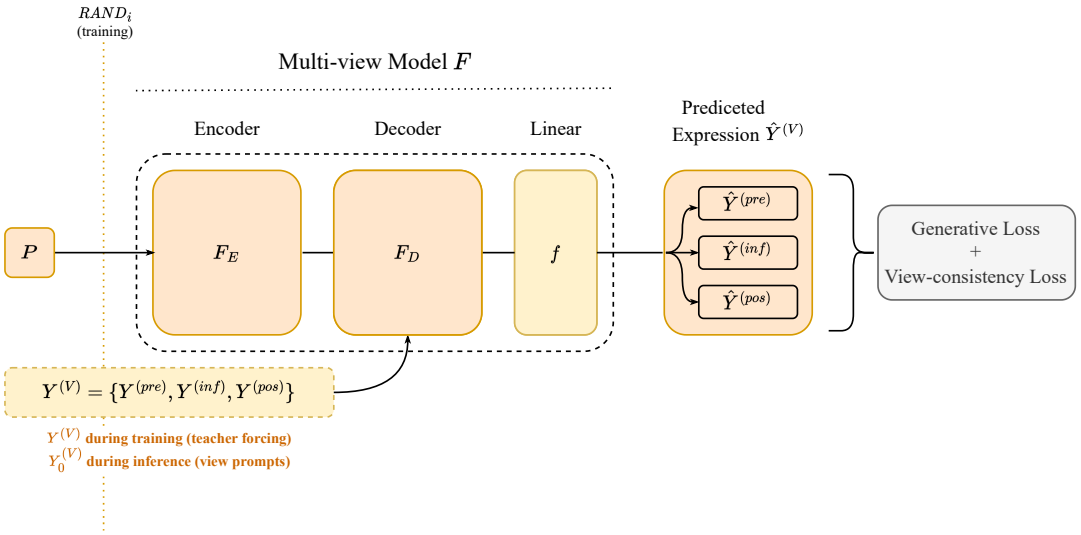


Figure 4.4: Our proposed multi-view BART model. The model generates a contextualized representation E of a given MWP P using the encoder F_E of the model. The decoder F_D trains through teacher forcing using E and $Y^{(v)}, v \in V$, generating one view v at a time. The training objective is to minimize the generative loss and the view-consistency loss. During inference, the decoder generates the prediction $\hat{Y}^{(v)}$ using E and a view prompt $Y_0^{(v)}$.

Dataset	#Train	#Valid	#Test	#Const	Avg. Len
MAWPS	1,589	199	199	17	30.3
SVAMP	3,138	-	1,000	17	34.7

Table 4.2: Dataset statistics.

4.4 Experiments

4.4.1 Datasets

We conduct our experiments on two widely used datasets: MAWPS [Koncel-Kedziorski *et al.* 2016] and SVAMP [Patel *et al.* 2021]. The statistics of the data set can be found in table 4.2. All problems in MAPWS are linear problems with one unknown variable that can be solved with a single expression. The 1000-MWP challenge test set for SVAMP consists of nine adversarial variations on 100 MWPs. The variations are applied to assess three model properties: sensitivity to questions, reasoning ability, and structural invariance.

Larger datasets exists in literature, including Math23K [Wang *et al.* 2017], MathQA [Amini *et al.* 2019], and GSM8K [Cobbe *et al.* 2021]. Math23K is not English, while MathQA and GSM8K contain more complex questions (including geometry, trigonometry, calculus, and statistics) which are beyond the scope of this research. SVAMP and MAWPS are widely used datasets, and we choose these following previous literature [Lan *et al.* 2022; Patel *et al.* 2021; Jie *et al.* 2022; Zhang *et al.* 2020c].

4.4.2 Baseline

This chapter concerns itself with finding the optimal quantity masking and expression notation for pre-trained language models (PLMs), particularly, Transformers. On this note, we only consider Transformers for a fair comparison. However, we include the state-of-the-art models to provide a benchmark. To avoid the implementation error that may cause unreproducible results of baseline models, we report the results of these baselines from the papers publications, as many previous papers have done [Zhang *et al.* 2020c; Shen and Jin 2020; Liang *et al.* 2021].

4.4.3 Transformers

We include the results from Griffith and Kalita [2020] who trained their Transformer from scratch, and to a maximum of four layers. They used the prefix, infix, and postfix notations. We consider the best results across their three Transformer architectures. We also include results from Lan *et al.* [2022] who trained the larger Transformer model from scratch using twelve layers in the encoder and decoder.

Pre-trained Language Models (PLM)

We consider PLMs to be purely pre-trained models, similar to Lan *et al.* [2022]. RoBERTa-RoBERTa [Lan *et al.* 2022] uses the pre-trained RoBERTa [Liu *et al.* 2019c] model as the encoder and decoder. BERT-BERT [Lan *et al.* 2022] uses the BERT [Devlin *et al.* 2018] model that was pre-trained with an auto-encoding objective as the encoder and decoder. mBART [Shen *et al.* 2021] uses multi-lingual BART [Liu *et al.* 2020b], a sequence-to-sequence denoising auto-encoder that has been pre-trained on massive monolingual corpora in a variety of languages using the BART objective. Gen&Rank [Shen *et al.* 2021] is a multi-task framework that generates multiple candidate solutions for MWPs, ranks them, and returns the highest ranked as the final solution.

State-of-the-Art (SOTA) Models

The current SOTA models include: RoBERTa-GTS [Patel *et al.* 2021], RoBERTa-Graph2Tree [Zhang *et al.* 2020c; Patel *et al.* 2021], MultiView [Zhang *et al.* 2022a] and Re-Deduction [Jie *et al.* 2022]. Patel *et al.* [2021] initializes the embeddings of the GTS Xie and Sun [2019] and Graph2Tree [Zhang *et al.* 2020c] model with RoBERTa embeddings. The RoBERTa-GTS and RoBERTa-Graph2Tree hold the SOTA for seq2tree and graph2tree architectures, respectively. Re-Deduction recently outperformed Graph2Tree with a full RoBERTa encoder and a novel deductive reasoning decoder that extracts relations between quantities in a given context. The MultiView model builds on Re-Deduction by introducing a tree decoder alongside the deductive decoder, achieving even better results.

4.4.4 Training Details

We use the pre-trained weights of BART-large ("facebook/bart-large") and implementations of BART offered by the Transformers library[1]. We utilize the PyTorch[2] framework for all our experiments. The loss is optimized with the Adam optimizer [Loshchilov and Hutter 2017; Kingma and Ba 2014] for 50 epochs. We use the following training hyper-parameters: a batch size of 32, a learning rate of $3e-5$, a warm-up ratio of 0.1, a weight decay of 0.01, dropout of 0.1, and a weight of $\alpha_{CON} = 2$ for the generative consistency loss objective. We run the model for 50 epochs. We fine-tune our models independently on the datasets used.

4.4.5 Evaluation Metrics

We report the accuracy percentage of our experiments in accordance with previous works [Wang *et al.* 2017; Shen *et al.* 2021; Zhang *et al.* 2022a; Jie *et al.* 2022]. Specifically, we consider a generated expression to be correct if it evaluates an answer that is equal to the corresponding ground truth answer. We use the answer rather than the

[1] https://github.com/huggingface/transformers
[2] https://pytorch.org/

ground truth expression due to the mathematical laws (association, distribution, and commutativity) governing the mathematical expression. We report the average after five repetitions of our experiments using various seeds. We report on the 5-fold cross-validation sets for the MAWPS dataset. The ablation studies are conducted on the test set for SVAMP and 5-fold cross-validation for MAWPS, similar to previous works [Wang *et al.* 2017; Shen *et al.* 2021; Zhang *et al.* 2022a; Jie *et al.* 2022]. Unless otherwise specified, we use the postfix-view [Griffith and Kalita 2020] to report the final results for multi-view settings.

4.4.6 Overall Results

Our model's evaluation results and baselines are summarized in Table 4.3. We report the results of our proposed methods, which include (i) randomizing the NUM_i token; (ii) generating multiple views by using the first token in the decoder generation as a prompt; and (iii) enforcing the decoder to be consistent when generating multiple views.

Model Comparison

From the results, we observe that BART outperforms the pre-trained models that have been used in the literature on MAWPS and SVAMP, including mBART. The mBART model has the same architecture as our BART, only differing on the word embedding sizes and pre-training objectives, with mBART pre-trained on multi-lingual corpora. This could suggest that the BART model benefits from the English pre-training objective. However, the results are not conclusive given that the Transformer model without pre-training outperforms the mBART model. This suggests that further hyper-parameter tuning might need to be done on mBART to realize its full potential.

BART outperforms BERT and Roberta-based models (presented in Table 4.3) which were pre-trained with an auto-encoding objective. This suggests that BART's sequence-to-sequence objective is more appropriate for the MWP task, and the natural language understanding that the former models possess is not sufficient.

We also find that the BART model does not suffer from invalid expressions across all notations, similar to Shen *et al.* [2021] and Lample and Charton [2019]. This suggests that we do not explicitly need to use a tree decoder in order to enforce correctness. Particularly, during generations, we observe that the Transformer first learns the notation, before it starts to generate the correct solutions. However, we note that the results we get from BART are less than what SOTA models get using tree decoders, which suggests that these decoding mechanisms capture more structural information than just enforcing correct expression generation. This also motivates the use use of extra objectives than the generative objective that BART uses in order to capture more information.

NUM Quantity Masking

To evaluate the effectiveness of the $RAND_i$ masking that we proposed, we evaluate it against $NONE$, NUM, and NUM_i masking. We observe that NUM masking performs

worse than $NONE$ masking in SVAMP, however, it performs better in MAWPS. We hypothesize that this is due to the quantity sensitivity introduced in SVAMP, where extra quantities that are not relevant to the question are added to the MWP. This result shows that quantity masking also depends on the dataset.

$NONE$ Quantity Masking

We also note that not masking the quantities, that is, $NONE$ masking performs worse than NUM_i masking. One possible explanation is the increased search space, that is, the model needs to learn more tokens to produce the same equation template as NUM_i. This makes it harder to consolidate all the multiple expressions that result. Furthermore, several MWPs have repeating quantities, which makes it hard for the model to learn which quantity in the MWP is referred to by the ground truth.

NUM_i Quantity Masking

We observe that NUM and $NONE$ masking performs worse than the quantity indicator masking techniques, NUM_i and $RAND_i$. This supports our hypothesis that the BART model benefits from knowing the location of the quantity rather than having to learn it.

The NUM_i assumes that each quantity is unique, thus easily indicating to the model which quantity in the MWP results in the correct ground truth expression. Our results validate this hypothesis. Particularly, we see a 5.7% and 10.3% relative improvement in SVAMP and MAWPS respectively, when we apply NUM_i masking.

$RAND_i$ Quantity Masking

As an improvement to NUM_i masking, we propose $RAND_i$ masking. Our results show that this masking strategy improves performance across multiple datasets and all notations. The relative improvements from this simple strategy for prefix, postfix, and infix notations are 0.7%, 1.4%, and 0.3% for SVAMP and 1.6%, 0.9%, and 0.7% for MAWPs, respectively.

Interestingly enough, we notice that the infix notation benefits the most from the $RAND_i$ masking. This suggests that the infix notation's performance is likely due to the quantity masking rather than the notation itself. We see that the infix notation outperforms the other views with an absolute maximum accuracy of 0.3%.

We also observe that the $RAND_i$ masking closes the performance gap between the different notations. $RAND_i$ is the least beneficial to the postfix notation. This could be due to the way the decoding process works. The postfix traversal needs to analyze the quantities before it generates the operator. This gives postfix an advantage over the prefix and infix notations which often produce NUM_i mistakes. Given the positive improvements on $RAND_i$, we run all our multi-view experiments with the $RAND_i$ mask.

Single-View Expression Notation

Our results show that the model learns different heuristics to solve MWPs based on the decoding notation used in the ground truth. We note that the notation performance is consistent with the results reported from Griffith and Kalita [2020], who found that postfix notation outperforms the other two notations across multiple datasets, while prefix tends to outperform the other notations when only observed from a single dataset. They further showed that the model architecture affects the performance of the model over the learning notation.

Our results show that, for the BART architecture, the postfix outperforms the other two notations on individual datasets. That is, the BART model can better use the quantities in the encoder to generate the correct operator than to first generate an operator followed by the correct quantities, or to iterate between quantities and operators. This raises the possibility that the results reported in the literature are not fully indicative of what the models are capable of, and it further supports the case for using the postfix notation in subsequent studies.

Contrary to Griffith and Kalita [2020], we find that the infix notation does not lag too far behind the other notations. This is likely a result of the pre-training objective that reduces the burden of having to learn the more difficult infix notation. The increase based on $RAND_i$ suggests that the infix notation might learn heuristics that are sufficient to generate the correct expression, then learn shallow heuristics that

Multi-view Expression Notation

As a strategy to force the BART model to learn more structural information related to math expressions, we task it to generate multiple views of the same expression of the ground truth. Our experiments show that generating multiple views only has a slight boost in performance over using the $RAND_i$ masked BART (Postfix) model. Specifically, we notice 0.1% and 0.6% accuracy gains over the $RAND_i$ results in SVAMP and MAWPS, respectively. With these gains, the $RAND_i$ Multi-view BART (Postfix) model outperforms the SOTA seq2tree RoBERTa-GTS model by 0.5%.

Generation Consistency

From Figure 4.5 we see that the context representation of each node/token diverges over time as measured by the cosine similarity. The divergence suggests that the model learns different heuristics per view that are less likely to translate to the traversal of an expression tree. Effectively, this treats the multi-view generation task as a data augmentation technique.

We seek to minimize cosine distance between context representations of the same nodes to encourage the model to store more structural information per token it generates. By adding this objective, our $RAND_i$ masked Multi-view BART (Postfix-Gen-Con) model achieves 90.5% and 42.9%, on MAWPS and SVAMP, respectively. These represent 0.6% and 2.0% accuracy improvements on MAWPS and SVAMP, respectively.

Model	MAWPS	SVAMP
Transformers (Scratch)		
(2) Prefix-Transformer [Griffith and Kalita 2020]	84.7	-
(3) Infix-Transformer [Griffith and Kalita 2020]	62.4	-
(2) Postfix-Transformer [Griffith and Kalita 2020]	83.1	-
Transformer [Lan *et al.* 2022]	85.6	20.7
Transformers (pre-trained)		
Transformer (R) [Patel *et al.* 2021]	87.1	38.9
Roberta-Roberta [Lan *et al.* 2022]	86.9	24.8
BERT-BERT [Lan *et al.* 2022]	88.4	30.3
mBART [Shen *et al.* 2021]	80.1	-
Gen&Rank [Shen *et al.* 2021]	84.0	-
SOTA		
RE-Deduction [Jie *et al.* 2022]	92.0	45.0
Roberta-Graph2Tree [Patel *et al.* 2021]	88.7	43.8
Roberta-GTS [Patel *et al.* 2021]	88.5	41.0
MultiView [Zhang *et al.* 2022a]	92.3	-
$NONE$-mask		
BART (Postfix)	81.6	38.8
NUM-mask		
BART (Postfix)	86.6	30.5
NUM_i-mask		
BART (Postfix)	89.2	40.6
$RAND_i$-mask (Ours)		
BART (Posfix)	89.9	40.9
$RAND_i$-masked Multi-view (Ours)		
BART (Postfix)	90.0	41.5
BART (Postfix-Gen-Con)	90.5	42.9

Table 4.3: Overall results of the multi-view model compared to baseline models.

The results indicate that aligning the token representation of each node in the expression tree can help the model. This shows that this strategy is not limited to tree decoders where it has been previously used. Furthermore, for sequence decoders, this can be applied at a more granular level.

4.4.7 Expression Notation

We analyze the performance of each view/notation within the multi-view notation and compare these against having only a single. We report the results on Table 4.4. What we find is that the performance across all views improves within the multi-view decoding compared to training the views individually.

We find that concatenating the views as one continuous output stream also outperforms the single view, however, less so compared to the prefixed multi-view setup. During generation, we observe that the model can successfully generate the first view, however,

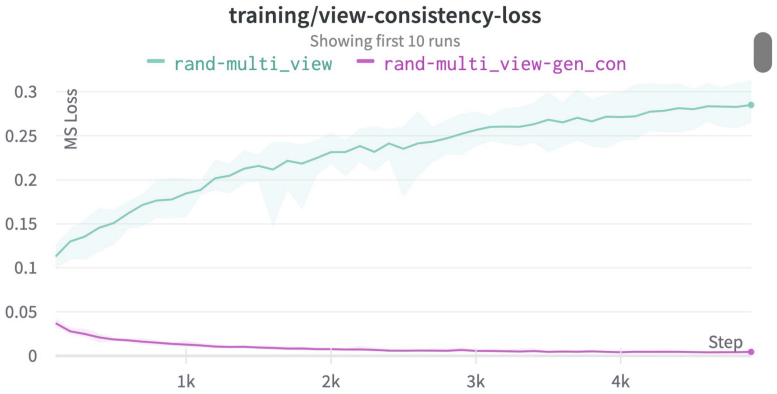


Figure 4.5: Divergence of context representations for the multi-view task during training.

	MAWPS				SVAMP			
Model	Prefix	Infix	Postfix	Oracle	Prefix	Infix	Postfix	Oracle
Single View								
NUM_i	40.0	38.6	40.6	-	88.0	89.1	89.2	-
$RAND_i$	40.7	41.0	40.9	-	89.6	90.0	89.9	-
Multiple Views								
Multi-view (concat)	4.5	7.1	41.9	42.9	5.6	16.8	90.2	90.8
Multi-view (prefixed)	42.6	42.1	42.9	48.3	89.9	90.6	90.5	91.4

Table 4.4: Performance of each view in the multi-view setup compared to when they are trained separately.

after the delimiter, the model tends to generate invalid expressions. This is probably caused by the model's preference for shorter expressions [Lample and Charton 2019; Zhang *et al.* 2022a]. These findings are consistent with previous works [Liu *et al.* 2019b; Xie and Sun 2019; Zhang *et al.* 2020c] that found that the performance of generation models decreases rapidly as the expression length increases. We proposed this as a simple imitation of a chain of thought, and we find that it does not work as well as anticipated.

We further note that the improvements brought about by multi-view are minimal on MAWPS. A probable cause for this may be that a simple heuristic is sufficient to solve the MAWPS, as shown in Patel *et al.* [2021]. However, the SVAMP requires deeper heuristics, which the multi-view provides.

We notice that each of the views tends to be able to answer different questions successfully. We take the oracle of the produced results and we find that the SVAMP performance increases from 42.6% to 48.3%, an accuracy difference of 5.7% over the $RAND_i$-masked BART (Postfix). Our oracle technique considers an MWP solved if one of the views generates a correct answer. This motivates the work we embark on in the next chapter, where we aim to train the BART model to select the best view rather than naively taking a single view.

Model	MAWPS	SVAMP
Postfix, Infix	90.3	42.6
Oracle	91.2	47.1
Postfix, Prefix	90.2	42.1
Oracle	91.1	45.9
Postfix, Infix, Prefix	90.5	42.9
Oracle	91.6	48.3

Table 4.5: Results showing the impact of using a different number of views and view combinations.

4.4.8 Multi-view Notation

We further analyze our proposal of multiple views with a different number of views. Table 4.5 shows our experimental results. We find that the three views are complementary and that having all three is better than one decoder. Moreover, three views are also better than two. We note that the infix view improves the postfix notation that we report better than the prefix notation. The heuristics to solve prefixes and postfix notation are likely quite similar such that learning both of them does not improve one or the other greatly.

4.5 Conclusion and Future Work

In this chapter, we analyzed the data representations used in literature to train and evaluate models for Math Word Problem (MWP) solving. In particular, we took a closer look at how the quantities are masked, and what mathematical expression notation the model is expected to generate in order to solve MWPs. In section 4.2 we surveyed the literature in MWP and found three quantity masks found in the literature, namely, $NONE$, NUM, and NUM_i masking, where i is the i^{th} quantity.

We hypothesized that these different quantity representations used in literature are inefficient and can mislead the model to learn heuristics tied to the quantity rather than the relations. To this end, we propose a new strategy to mask quantities, $RAND_i$, where we randomly assign the quantity masks to the quantities, treating i as a quantity indicator rather than an index. Our experimental results prove that our hypothesis holds across two widely used benchmark datasets regardless of the expression notation used.

Besides the quantity masking, we investigate expression representation, which determines the heuristics the model learns to represent. Inspired by multiple expression views [Shen and Jin 2020; Zhang *et al.* 2022a] and aligning similar sub-tree expressions [Li *et al.* 2021c; Zhang *et al.* 2022a], we adapt this for pre-trained Transformers, particularly BART. Different to multi-view decoding above, we use only a single sequential decoder. As such, our multi-view decoding strategy can be seen as a data augmentation technique. Since the model architecture is unchanged, this allows us to fairly compare this data representation to other single expression notations used in

literature, namely, prefix, postfix, and infix.

Our experimental results show that augmenting the data with multiple expression trees helps the model capture more structural information without needing to use a specialized decoder architecture [Jie *et al.* 2022; Xie and Sun 2019]. We find that the model can better generate the prefix, infix, and postfix expressions when it is trained under a multi-view setting rather than on just a single view. We also show that adding an alignment objective between the multiple views improves performance across all views. This reinforces our hypothesis that we can add several objectives that can help the Transformer learn structural information, rather than changing the structure entirely.

Analyzing the oracle of the generated views, we find that the model can better answer MWPs using certain views than others. The strategy used in literature to take the best-performing view fails to take advantage of this [Zhang *et al.* 2022a; Shen and Jin 2020; Meng and Rumshisky 2019]. In the next chapter, we explore several ways to choose between these views to generate the final expression.

While our results do not outperform SOTA [Patel *et al.* 2021; Jie *et al.* 2022; Zhang *et al.* 2022a], we have shown that data representation under low resources is critical for great performance on MWPs. By simply changing the quantity masking and expression notation, we gain 2.2% relative performance over the widely adopted prefix notation with NUM_i masking for the SVAMP dataset. Furthermore, by adding a view token-alignment objective across our multi-views, we gain a further relative improvement of 3.4%.

4.5.1 Future Direction

This chapter explored the impact of the representation of MWP data on the BART model. The following details limitations and future work that can arise from our results.

Data Representation Model Robustness

We only conduct experiments using the BART [Lewis *et al.* 2019] pre-trained language model. It is worth testing the data representation's robustness across multiple pre-trained models [Liu *et al.* 2020b; Raffel *et al.* 2020], and also training from scratch. It would also be interesting to see how the seq2seq results reported in the literature improve with the proposed data representation [Wang *et al.* 2017; Lan *et al.* 2022; Shen *et al.* 2021; Wang *et al.* 2018c]. In particular, Zhang *et al.* [2022a] uses only two of three views. It is likely that adding an infix view or the flattened version thereof [Chen *et al.* 2021b] can further improve the results reported.

Other Data Representation Views

We only considered three widely used data representations, namely, prefix, infix, and postfix notations. However, we find other math expression notations used in literature. However, other representations have emerged in the literature. The sub-tree representation [Jie *et al.* 2022; Cao *et al.* 2021] can help the model reuse expressions generated

sub-expressions, while the M-tree representation [Wang *et al.* 2022a] unifies the multiple ground truths that result from the mathematical properties. Chen *et al.* [2021b] represents the reasoning program in a flattened format and generates the right parentheses forcibly after generating two consecutive operands. It is worth seeing how these representations work using the Transformer model and the impact they might have under an extended multi-view setting.

Rather than taking the average, another strategy would be to weigh the postfix view higher, such that the infix and prefix views are treated as auxiliary tasks reinforcing the postfix view. We leave the study of this and the impact of this for future work.

Expression Normalization

Expression Contraction For our experiments, we used the ground truth as it was. However, it might be beneficial for the model to consolidate different expressions that arise due to the distributive, associative, and commutative properties of the mathematical expressions. Zhang *et al.* [2022a] found that expanding all the ground truth expressions worsens single-view performance, however, it boosts performance for multiple views. This shows that normalization needs to be accounted for effective data representation. Lample and Charton [2019] and Jie *et al.* [2022] found that the model favors shorter solutions, which suggests that finding contracted ground truth solutions might aid the model more than expansion. However, these expressions might also be more complex and harder to learn. Alternatively, using a unified tree structure as proposed by Wang *et al.* [2022a] might work.

Soft-labels Besides finding a unified representation, we could augment the data with multiple ground truth labels. Our experiment showed that augmenting multiple views is beneficial, it is likely that augmenting alternatives to solving the question could further help the model. We note that it is possible to generate all simplified ground truth label variations. For example, $n_1 - n_2$ only has this variation while $n_1 + n_2$ has another variation, $n_2 + n_1$. These variations can be used to create soft labels to train the model via the cross-entropy loss function.

The first example will still have hard labels, 1, and 0 elsewhere. However, the second example would have 0.5 for the n_1 and n_2 tokens at the first position, and 0 elsewhere. The second position is 1 and 0 elsewhere. The third position would be 1 for n_1 or n_2 depending on whether the first position was n_1 or n_2. Having these soft labels, we can train the model with both $n_1 + n_2$ and $n_2 + n_1$. This frees the training process expression property bias. We ran experiments under associative, and commutative properties and found this to boost performance, however, we have left the full investigation for future work.

Other Datasets

We ran our experiments on the SVAMP and MAWPS datasets, which are low-resource MWP datasets. It is worth testing the proposed data augmentation on MathQA [Amini *et al.* 2019] and Math23k [Wang *et al.* 2017].

We note that the MathQA dataset has questions spanning a wide variety of topics in mathematics and physics. This introduces several repeating quantities in the question which the model needs to learn to reuse. This is hard to do with NUM_i masking which treats each quantity as unique, or with NUM masking does not indicate which quantity is being referred. We suspect that a unique masking technique can help, where i is incremented for every unique quantity encountered. The i can further be randomized to avoid the model overfitting to this indicator.

Chapter 5

Multi-tasking Multi-view Decoders

5.1 Introduction

In Chapter 4 we discuss techniques for augmenting data that can be used to enhance the performance of pre-trained seq2seq Transformers. In particular, we fine-tuned the pre-trained BART [Lewis *et al.* 2019] model on the SVAMP and MAWPS benchmark datasets, and we showed that randomly masking the quantities appearing in MWPs generates better performance than quantity index masking.

Given that the datasets are small, the model repeatedly encounters the same data leading it to overfit to the correct answer faster than learning more generalizable heuristics of correct reasoning [Cobbe *et al.* 2021]. To force the model to learn from the limited data, research introduces auxiliary objectives that force the model to learn to reason in depth [Qin *et al.* 2021; Wu *et al.* 2020b 2021d].

While auxiliary tasks improve performance against the baseline, the generation task only expresses the solution tree from a single view. Our experiments in Chapter 4, alongside the works of Shen and Jin [2020], Zhang *et al.* [2022a], and Meng and Rumshisky [2019], show that having multiple views results in better performance than just having a single view.

Having multiple views presents the challenge of choosing the final view to use. Zhang *et al.* [2022a] has two decoders producing two views and simply choose the best-performing decoder during training. From the previous chapter, we found that this limits the capability of the model, given that each decoder might be better suited to solve certain questions than the chosen one, though it might be best on average. Shen and Jin [2020] performs a beam search and takes the view that has the highest score of all decoders. However, mathematical expressions are susceptible to minor errors, and the generation objective does not explicitly account for them [Shen *et al.* 2021].

Shen *et al.* [2021] proposed a multi-tasking framework for solving MWPs where they generate multiple candidate solutions to MWPs using beam search, then they apply a verifier to choose the most likely candidate. Cobbe *et al.* [2021] showed that the use of verifiers can help correct the mistakes the model makes, achieving results ap-

proximately the same performance boost as a 30x model size increase. In this chapter, we propose to extend the verifier to a multi-view setup, where rather than choosing candidates from one view, we choose candidates across multiple views.

We consider binary classification similar to Shen *et al.* [2021] and metric learning to train the verifier. The metric learning objective aims to learn an embedding space where the MWPs will be embedded closer to their correct solutions than to the incorrect solutions. This requires the model to maximize the mutual information between the MWP and the solution while discarding any non-predictive information in the MWP [Sun and Zhang 2021; Li *et al.* 2021b].

Shen *et al.* [2021] uses the end-of-string token to verify the candidates. However, the results from Reimers and Gurevych [2019] suggest that other pooling mechanisms such as mean pooling might be better suited for this task. To this end, we experiment with several pooling strategies to train the verifier.

Different to Chapter 4 where our interest was to investigate different data representations under the same model architecture, in this chapter we use multiple decoders to generate each view similar to related work [Zhang *et al.* 2022a; Shen and Jin 2020]. To keep our model parameters low, we apply a simple layer-wise pruning technique [Shleifer and Rush 2020].

In summary, the contributions of this chapter are as follows:

- We propose a multi-view multi-tasking framework to solve Math Word Problems. We provide comprehensive impacts of different pooling techniques and verification architectural and training objective designs.
- Experiments on the SVAMP and MAWPS benchmark datasets show the effectiveness of the proposed architecture.

The rest of the chapter is structured as follows: in Section 5.2 we discuss related work with multi-tasking and verification objectives. Section 5.3 follows with a discussion of our methodology. We present and discuss the results of our experiments in Section 5.4. Thereafter, we conclude the chapter and discuss plausible future work in Section 5.5.

5.2 Related Work

In this section, we discuss related work to our proposal. Our proposed model draws from several concepts, including multi-view learning, multi-tasking, and verification. Having already reviewed multi-view learning and generation consistency in Section 4.2, in the following subsections, we discuss the remaining concepts.

5.2.1 Multi-tasking Models

Multi-tasking has been used in NLP to help models learn complementary tasks concurrently [Collobert and Weston 2008; McCann *et al.* 2018; Li *et al.* 2021a; Zhang *et al.* 2022b]. Pikos *et al.* [2021] adds a reasoning order prediction auxiliary task to the

BERT pre-training masked language modeling objective in order to help it learn mathematical rules better. Huang *et al.* [2021] introduced a memory-augmented subtask that recalls problems with similar expression structures using an auxiliary relation and retrieval task.

Liang *et al.* [2021] adds several auxiliary tasks during BERT's pretraining, including counting quantities in the question, comparing the magnitudes of quantities, predicting whether the quantities and answer are integers, and predicting the operator between quantities and the distances they appear in the expression tree. Qin *et al.* [2021] enforces different kinds of symbolic reasoning with four extra tasks: counting the number of quantities, finding their locations, predicting commonsense constants, and a consistency checker.

Shen *et al.* [2022] and Li *et al.* [2021c] propose contrastive learning objectives which support the model to better understand patterns and to differentiate semantically similar examples that have different mathematical logic. Our work is closely related to Shen *et al.* [2021] who designed a multi-task learning framework, where they first generate the candidate expressions, followed by a verification task that retrieves the most likely candidate.

5.2.2 Multi-Decoder Models

Multiple decoders have the ability to enrich the encoder representation similar to multi-tasking [Hao *et al.* 2020]. Liu *et al.* [2020a] uses the Transformer encoder to produce embeddings that are used by several task-specific heads to perform tasks such as sentence classification; pairwise similarity, ranking, classification; and reading comprehension. Hao *et al.* [2020], Inaguma *et al.* [2021a] and Inaguma *et al.* [2021b] share the knowledge learnt by an auto-regressive decoder with a non-auto-regressive decoder by using a shared encoder. Wu *et al.* [2021a] shares a single encoder with five decoders which are used simultaneously for anomaly detection, event detection, and anomaly diagnosis.

The use of multiple decoders has also been adopted to solve MWPs. Zhang *et al.* [2020b] distills the knowledge of a Graph2Tree [Zhang *et al.* 2020c] model to two student decoders that are encouraged to generate diverse solutions. Meng and Rumshisky [2019] shares an encoder with two decoders, one that generates the math expression and the other that generates the expression in reverse order. Zhang *et al.* [2022a] and Shen and Jin [2020] generate two views, a prefixed view, and a postfix view. The prefix views are generated using a tree decoder, while the postfix view is generated using a sequential decoder in Shen and Jin [2020] and a relational decoder in Zhang *et al.* [2022a]. Different from these approaches, all our decoders are sequential. We also use the infix view in addition to the prefix and postfix views they consider.

5.2.3 Verifiers

Greedy search and beam search are among the widely used approaches to generate samples. However, these strategies are prone to minor mistakes for the MWP task,

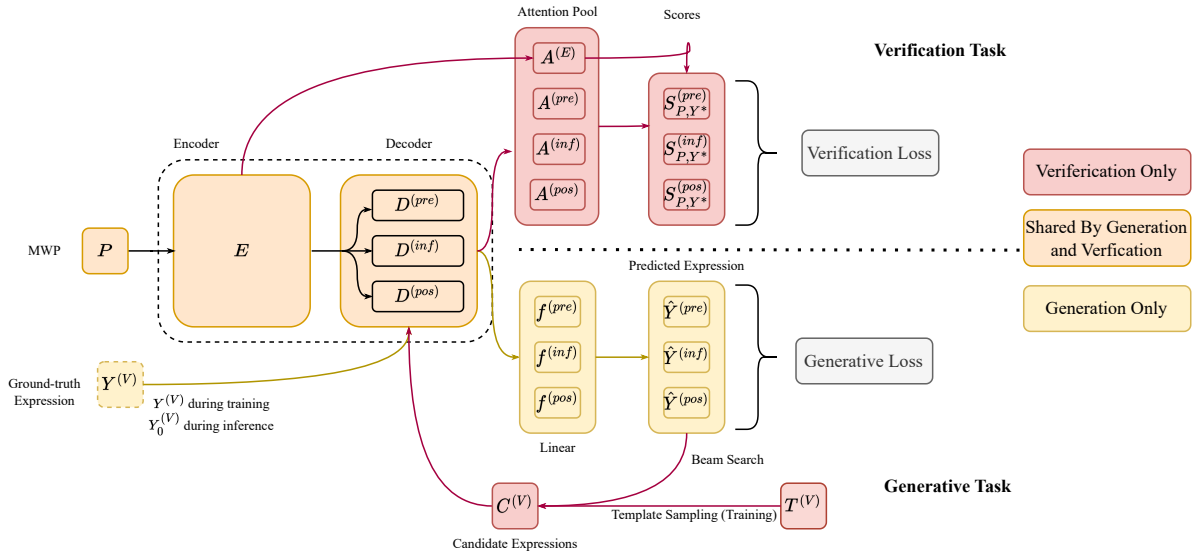


Figure 5.1: Our multi-view ranking (MVR) model. Three decoders generate view-specific expressions from each encoded MWP. This forms our multi-view generation objective. Candidates for the verification objective are generated by random template sampling and beam search. These are passed through the decoders to get the hidden states, which are pooled with the encoder hidden states to get a fixed representation as part of our verification objective. Only beam-search generates candidates during inference.

whereas common NLP tasks are unaffected by these mistakes [Shen *et al.* 2021]. Wang *et al.* [2022b] proposed self-consistency, which first samples a wide range of reasoning paths rather than just choosing the greediest one, and then chooses the most consistent response by marginalizing out the sampled paths. We use a similar technique as our baseline, where we take the most consistent answer across the generated views.

Shen *et al.* [2021] proposed a two-stage generation process, where they apply beam search to generate multiple candidates for a given word problem, then use a verifier to rank each solution candidate returning the highest scored expression as the final solution. This architecture was later adopted by Cobbe *et al.* [2021] who showed that verifiers are easier to train and they perform similarly to scaling the model more parameters by thirty times more. Similarly, we train verifiers to rank the expressions generated from the multiple views.

5.3 Methodology

The framework of our model is shown in figure 5.1. It consists of two main objectives: multiview generation and verification. There is a single encoder that is shared by all the decoders. The encoder and decoders follow the BART architecture. The verification task shares the encoder and decoder weights with the generation task. There is a pooling mechanism in place before passing the decoder outputs to the verification layer. The expression that the verifier scores the highest from all the generated expressions across

the decoders is returned as the final solution.

5.3.1 Multi-view Generation

The model encoder and decoders are the same as described in Section 4.3.2. Unlike in the previous chapter, we use multiple decoders to generate multiple views, where each decoder is trained to generate only one view. These decoders share the encoder component similar to Zhang *et al.* [2020b 2022a] which forces the encoder to generate a unified representation of the MWP. Having multiple decoders over one allows each to specialize, which can increase its generation power. To improve the consistency among the generated views, we align the tokens generated in all the views using the same method described in Section 4.3.5.

Shared Encoder

As described in Section 4.3.2, the encoder takes in a math problem as a sequence of tokens P and produces the embedded representation of the inputs E. Given that we now have multiple decoders that will use the representations E, we have several options we can take: (i) have a separate encoder for each decoder, (ii) share the encoder and add a view-prompt to the encoder, and (iii) share the encoder without a prompt.

In the first case where the encoder is not shared, it will in essence be training three separate models and then taking an oracle of the results. This increases the number of parameters drastically compared to the shared alternatives. When sharing the encoder with a view prompt, the encoder states can be different for all prompts. While this approach has fewer parameters than the first, it has similar computational costs, since each math problem has to be encoded several times.

The third approach reduces the parameters and computational costs. This approach forces the encoder to generate generic embedding states. Motivated by this, we use a shared encoder for all of our experiments, unless explicitly stated otherwise. Besides computational costs, it might be beneficial to have fewer parameters to prevent the model from overfitting given that we have small datasets.

Multi-view Decoders

Similar to encoders, decoders can be view-specific or shared. In Chapter 4 we designed a single-shared decoder that generated multiple views using view prompts. Our interest was to investigate different data representations while keeping all other conditions constant for a fair comparison. While the single decoder performed better when it was trained on multiple views compared to one, it had to learn various decoding logic, which can be shared between specialized view-specific decoders. In this chapter, we use view-specific decoders to generate views.

Shared Embeddings

Press and Wolf [2016] found that sharing the embeddings between the encoder and the decoder elicits better performance than learning them independently. This has been adopted in the design of the Transformer [Vaswani *et al.* 2017] and BART [Lewis *et al.* 2019]. In our design of the multi-decoder Transformer, we have extended this idea by ensuring that all the decoders share their embeddings with the encoder. In other words, the input embeddings of all decoders are linked to the encoder's input embedding and subsequently tied to their respective output embeddings [Press and Wolf 2016; Inan *et al.* 2016].

Training

During training, we pass math word problems through the encoder to obtain their respective representations. We pass these representations through all the decoders to get multiple views of the solutions. The decoders are simultaneously trained on the same ground truth expressions, however, different notations.

Given a problem P, and a set of views $V = \{\textit{prefix, infix, postfix}\}$, our multi-view generation task is to train the corresponding decoders $D^{(v)}$ so that each produces the correct mathematical expression $\hat{Y}^{(v)}$ that answers P, where $v \in V$. We aim to minimize the weighted sum of the cross-entropy loss between the decoder's generation probabilities and the ground truth. This can be denoted mathematically as follows:

$$
\begin{aligned}
d_i^{(v)} &= F_{D^{(v)}}(E; y_{0:i-1}^{(v)}) \\
\mathcal{P}(\hat{y}_i^{(v)}|P, \hat{y}_{0:i-1}^{(v)}) &= \text{softmax}(f(d_i^{(v)})) \\
\mathcal{P}(\hat{Y}^{(v)}|P) &= \prod_{i=1}^{M+1} \mathcal{P}(\hat{y}_i^{(v)}|P, \hat{y}_{0:i-1}^{(v)}) \\
\mathcal{L}_{GEN}^{(D^{(v)})} &= \sum_{p} -\log \mathcal{P}(\hat{Y}^{(v)}|P) \\
\mathcal{L}_{GEN} &= \sum_{v \in V} \alpha_v \mathcal{L}_{GEN}^{(D^{(v)})}
\end{aligned}
\tag{5.1}
$$

where α_v is the associated weight for each view; $\hat{y}_{0:i-1}^{(v)} = (y_0^{(v)}, \hat{y}_1^{(v)}, \hat{y}_2^{(v)}, ..., \hat{y}_{i-1}^{(v)})$; $D^{(v)} = (d_0^{(v)}, d_1^{(v)}, d_2^{(v)}, ..., d_M^{(v)}, d_{M+1}^{(v)})$, where $d_i \in \mathbb{R}$; M is the number of tokens in Y; M is the number of tokens in Y; E is the output of the encoder given by Equation (4.1); index 0 and $M+1$ are special tokens ($[bos]$ and $[eos]$) indicating the start-of-decoding and end-of-decoding, respectively; f is a linear layer that maps the dimension of each decoder state $d_i^{(v)}$ to the dimension of the output vocabulary. We use $\alpha_v = 1/|V|$ for all our experiments, which corresponds to taking the mean loss objective of the individual views. During inference, we have three solutions generated from the decoders. To choose the final solution, we could further fine-tune a single view [Raffel *et al.* 2020] or use verifiers [Shen *et al.* 2021]. All of our experiments use the latter unless otherwise noted.

Model Compression

Introducing multiple decoders increases the number of parameters of the model quite substantially. This increases the computational and memory requirements for training and inferring from the model. We apply the "shrink-and-fine-tune" (SFT) strategy proposed in Shleifer and Rush [2020] to reduce the BART parameters. SFT extracts a student model from the maximally spaced layers of a fine-tuned teacher. We use the fine-tuned multi-view BART model from Chapter 4 as our teacher model. We only compress the decoder layers, taking the first and last layers of the BART model.

5.3.2 Verification

Our generation objective produces three solutions, one for each view. Our experiments in Chapter 4 warranted investigations on how to select the best solution from multiple views. This chapter proposes training verifiers to rank the solutions from our multi-decoder setup and return the highest scoring one as the best solution. To train the verifier network, we apply an attentive pooling mechanism to the decoder states in order to get the fixed context states that the verifier network expects. We train our model using the supervised contrastive learning loss.

View-consistency Votes (Baseline)

In Chapter 4 we simply chose the best-performing view during training and used that during inference. Our ablation studies revealed that this strategy is representative of the solution generation power of the multi-view model. Two approaches to choosing between multiple candidates exist in the literature: self-consistency [Wang *et al.* 2022c] and verification [Shen *et al.* 2021].

Self-consistency first prompts the language model with example chains of thought, which generates a diverse set of reasoning paths, returning the most common answer. We adopt a similar strategy to design a view-consistency approach. Our view-consistency approach uses beam-search to generate multiple candidate solutions across all our view decoders. Then we take the most consistent answer across all of our views. The consistent answer is the most frequent solution to the MWP across all candidate solutions.

Cobbe *et al.* [2021] found that increasing the number of beams and diversity increases the probability of incorrect solutions. Instead of simply taking the most consistent answer, we also consider the position they occur. We rank the generated expressions across all views, returning the highest-ranked consistent answer as the final solution. For each answer a in an ordered collection of answers $A^{(v)}$ generated by beam search for each view v, we sum the reciprocal indices of a across all views to give it a score. This is given by:

$$S_a = \sum_{v \in V} \sum_{A^{(v)}=a} \frac{\alpha_S^{(v)}}{i_a^{(v)}}$$

where S_a is the answer score; $i_a^{(v)} \in 1, 2, ..., |A^{(v)}|$ denotes the indices that a appear per

view, where $|A^{(v)}|$ is the cardinality of $A^{(v)}$; $\alpha_S^{(v)}$ is the weight of the vote score for each view v. The view weight helps to account for the fact that the views have different generation potentials. The weight can be set to $\alpha_S^{(v)} = 1$ for equal weighting or to the weighted accuracy of the validation set $\alpha_S^{(v)} = acc(v)/\Sigma_{(v \in V)} acc(v)$. Furthermore, the weight is used to break score ties, where we up-weigh the stronger views. Another strategy to break the ties would be to increase or decrease the number of candidates until the tie breaks, which is more tedious.

Verifier Ranking

Wang *et al.* [2022c] observed that self-consistency is less effective for short expressions due to the limited diversity within such expressions. To this note, we propose the weighted view consistency as our baseline. The other alternative to view consistency is verification which has been shown to work for shorter expressions [Shen *et al.* 2021]. In this approach, we train a single MLP layer that takes fixed decoder representation states and outputs a score that represents the likelihood of it being the correct answer. Having a verifier allows us to consider multiple candidates for each decoder, helping the decoder recover from its errors [Shen *et al.* 2021].

The rest of this subsection discusses the verifier setup. In particular, we discuss generating expressions for training, modifications to the decoder attention mask to account for verification as opposed to expression generation, pooling a context vector from the generated expressions to the verifier layer, and the training objective for the verifier.

Expression Generation

The verifier needs to see positive and negative examples in order to learn to distinguish a correct expression from an incorrect expression. To do this, we can generate several expressions using the model or we can create variations of the ground truth expression by altering several tokens [Shen *et al.* 2021].

Model Generation The first strategy is to produce candidate solutions from the BART model, similar to Shen *et al.* [2021]. Specifically, we produce top-K expressions using beam search. We evaluate each of these expressions and label them as positive if they correspond to the ground truth and negative otherwise. This method generates expressions that the model considers to be the most accurate. The verification task evaluates the model's representation of these expressions, enabling the model to rectify its errors.

Random Generation The model is only capable of generating solutions that have limited variations. As the model trains for several epochs, it encounters the same examples, as such, becomes more confident in its predictions [Cobbe *et al.* 2021]. To increase the diversity of the training set, Shen *et al.* [2021] applies several expression tree disturbances such as expanding subtrees, deleting, swapping, and replacing nodes. We find this strategy to be limited, as it is as good as the variations we apply.

We propose a simpler and more general approach to generate diverse training expressions. To generate templates, we consider all the possible valid prefix notations that

can be generated for a set of Q quantities. Figure 5.2 shows an example of these. Two quantities only have one template of length three oqq, and three quantities have two templates of length five, $ooqqq$ and $oqoqq$, where o and q denote the operators and quantities/operands, respectively.

During training, we first sample K random templates to use for each example in the batch. For a template of length L, we randomly sample $(L+1)/2$ quantities from the word problem and $(L-1)/2$ operators from the operator set $O = \{+, -, *, /\}$. We randomly substitute the quantities and operators into each sampled template. All our sampling is with replacement. We label the randomly generated expressions true or false based on whether they evaluate to the same answer as the ground truth.

We note that the different templates have a different number of variations. For example, the template with $L = 3$ only has one variation, while templates with $L = 9$ have 14 variations. For this reason, we propose a weighted template sampling strategy. The probability p_t of choosing a template t can be denoted by the following equation:

$$p_t = \frac{\gamma_t J_t^{-1}}{\sum_{i=1}^{|T|} \gamma_i J_i^{-1}} \tag{5.2}$$

where T is a set of expression templates, J_t is the number of variations for expression t with length L_t, and γ_t is the weight associated with template t. Setting $\gamma_t = 1$ makes sampling templates of different lengths equally probable.

We note that if the ground solution has $L = 3$, a template with $L = 9$ might be too trivial for the verifier to learn from. Consequently, this might slow down the training since these easy examples do not yield substantial gradient updates that enhance learning, wasting valuable training time. We want to sample potentially hard examples for the model to learn [Hermans *et al.* 2017]. For this reason, for each ground truth Y, we propose to set γ_t using the following equation:

$$\gamma_t = \frac{1}{(|L_t - L_Y| + 1)} \tag{5.3}$$

where L_t represents the length of the template t out of the set of possible lengths $L = \{3, 5, 7, ..., M\}$; L_Y represents the length as the ground truth expression Y; and $|\cdot|$ denote the absolute value. Unlike Shen *et al.* [2021], we update our expressions at every batch, rather than in between epochs.

Decoder Masking

During training, the decoder applies a causal mask, which prevents it from attending to future tokens. This simulates the generation during inference where future tokens will not be available. However, for the verification task, all tokens are available to the model, similar to inputs to the encoder. As a result, for the verification task, we can use a causal mask to emulate the generation objective or verify without a mask, similar to the encoder.

We note that it might be useful for the verifier to be able to attend to future tokens. This could allow it to verify the expressions at a token level rather than at a global

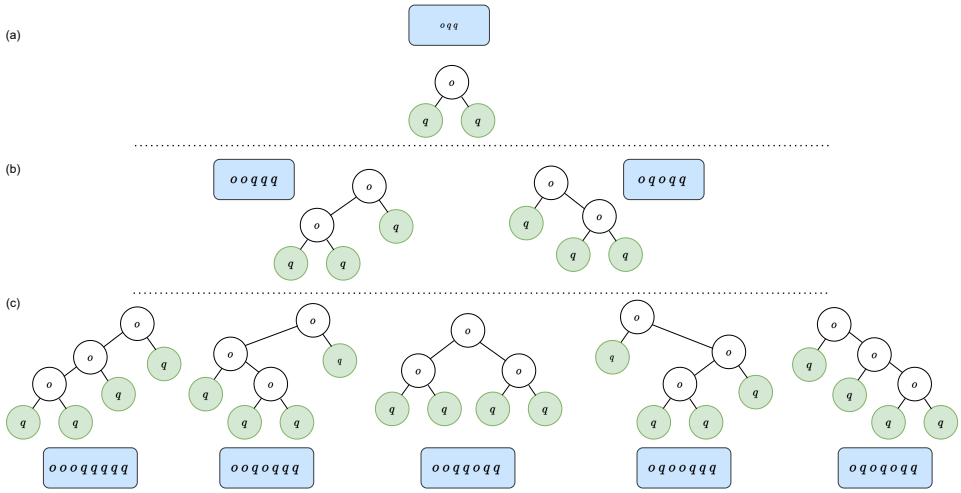


Figure 5.2: Examples of the possible expression trees with 2 to 4 quantities.

level [Cobbe *et al.* 2021]. Since the generation task does not pay attention to future tokens, the model can learn to separate the generation task from the verification task by associating these with different attention.

However, we note that the decoder was not pre-trained to pay attention to future tokens and this can cause discrepancies during training [Reimers and Gurevych 2019]. Using a causal mask allows the verifier to have the same reasoning as the decoder had during the generation. To this end, we empirically choose to verify with an attention mask during training and inference.

Pooling

The generated expressions have several states that vary in length. We need a way to represent the solution as a single state. Several pooling strategies have been proposed, including max pooling, mean pooling, first/last token pooling, and weighted sum pooling. Shen *et al.* [2021] uses last token pooling, however, this has been shown to be inefficient [Reimers and Gurevych 2019]. We empirically analyze multiple pooling techniques (as discussed in Section 3.4) and choose the weighted sum pooling strategy.

Single Token Pool The single token pool simply takes a single state that is expected to be representative of the entire sequence. Our single token pool experiment uses the last generated state (eos-token) following previous work [Shen *et al.* 2021; Cobbe *et al.* 2021; Neelakantan *et al.* 2022; Muennighoff 2022]. We note that we are not restricted to eos-token similar to these proposed works since we are not using a causal mask.

Mean Pool The results from Reimers and Gurevych [2019], Choi *et al.* [2021] and Mosbach *et al.* [2020] found that the mean pooling strategy performs better than the first token pooling mechanism. To this end, we consider the mean pooling strategy. Given that the BART model [Lewis *et al.* 2019] was not pre-trained with a single token pooling objective, a mean pool might be advantageous. Mathematically, the mean pool

gives us:

$$c = \frac{1}{|H|} \sum_{h \in H} h$$

Weighted Pool The mean pool treats all states as equivalent to establish the context that will be used to determine whether an expression is valid. However, incorrect states should contribute differently to the creation of the context vector than correctly predicted states. To accomplish this, we allow the verifier to learn which states to prioritize using the attention mechanism introduced by Bahdanau *et al.* [2014]. The context vector c is calculated as a weighted representation of the generated states:

$$c = \sum_{i=1}^{N} a_i h_i \tag{5.4}$$

where

$$a_i = \frac{\exp\left(\text{score}\left(q, h_i\right)\right)}{\sum_j^N \exp\left(\text{score}(q, h_j)\right)}$$

and

$$\text{score}(q, h_i) = v_a^\top \tanh\left(W_a[q, h_i]\right)$$

where v_a and W_a are trainable parameters; q represents the query state. The attention pooling strategy shows which state the verifier considers and to what degree when verifying the expressions. This can help in understanding and diagnosing the verifier.

Verification Objective

Given the context vectors for each candidate expression, the objective of the verifier is to generate a score that can be used to rank several candidate expressions. We consider two approaches to score the candidate expressions: classification-view verification and metric-view verification.

Classification-view Verification Our classification-view verification task adopts the same scoring architecture that was used by Shen *et al.* [2021]. In this setup, the pooling mechanisms' context vectors $c^{(v)}$ are sent through an MLP layer that is connected to each decoder. We train our verifier using the binary cross entropy loss (BCE). Formally, this is defined as:

$$\mathcal{L}_{BCE}^{(v)} = -\frac{1}{|\mathcal{C}^+ \cup \mathcal{C}^-|} \left[\sum_{(P,Y^*) \in \mathcal{C}^+} \log \mathcal{P}(1|P, Y^*) + \sum_{(P,Y^*) \in \mathcal{C}^-} \log \mathcal{P}(0|P, Y^*) \right] \tag{5.5}$$

where $\mathcal{C}^+$ and $\mathcal{C}^-$ respectively denote the sets of positive and negative examples generated from the beam search and from the proposed template strategy; Y^* represents the set of candidate expressions in $\mathcal{C}^+ \cup \mathcal{C}^-$; and the notation $(P, Y^*) \in \mathcal{C}^+$ represents all the candidate expressions that evaluate to the correct solution for the MWP P.

Metric-view Verification The metric-view verification aims to learn a metric space where the MWP is closer to the correct candidate solutions than to the incorrect ones. Our training setup uses an MWP context vector u and the candidate context vectors c. The MWP context vector is generated by applying a pooling mechanism in the hidden states of the encoder followed by a projection layer [Chen *et al.* 2020b]. The encoder context vector u acts as an anchor, while the decoder context vector c represents positive and negative examples. We train the model using the Multi-Similarity loss defined as follows:

$$\mathcal{L}_{MS} = \frac{1}{|\mathcal{C}^+ \cup \mathcal{C}^-|}\left[\frac{1}{\beta_1}\log[1 + \sum_{(P,Y^*)\in\mathcal{C}^+}\exp(-\beta_1(S_{P,Y^*} - m))]\right. \\ \left. + \frac{1}{\beta_2}\log[1 + \sum_{(P,Y^*)\in\mathcal{C}^-}\exp(\beta_2(S_{P,Y^*} - m))]\right] \quad (5.6)$$

where $\mathcal{C}^+$ and $\mathcal{C}^-$ are the positive and negative sets, respectively; m is a margin, β_1 and β_2 are the hyper-parameters that represent the positive examples weight, and negative examples weight respectively; S_{P,Y^*} is the scoring function, where we use cosine similarity:

$$S_{P,Y^*} = \frac{u \cdot c^*}{||u|| \cdot ||c^*||}.$$

Multi-view Verification The multi-view verification objective can be denoted as:

$$\mathcal{L}_R = \sum_{v\in V}\alpha_R^{(v)}\mathcal{L}_R^{(v)} \quad (5.7)$$

where $\alpha_R^{(v)}$ represents the weight associated with each view, and $\mathcal{L}_R^{(v)}$ is given by equation (5.5) or equation (5.6). We weigh all our views equally $\alpha_R^{(v)} = 1/|V|$.

Expression Mining We adopt the multi-similarity (MS) mining strategy [Wang *et al.* 2019b] as a tool to get hard examples to train our model and also to correct class imbalances that arise from our dataset [Dong *et al.* 2017]. We also considered focal loss [Lin *et al.* 2017a]. However, we note that our class imbalance is not severe enough for us to benefit from this.

The MS mining strategy chooses negatives that are closer to an anchor than its hardest positive and chooses the positives that are further from an anchor than its hardest negative. More formally, given the anchor P and a set of candidate expressions Y^*, the positive examples are chosen if

$$S^+_{P,Y^*} < \max_{(P,Y^-)\in\mathcal{C}^-} S_{P,Y^-} + \epsilon$$

where ϵ is a margin. Negatives are chosen if

$$S^-_{P,Y^*} > \min_{(P,Y^+)\in\mathcal{C}^+} S_{P,Y^+} - \epsilon$$

The MS mining strategy can downweigh the loss contributions of the dominant and simpler classes that the model masters early on. This allows the model to continue learning with reduced risks of overfitting.

Training Objective

Our final training objective to train our multi-view verification model accounts for the generation loss, the consistency loss proposed in Chapter 4, and the weighted verification/ranking loss defined by equation (5.7). Our training objective can be denoted by the following equation:

$$\mathcal{L} = \mathcal{L}_{GEN} + \alpha_{CON}\mathcal{L}_{CON} + \mathcal{L}_{R} \tag{5.8}$$

where α_{CON} is a hyper-parameter that weighs the contribution of the consistency objective. Different from Shen *et al.* [2021] and Cobbe *et al.* [2021], we train our model with the verifier from scratch. This makes the verifier also act as an auxiliary task that can benefit the model's generation capability.

5.4 Experiments

This section discusses the results of the experiments we ran to validate the effectiveness of our multiview ranking (MVR) model. We conduct a series of ablation studies that show the importance of the components of the MVR model. We also demonstrate a case study to show the effects of using our proposed methods.

5.4.1 Datasets

We use the SVAMP [Patel *et al.* 2021] and MAWPS [Koncel-Kedziorski *et al.* 2016] datasets described in section 4.4. All our experiments use the $RAND_i$ strategy that was proposed in Chapter 4. We ran all of our ablation studies on the SVAMP dataset, while the MAWPS dataset with a five-fold validation is used for the final model. We focus on SVAMP since it showed that there is a potential for an increase in the absolute accuracy of 5.7% from the methods that this chapter aims to investigate.

5.4.2 Baseline

We compare our results with state-of-the-art results found in the literature. These can be grouped based on the encoder-decoder architecture they use. Particularly, we have 4 major categories: sequence-to-sequence, pre-trained language models, seq2tree, and graph2tree. Similar to Chapter 4 and previous work, we report the results of these baselines from publications. The following describes our baseline models, with pre-trained language models already discussed in the previous chapter.

Sequence-to-sequence

Sequence-to-sequence (seq2seq) models treat MWP as a translation problem from one sequence to another. DNS [Wang *et al.* 2017] uses a vanilla seq2seq model to translate MWP to equation templates directly. Math-EN [Wang *et al.* 2018c] utilizes an equation normalization method to address the problem of duplicated equations. S-Aligned [Chiang and Chen 2018] adopts a stack to track the semantic meanings of numbers. SAU-Solver [Qin *et al.* 2020] replace the leaf nodes with codes that store information

about the path from the tree to themselves, then train a sequence-to-code model to generate the codes. Transformer [Lan *et al.* 2022] uses the vanilla Transformer [Vaswani *et al.* 2017] to solve MWPs.

Seq2tree

GTS [Xie and Sun 2019] generates expression trees by a tree-structured neural network in a goal-driven manner. GSGSF-6L [He and Xiao 2022] proposes an ELECTRA [Clark *et al.* 2020] based encoder and a Goal Selection and Feedback decoding module that improves on GTS. Roberta-GTS [Patel *et al.* 2021] replaces the GTS encoder with pretrained Roberta [Liu *et al.* 2019c] embeddings. Group-ATT [Li *et al.* 2019] extracts multiple features in MWP by a group multi-attention mechanism such as a question or quantity-related attentions in the seq2seq model. TSN [Zhang *et al.* 2020b] generates diverse candidate expressions with a multiple-decoder network. T-RNN [Wang *et al.* 2019a] applies a seq2seq model to predict a tree-structure template, which includes inferred numbers and unknown operators. Then, An RNN is used to obtain unknown operator nodes in a bottom-up manner.

RE-Deduction [Jie *et al.* 2022] treats MWPs as a complex relation extraction problem, where they interactively construct target expressions, where each step involves a primitive operation over two quantities defining their relation.

Graph2Tree

Graph2Tree models extend the seq2tree replacing the sequence decoder with a graph encoder that explicitly learns the structural information contained in MWPs. Graph2Tree [Zhang *et al.* 2020c] is an early graph2tree model that leverages an external graph-based encoder to enrich the quantity representations in the problem. Roberta-Graph2Tree [Patel *et al.* 2021] initializes the graph encoder with a pretrained Roberta [Liu *et al.* 2019c] embeddings. EEH-G2T [Wu *et al.* 2021b] constructs edge-labelled graphs for problem representation and captures common sense information based on external knowledge bases.

CLRSolver [Wu *et al.* 2022] is a clause-level relationship-aware math solver that learns dependency relationships from the clause semantics in a global view. CogSolver [Liu *et al.* 2022] uses a knowledge-aware module and a commutative module to improve its reasoning ability, where the knowledge is organically integrated into the answer reasoning process.

TeacherSup [Liang and Zhang 2021] uses a teacher module that is designed to make the MWP encoding vector similar to the correct expression and dissimilar from the incorrect expression, which are manipulated from the correct solutions. RPKHS [Yu *et al.* 2021] is reasoning with pre-trained knowledge and hierarchical structure network, which contains a pre-trained knowledge encoder and a hierarchical reasoning encoder. MultiE&D [Shen and Jin 2020] is an oracle of sequence-based encoder-decoder with graph-based encoder-decoder to obtain better semantics and reasoning.

5.4.3 Experimental Setup

We use the same evaluation metrics and training setup discussed in Section 4.4. We set the following hyper-parameters: m, β_1, and β_2 in Equation (5.6) are set to $1, 2, 50$ respectively, similar to Wang *et al.* [2019b].

Model

We conduct all our experiments with a layer-pruned BART model, with two decoder layers. We take the first and last layers of our fine-tuned BART model from Chapter 4. From our preliminary experiments, we found that this performs similarly to the full model when fine-tuned for the same task. Similar results were reported by Shleifer and Rush [2020].

Ablation Setup

We consider a single view for ablation studies, which simplifies our model architecture. That is, we study different architectural design choices from a single view and then apply the final result across multiple views. We use the postfix view as our base. Unless stated otherwise, the model is trained from a single view.

5.4.4 Overall Results

We present the results of our Multi-view Ranking (MVR) BART model in table 5.1. We show the results of using different strategies to select the final solution: taking the postfix view, using the voting mechanism with greedy search (Voting 1), and beam search (Voting 4), and using the verifier with greedy search (Verifier 1), and beam search (Verifier 4). We compare our MVR model with the Multi-view (MV) model from the previous chapter using postfix notation, and the voting mechanism we proposed: Voting 1 and Voting 4. The comparison extends to several baseline models found in the literature. Our results show that adding the voting mechanism and the verifier improves the overall performance of the multi-view model, resulting in SOTA accuracy in both MAWPS and SVAMP datasets.

MV Voting

Our results using the multi-view model (MV) from Chapter 4 revealed that the model can produce an untapped potential if we were to consider all views rather than choosing one. This chapter proposes a view-consistency voting mechanism to select the solution most agreeable by multi-views.

Voting-1 We notice that adding a simple voting mechanism to the results we found in Chapter 4 results in an improvement in absolute precision of 1.2%, outperforming Graph2Tree at 44.1%. The 1.2% represent examples where the postfix view was incorrect, however, the prefix and infix views were. The Voting-1 mechanism captures 91.3% of the oracle result (48.3%) from the three views.

Model	MAWPS	SVAMP
Seq2seq		
DNS [Wang *et al.* 2017]	59.6*	24.2*
T-RNN [Wang *et al.* 2019a]	66.8	26.1*
Math-EN [Wang *et al.* 2018c]	69.2	25.0
S-Aligned [Chiang and Chen 2018]	-	26.1*
Group-ATT [Li *et al.* 2019]	76.1	21.5*
Transformer [Lan *et al.* 2022]	85.6	20.7
Seq2tree		
GTS [Xie and Sun 2019]	82.6	29.1*
SAU-Solver [Qin *et al.* 2020]	-	29.7
GSGSF-6L [He and Xiao 2022]	84.5	-
Roberta-GTS [Patel *et al.* 2021]	88.5	41.0
Graph2tree		
Graph2Tree [Zhang *et al.* 2020c]	83.7	36.5
TSN [Zhang *et al.* 2020b]	84.4	29.0*
EEH-G2T [Wu *et al.* 2021b]	84.8	-
CLRSolver [Wu *et al.* 2022]	82.2	-
CogSolver [Liu *et al.* 2022]	82.9	-
RPKHS [Yu *et al.* 2021]	89.8	-
Roberta-Graph2Tree [Patel *et al.* 2021]	88.7	43.8
Seq2DAG		
RE-Deduction [Jie *et al.* 2022]	92.0	45.0
Multi-view/multi-decoder		
MultiE&D [Shen and Jin 2020]	-	32.4*
TeacherSup [Liang and Zhang 2021]	84.2	-
MultiView [Zhang *et al.* 2022a]	92.3	-
PLMs		
Transformer (R) [Patel *et al.* 2021]	87.1	38.9
Roberta-Roberta [Lan *et al.* 2022]	86.9	24.8
BERT-BERT [Lan *et al.* 2022]	88.4	30.3
mBART [Shen *et al.* 2021]	80.1	-
Gen&Rank [Shen *et al.* 2021]	84.0	-
(Ours)		
MV (Postfix)	90.5	42.9
MV (Voting-1)	90.7	44.1
MV (Voting-4)	91.0	46.3
MVR (Postfix)	90.7	46.6
MVR (Voting-4)	92.6	49.6
MVR (Verifier)	**93.0**	**50.3**

Table 5.1: Overall results of our Multi-view Ranker (MVR) compared to our baseline and models proposed in previous work. The '*' represents results that are not reported by the original authors but reproduced in other publications.

Voting-4 We observe that we can further improve this result by 2.1% absolute accuracy when we increase the number of beams per view to four. The Voting-1 strategy is quite greedy, only allowing one vote per view. Increasing the beam size increases the number of candidates and the potential of finding the most consistent solution.

The Voting-4 algorithm results in an accuracy of 46.3%, which is relatively close to (95.6% of) the greedy search oracle that we set out to achieve. Furthermore, this improves the SOTA results by an absolute accuracy of 1.3%. However, the oracle accuracy of the views with four beams is 68.4%. The Voting-4 algorithm can only capture 67.7% of all correct answers. This shows that there is a larger potential that we can achieve with better algorithms.

Multi-view Ranking (MVR) Postfix

The MVR (Postfix) model treats the verifier as an auxiliary task. This model achieves an accuracy of 46.6%, an accuracy improvement of 1.7% from our MV (Postfix) model on the SVAMP dataset. This result indicates that verification is a strong auxiliary task, outperforming our view-consistency voting strategy. The discriminative nature of the verification improves the generative task of the model [Goodfellow *et al.* 2020]. Given that we use a single MLP layer for verification, most of the reasoning process has to be handled by the decoder, which consequently improves the generative process.

MVR Voting and Verification

As we saw from the previous chapter, simply taking the results of a single view in a multi-view setup does not fully represent the model capability. Using the Voting-4 algorithm, we report an accuracy of 49.6%, and an accuracy of 50.3% when using the Verifier, representing a pass mark in most states. Furthermore, we get SOTA results when using any of the two strategies above, pointing to the effectiveness of considering all views over one. Other than being a strong auxiliary task, our results show that the verifier is a more effective strategy than the voting algorithm.

We find that the oracle accuracy of the four most generated beams in our MVR is 72.8%. This result suggests that difficult datasets, such as the SVAMP, can be solved by joint verification and generation collaboration. Although it appears that our verifier is performing well, its accuracy is only 69.1% of the oracle candidate solutions. This suggests that more effective verifiers may be required to capture this possibility.

5.4.5 Ablation

We present the importance of each component of our model in Table 5.2. We report the results using a single-view ranker (SVR) model, where the view is the postfix notation. The results are shown using greedy search (SVR), a beam search with 4 beams selected using the verifier (SVR-R), and the oracle (SVR-E) showing the accuracy for a perfect verifier.

We break down the SVR model by replacing each component at a time and report the results. We take this approach in order to limit the number of possible hyper-

parameter combinations. The components we investigate are training objective, causal attention, pooling mechanism, verifier training size, and batch-hard mining. Our verifier is trained with the MS objective. It uses a full-decoder-attention mechanism and takes in a weighted-sum pooling mechanism from the decoder's hidden states. We pass 16 candidate solutions per MWP through the MS miner before presenting them to the verifier.

Training Objective

We proposed to train our verifier using binary cross-entropy (BCE) or multi-similarity (MS) loss functions. We find that the MS objective marginally outperforms the BCE objective. By consolidating the representation of the encoder and correct candidates, the MS objective directly encourages the encoder to generate embeddings that predict the correct solution. This result also suggests that the verification task can be better learned as a metric learning task than as a classification task.

Decoder Attention

Our experimental results show that using full attention in the decoder helps the model generate more correct expressions, and the verifier to produce better scores than using causal attention. The full attention allows the verifier and attention mechanism not to think left to right, which allows it to generate more complex logic to verify the generated expressions. The verifier might be learning closely tied heuristics to the generation task, which inhibits its discriminative potential, resulting in an absolute accuracy difference of 0.5% between SVR and SVR-R.

Since we do not give a prompt to know whether it is generating or verifying, we hypothesize that the model might be using its access to future tokens to separate these tasks. Having a way to separate tasks is important, as we saw when we replaced our single-decoder multi-view architecture with a multi-decoder multi-view architecture. The result further suggests that the verifier can utilize the forward attention even though our model was not pre-trained on it.

Pooling

Replacing the attention pooling mechanism with either a mean pool or last-token (eos) pool results in an absolute accuracy drop of 0.7% and 1.2% respectively. The results indicate that the results of Shen *et al.* [2021] could potentially be improved by changing the pooling mechanism, proving our hypothesis. The mean pooling mechanism performs poorly compared to the last-token pooling mechanism.

Given that encoder and decoder expressions differ substantially in length, the encoder is forced to encode information such that irrelevant states are ignored when performing a mean pool. Jie *et al.* [2022] only pools the masked quantities of their Roberta encoder, which would force the model to encode all the information into these tokens. The result suggests that the encoder learns a similar strategy or that the model struggles to use the last token to encode the MWP and candidate solutions. In our experiments,

Config	SVR	SVR-R	SVR-E
Best Config	**46.7**	**47.5**	66.3
Training Objective			
BCE	46.1	47.1	**66.8**
Decoder Attention			
Causal	46.0	46.5	63.4
Pooling			
eos	45.5	46.3	65.2
mean	46.3	47.0	66.0
Training Size			
8	43.8	44.4	60.4
32	46.5	46.9	66.2
Mining			
None	46.2	46.6	64.4

Table 5.2: Ablation results. SVR, SVR-R, and SVR-E denote the single view ranking model results using the postfix, verifier, and beam-4 oracles respectively. The best config uses MS loss, full decoder attention, attention pooling, 16 beams, and MS mining. The results show the replacement of one component at a time.

the eos-token pool would collapse quite often than the mean pool, suggesting that the latter is true.

Training Size

We analyze the impact of the number of training samples to train the verifier, similar to previous work [Shen *et al.* 2021; Cobbe *et al.* 2021]. We find that 16 performs substantially better than 8 and 32. Our results are consistent with the 20 reported by Shen *et al.* [2021]. We notice that the model performs below SOTA when the training set is small. This indicates that the verifier requires more examples to properly learn the task at hand.

Mining

We notice an absolute accuracy drop of 0.9% when we do not use the multi-similarity miner. This is likely due to a significant number of easy examples making their way into the training objective. This hinders the learning potential of the model.

5.4.6 Multi-view and Verification

We investigate the performance of each view in the multi-view setup and single-view verification setup. The results are presented in Table 5.3. From the results, we can see that SVR-R has the best accuracies compared to the multi-view setups.

View	SV	MV	MVR	SVR	SVR-R	SVR-E
Prefix	40.7	42.6	46.1	44.2	**46.4**	66.1
Infix	41.0	42.1	**45.0**	43.9	44.8	65.7
Postfix	40.9	42.9	46.6	45.5	**46.7**	66.3
Oracle	-	48.3	52.6	51.2	69.6	73.1

Table 5.3: Performance of different views when using multi-view and verification learning objectives. Here SV represents the single view performance.

5.4.7 Qualitative Analysis

In Table 5.4, we provide a qualitative analysis of the performance of the SV, SVR, and SVR-R models on a selection of examples, each with various applied variations. These variations encompass three critical aspects: question sensitivity, reasoning ability, and structural invariance. Question Sensitivity refers to whether the model's answer is influenced by changes in the question itself. Reasoning Ability examines whether the model can accurately identify shifts in reasoning, particularly in response to subtle alterations in the problem text. Structural Invariance assesses whether the model remains consistent in the face of superficial structural changes that do not impact the answer or the reasoning required to solve the example.

Looking at the results, it becomes evident that the baseline SV model encounters challenges when faced with reasoning alterations and structural changes introduced in MWPs. This suggests that the model may rely on patterns it learned during training to solve these problems. The SVR model does not fully resolve this. For instance, in the third example, both models performs correctly. However, in the fourth example, the introduction of irrelevant information causes both of the models to produce incorrect solutions. Importantly, all the generated solutions exhibit valid syntax. This implies that there's no immediate need for specialized neural architectures to ensure syntactically valid expressions. Notably, many incorrect answers from the models are associated with questions containing additional quantities, which can mislead the models.

Another observation is that correct solutions tend to appear higher in the beam output, yet the models do not generate all possible equivalent expressions. Furthermore, the generated solutions are not always identical to the ground truth but can still evaluate to the correct answer. This suggests that the model's approach to solving MWPs may differ from what the ground truth suggests.

In example 3, we can observe that the verifier in the SRV-R model plays a crucial role in rectifying model mistakes by re-ranking the beams. However, the SVR model seems to exhibit a bias toward longer solutions, as demonstrated in examples 3 and 9. In contrast, the SV model leans toward shorter solutions and generates fewer variations, sometimes leading to incorrect longer solutions. Additionally, the ranker in SVR-R may not be effective when none of the candidate solutions are correct, highlighting the challenges associated with this model variant.

Question	Variation	Ground Truth	SV	SVR	SVR-R	Beam-4
faye was placing number0 pencils into rows with number1 pencils in each row . how many rows could she make ?	Structural Invariance	/ number0 number1	/ number0 number1	/ number0 number1	/ number0 number1	[/ number0 number1, - number0 number1, * number0 number1, + number0 number1]
faye was placing some pencils equally into number0 rows . if she had number1 pencils how many pencils did she place in each row ?	Reasoning Ability, Structural Invariance	/ number1 number0	/ number1 number0	/ number1 number0	/ number1 number0	[/ number1 number0, - number1 number0, + number1 number0, * number1 number0]
faye was placing her pencils into number0 rows with number1 pencils in each row . how many pencils does she have ?	Reasoning Ability	* number0 number1	* number0 number1	* number0 number1	* number0 number1	[* number0 number1, * number1 number0, * number0 number0, * number0 number2]
faye was placing her pencils and crayons into number0 rows with number1 pencils and number2 crayons in each row . how many pencils does she have ?	Reasoning Ability, Structural Invariance	* number0 number1	* number1 number2	+ * number1 number2 number0	+ * number1 number2 number0	[+ * number1 number2 number0, + * number0 number1 number2, + * number1 number0 number2, * number1 number2]
faye was placing her pencils into rows with number0 pencils in each row . she had number1 packs of pencils each one having number2 pencils . how many rows could she make ?	Reasoning Ability	* number1 / number2 number0	/ number1 number2	/ * number1 number2 number0	/ * number2 number1 number0	[/ * number1 number2 number0, / * number1 number0 number2, / * number2 number1 number0, / * number1 number3 number0]
faye had number0 packs of pencils each one having number1 pencils . she was placing her pencils into rows with number2 pencils in each row . how many rows could she make ?	Reasoning Ability, Structural Invariance	* number0 / number1 number2	/ number0 number1	/ * number0 number1 number2	/ * number0 number1 number2	[/ * number0 number1 number2, + * number0 number1 number2, / * number0 number2 number1, / * number1 number2 number0]
faye was placing her pencils and crayons into number0 rows with number1 crayons and number2 pencils in each row . how many crayons does she have ?	Reasoning Ability, Structural Invariance, Question Sensitivity, Structural Invariance	* number0 number1	* number1 number0	* number0 number1	* number1 number0	[* number0 number1, * number1 number0, * number2 number0, * number2 number0 number1]
faye was placing her pencils and crayons into number0 rows with number1 crayons and number2 pencils in each row . how many pencils does she have ?	Reasoning Ability, Structural Invariance, Structural Invariance	* number0 number2	* number1 number2	* number1 number2	* / number1 number0 number2	[* number1 number2, * number1 number0, * * number1 number2 number0, * / number1 number0 number2]
faye was placing her pencils into rows with number0 pencils in each row . if she had number1 pencils and number2 crayons how many rows could she make ?	Structural Invariance	/ number1 number0	/ number1 number0	/ + number1 number0 number2	/ number1 number0	[/ + number1 number0 number2, / number1 number0, / * number1 number0 number2, / + number1 number2 number0]
faye was placing her pencils and crayons into number0 rows with number1 pencils and number2 crayons in each row . how many crayons does she have ?	Reasoning Ability, Structural Invariance, Question Sensitivity	* number0 number2	* number1 number2	* number0 number2	* number0 number2	[* number0 number2, * * number0 number2 number1, * number1 number2, * number0 number3]

Table 5.4: Qualitative performance of the different models. Correct and incorrect solutions are colored green and red, respectivey. Beam-4 represents the candidate solutions generated by the model. Questions adapted from SVAMP [Patel *et al.* 2021].

5.5 Conclusion

This chapter proposed a multi-tasking multi-view Transformer model to solve Math Word Problems (MWPs). The model is tasked with solving MWPs by generating several candidate solutions using multiple decoders, each of which generates a prefix, postfix, or infix view. The model is further given the task to rank all of these expressions to find the best candidate which is used as the final answer.

We conducted several experiments to prove the effectiveness of this strategy in Section 5.4. In particular, we compare the verification task against using a hand-crafted voting mechanism and simply choose the view the model can generate the best during training. We find that not only is the verifier useful for selecting the final solution, it is also an effective auxiliary task that resulted in an absolute accuracy improvement of 3.7%.

We also empirically show the importance of several design choices we made. Our results indicate that the number of samples used to train the verification task is quite critical for effective learning. In the best configuration, our model outperforms the current SOTA model by an absolute accuracy of 5.3%.

This chapter has shown that multiple decoders are better than one for a multi-view generation.

5.5.1 Future Work

Token-Level Verification

In Chapter 4 we proposed a strategy to create soft labels, which is left out for future work. We note that soft labels can be useful for a local fine-grained verification task. Cobbe *et al.* [2021] found that a token-level outperforms our global sequence-level verification task. However, they still use hard labels. The soft-label verification task predicts the probability of the token given all the tokens that have been generated. We note that after the first incorrect token, the rest of the expression will be incorrect, a problem faced with the seq2seq model that is often handled with teacher forcing, which is not directly applicable in this case.

An alternative would be to extend the soft-label verification task can a bi-directional verification task, where the soft-label ground truth probabilities are generated left-to-right and right-to-left. Alternatively, a correction task can be introduced, where the model is given a candidate expression which it should output if it is correct, otherwise it should correct the expression.

Verification Training objective

We adopted the multi-similarity (MS) loss function to train our model. However, several other metric learning objectives have been proposed in the literature that outperform the MS objective [Wang *et al.* 2018b; Sun *et al.* 2020]. These are surveyed in Musgrave *et al.* [2020] and Kaya and Bilge [2019]. Given that our model is only accurate 69.1%

of the time, it is worth investigating other learning objectives that can learn the task more effectively. We also note that it is possible to reap the benefits of the BCE objective alongside the MS objective. That is, we can train our model with both objectives. We leave the impact of this study for future work.

Verification Training Set

We investigate the impact of the size of the training samples used to train the verifier. We took on a simple strategy of using half-generated beams and half-randomly sampled templates. The beam search provides expressions the model most considers true, while the random sampling method gives a diverse sample pool to train the model. The diversity of the beam search decreases as we train the model. Cobbe *et al.* [2021] use temperature to control this which we did not consider for our experiments. Alternatively, a curriculum could be introduced that weighs the contribution of the two training set creation strategies.

Verification Task

We note that our multi-view BART model can only outperform SOTA when we use a verifier. However, the verification task can be added to the SOTA models Zhang *et al.* [2022a] and Jie *et al.* [2022]. We expect this to outperform our current implementation with BART. The relative improvements that the verifier will attribute to tree decoders and DAG-decoders can also give a better understanding of the importance of the decoder for verification. We note that the multi-view setup from Zhang *et al.* [2022a] can likely be improved by considering a generation-verification framework.

Chapter 6

Conclusion

This research aimed to investigate the ability of pre-trained models to solve Math Word Problems (MWPs) with limited data points to fine-tune. Research has reported outstanding results using pre-trained language models to solve natural language tasks [Nguyen *et al.* 2021; Kalyan *et al.* 2021] including with few-shot paradigms [Brown *et al.* 2020; Logeswaran *et al.* 2020; Chen *et al.* 2021a]. However, the sequence-to-sequence structure that these models have been shown to perform poorly on MWPs.

To successfully solve MWPs, the model has to learn to extract the relations between the quantities in an MWP and common sense constants such as π conditioned on a question that is asked about an unknown quantity. This information has to be translated into a valid abstract syntax tree that produces the correct value for the quantity question. MWPs are structured NLP reasoning tasks that SOTA has shown can be learned with graph-based encoders and tree-decoders or DAG-decoders.

Previous work found that these specialized neural networks can benefit from being pre-initialized with pre-trained embeddings. This indicates that PLMs are not necessarily incapable of this task; however, they fail to encode some information that graphs and trees can. This research aimed to investigate why this might be the case by investigating the data representation and the training objective from which we expect the PLMs to learn.

We hypothesized that the generative training objective is insufficient to enforce that the seq2seq PLMs learn the structural information of MWPs. To this end, we proposed to reinforce the learning objective with additional objectives that can enforce that the model understands the MWP reasoning task at hand, rather than just a simple NLP generation task, which tends to be robust to minor errors [Shen *et al.* 2021].

In particular, in Chapter 4 we propose a multi-view generation objective where the model has to generate three AST traversals. We hypothesized that if the seq2seq model can learn to generate all views of the AST, it should also have a better internal representation of the AST than if it generates only one. We conduct a series of experiments showing that our PLM can generate each view of the AST better when it learns of them together rather than individually.

Furthermore, in Chapter 4 we found that most works in literature mask quantities appearing in MWP by a quantity index counter to simplify the task. We hypothesized that this is one of the reasons for the reported shallow heuristics learned by PLMs. We proposed replacing this with a random-quantity indicator mask and ran experiments to show the effectiveness of this proposal. A 42.9% accuracy is achieved by the multi-view AST traversals with the random indicator, an increase of 2.3% over our best baseline's accuracy of 40.6% on the adversarial SVAMP [Patel *et al.* 2021] dataset.

In addition, in Chapter 4 we find that the correct answer lurks among the generated views. That is, if we can effectively select the correct solution among the candidates rather than as was done in the literature before [Zhang *et al.* 2022a], the proposed multi-view AST model can yield an accuracy of 48.3%, an improvement of 7.7% over our best baseline. This forms the basis of our work in Chapter 5.

In particular, Chapter 5 proposes a multi-view voting framework and a multi-view ranking framework to select the best answer from our multi-view setup proposed in Chapter 4. The voting framework returns the most consistent answer from the multiple views, while the ranking framework uses a single MLP layer as a verifier that generates scores used to rank the answers from the multiple views. We proved our hypothesis that both our multi-view selectors outperform beam search [Shen and Jin 2020; Meng and Rumshisky 2019], while the verifier in turn outperforms the voting mechanism.

In addition to using the verifier as a multi-view selector, we train it concurrently with our model as an auxiliary task, unlike previous work [Shen *et al.* 2021; Cobbe *et al.* 2021]. This auxiliary task allows the decoder to learn which expressions are incorrect in addition to which the correct expression. The experiments we conducted in Chapter 5 show that the verifier is effective as an auxiliary task and as a multi-view selector. The multi-view ranking mechanism yields accuracy improvements of 9.7% and 7.4% on our random indicator single-view and multi-view setup from Chapter 4, respectively.

Our experimental results prove the hypothesis we set out to test: the generative training objective can be reinforced with auxiliary tasks that can guide it to learn the structural information in MWPs with only limited datasets. In particular, in this research, we showed that the multi-view task and the verification task can be used as strong auxiliary tasks. Furthermore, they can be used together to generate the final answer, yielding superior results than the single view generation objective.

For future work, we propose extending our model agnostic framework to SOTA non-sequential decoders Jie *et al.* [2022] and Xie and Sun [2019] to compare the relative performance compared to their results and ours. Our multi-view tasks can also incorporate more data augmentations such as DAGs [Jie *et al.* 2022; Cao *et al.* 2021] and M-Trees [Wang *et al.* 2022a]. The verification objective we proposed evaluates an entire expression. This can be extended to a more granular level, which should improve its performance [Cobbe *et al.* 2021]. Furthermore, we can add several more objectives to guide our model [Qin *et al.* 2021; Liang *et al.* 2021].

Furthermore, we suggest that it would be a great research direction to distill the knowledge of tree decoders and DAGs into sequential decoders or to apply view consistency between these decoders to see if the sequential decoder can learn to encode structural

data.

References

[Allahyari *et al.* 2017] Mehdi Allahyari, Seyedamin Pouriyeh, Mehdi Assefi, Saeid Safaei, Elizabeth D Trippe, Juan B Gutierrez, and Krys Kochut. Text summarization techniques: a brief survey. *arXiv preprint arXiv:1707.02268*, 2017.

[Amini *et al.* 2019] Aida Amini, Saadia Gabriel, Peter Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. Mathqa: Towards interpretable math word problem solving with operation-based formalisms. *arXiv preprint arXiv:1905.13319*, 2019.

[Ba *et al.* 2016] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. *Layer Normalization*, 2016.

[Bahdanau *et al.* 2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

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

[Bengio *et al.* 2000] Yoshua Bengio, Réjean Ducharme, and Pascal Vincent. A neural probabilistic language model. *Advances in neural information processing systems*, 13, 2000.

[Bobrow 1964] Daniel G. Bobrow. A question-answering system for high school algebra word problems. In *Proceedings of the October 27-29, 1964, Fall Joint Computer Conference, Part I*, AFIPS '64 (Fall, part I), page 591–614, New York, NY, USA, 1964. Association for Computing Machinery.

[Boureau *et al.* 2010a] Y-Lan Boureau, Francis Bach, Yann LeCun, and Jean Ponce. Learning mid-level features for recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 2559–2566. IEEE, 2010.

[Boureau *et al.* 2010b] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

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

[Bucher *et al.* 2016] Maxime Bucher, Stéphane Herbin, and Frédéric Jurie. Hard negative mining for metric learning based zero-shot classification. In *Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14*, pages 524–531. Springer, 2016.

[Cakir *et al.* 2019] Fatih Cakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. Deep metric learning to rank. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1861–1870, 2019.

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

[Carion *et al.* 2020] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part I 16*, pages 213–229. Springer, 2020.

[Çelik *et al.* 2022] Ege Yiğit Çelik, Zeynel Orulluoğlu, RIDVAN MERTOĞLU, and Selma Tekir. Asking the right questions to solve algebraic word problems. *Turkish Journal of Electrical Engineering and Computer Sciences*, 30(7):2672–2687, 2022.

[Charniak 1969] Eugene Charniak. Computer solution of calculus word problems. In *Proceedings of the 1st International Joint Conference on Artificial Intelligence*, IJCAI'69, page 303–316, San Francisco, CA, USA, 1969. Morgan Kaufmann Publishers Inc.

[Charton *et al.* 2021] François Charton, Amaury Hayat, and Guillaume Lample. Learning advanced mathematical computations from examples. In *International Conference on Learning Representations*, 2021.

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

[Chen *et al.* 2016] Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*, 2016.

[Chen *et al.* 2018] Qian Chen, Zhen-Hua Ling, and Xiaodan Zhu. Enhancing sentence embedding with generalized pooling. *arXiv preprint arXiv:1806.09828*, 2018.

[Chen *et al.* 2020a] Tianlang Chen, Jiajun Deng, and Jiebo Luo. Adaptive offline quintuplet loss for image-text matching. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIII 16*, pages 549–565. Springer, 2020.

[Chen *et al.* 2020b] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[Chen *et al.* 2021a] Haoxing Chen, Huaxiong Li, Yaohui Li, and Chunlin Chen. Sparse spatial transformers for few-shot learning. *arXiv preprint arXiv:2109.12932*, 2021.

[Chen *et al.* 2021b] Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, et al. Finqa: A dataset of numerical reasoning over financial data. *arXiv preprint arXiv:2109.00122*, 2021.

[Cheng *et al.* 2016] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*, 2016.

[Chiang and Chen 2018] Ting-Rui Chiang and Yun-Nung Chen. Semantically-aligned equation generation for solving and reasoning math word problems. *arXiv preprint arXiv:1811.00720*, 2018.

[Cho *et al.* 2014] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[Choi *et al.* 2021] Hyunjin Choi, Judong Kim, Seongho Joe, and Youngjune Gwon. Evaluation of bert and albert sentence embedding performance on downstream nlp tasks. In *2020 25th International conference on pattern recognition (ICPR)*, pages 5482–5487. IEEE, 2021.

[Chopra *et al.* 2005] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[Chung *et al.* 2014] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[Church *et al.* 2021] Kenneth Ward Church, Zeyu Chen, and Yanjun Ma. Emerging trends: A gentle introduction to fine-tuning. *Natural Language Engineering*, 27(6):763–778, 2021.

[Clark *et al.* 2019] Kevin Clark, Urvashi Khandelwal, Omer Levy, and Christopher D Manning. What does bert look at? an analysis of bert's attention. *arXiv preprint arXiv:1906.04341*, 2019.

[Clark *et al.* 2020] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*, 2020.

[Cobbe *et al.* 2021] Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

[Collatz 2014] Lothar Collatz. *Functional analysis and numerical mathematics*. Academic Press, 2014.

[Collobert and Weston 2008] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167, 2008.

[Cui *et al.* 2016] Yin Cui, Feng Zhou, Yuanqing Lin, and Serge Belongie. Fine-grained categorization and dataset bootstrapping using deep metric learning with humans in the loop. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1153–1162, 2016.

[Dai and Le 2015] Andrew M. Dai and Quoc V. Le. Semi-supervised sequence learning. In *NIPS*, 2015.

[Dai *et al.* 2019] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

[Devlin *et al.* 2018] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[Dietterich and others 2002] Thomas G Dietterich et al. Ensemble learning. *The handbook of brain theory and neural networks*, 2(1):110–125, 2002.

[Ding *et al.* 2015] Shengyong Ding, Liang Lin, Guangrun Wang, and Hongyang Chao. Deep feature learning with relative distance comparison for person re-identification. *Pattern Recognition*, 48(10):2993–3003, 2015.

[Dong *et al.* 2017] Qi Dong, Shaogang Gong, and Xiatian Zhu. Class rectification hard mining for imbalanced deep learning. In *Proceedings of the IEEE international conference on computer vision*, pages 1851–1860, 2017.

[Dong *et al.* 2020] Xibin Dong, Zhiwen Yu, Wenming Cao, Yifan Shi, and Qianli Ma. A survey on ensemble learning. *Frontiers of Computer Science*, 14:241–258, 2020.

[Dosovitskiy *et al.* 2020] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[Dou 2022] Qingyun Dou. *Improving Attention-based Sequence-to-sequence Models*. PhD thesis, University of Cambridge, 2022.

[Eberts and Ulges 2019] Markus Eberts and Adrian Ulges. Span-based joint entity and relation extraction with transformer pre-training. *arXiv preprint arXiv:1909.07755*, 2019.

[Fan *et al.* 2021] Angela Fan, Shruti Bhosale, Holger Schwenk, Zhiyi Ma, Ahmed El-Kishky, Siddharth Goyal, Mandeep Baines, Onur Celebi, Guillaume Wenzek, Vishrav Chaudhary, et al. Beyond english-centric multilingual machine translation. *J. Mach. Learn. Res.*, 22(107):1–48, 2021.

[Farouk 2019] Mamdouh Farouk. Measuring sentences similarity: a survey. *arXiv preprint arXiv:1910.03940*, 2019.

[Fawagreh *et al.* 2014] Khaled Fawagreh, Mohamed Medhat Gaber, and Eyad Elyan. Random forests: from early developments to recent advancements. *Systems Science & Control Engineering: An Open Access Journal*, 2(1):602–609, 2014.

[Feigenbaum and Feldman 1963] Edward A. Feigenbaum and Julian Feldman. *Computers and Thought*. McGraw-Hill, Inc., USA, 1963.

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

[Freitag and Al-Onaizan 2017] Markus Freitag and Yaser Al-Onaizan. Beam search strategies for neural machine translation. *arXiv preprint arXiv:1702.01806*, 2017.

[Gage 1994] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, 1994.

[Gehring *et al.* 2017a] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, abs/1705.03122, 2017.

[Gehring *et al.* 2017b] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. In *International conference on machine learning*, pages 1243–1252. PMLR, 2017.

[Gers *et al.* 2000] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[Gholamalinezhad and Khosravi 2020] Hossein Gholamalinezhad and Hossein Khosravi. Pooling methods in deep neural networks, a review. *arXiv preprint arXiv:2009.07485*, 2020.

[Goldberger *et al.* 2004] Jacob Goldberger, Geoffrey E Hinton, Sam Roweis, and Russ R Salakhutdinov. Neighbourhood components analysis. *Advances in neural information processing systems*, 17, 2004.

[Goldwasser and Roth 2014] Dan Goldwasser and Dan Roth. Learning from natural instructions. *Machine learning*, 94(2):205–232, 2014.

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Goodfellow *et al.* 2020] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[Griffith and Kalita 2020] Kaden Griffith and Jugal Kalita. Solving arithmetic word problems using transformer and pre-processing of problem texts. In *Proceedings*

of the 17th International Conference on Natural Language Processing (ICON), pages 76–84, 2020.

[Grira *et al.* 2004] Nizar Grira, Michel Crucianu, and Nozha Boujemaa. Unsupervised and semi-supervised clustering: a brief survey. *A review of machine learning techniques for processing multimedia content*, 1(2004):9–16, 2004.

[Gupta and Malhotra 2015] Gaurav Gupta and Sumit Malhotra. Text document tokenization for word frequency count using rapid miner (taking resume as an example). *Int. J. Comput. Appl*, 975:8887, 2015.

[Gutmann and Hyvärinen 2010] Michael Gutmann and Aapo Hyvärinen. Noise-contrastive estimation: A new estimation principle for unnormalized statistical models. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 297–304. JMLR Workshop and Conference Proceedings, 2010.

[Gutmann and Hyvärinen 2012] Michael U Gutmann and Aapo Hyvärinen. Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics. *Journal of machine learning research*, 13(2), 2012.

[Hadsell *et al.* 2006] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[Hamilton *et al.* 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[Hao *et al.* 2020] Yongchang Hao, Shilin He, Wenxiang Jiao, Zhaopeng Tu, Michael Lyu, and Xing Wang. Multi-task learning with shared encoder for non-autoregressive machine translation. *arXiv preprint arXiv:2010.12868*, 2020.

[Harwood *et al.* 2017] Ben Harwood, Vijay Kumar BG, Gustavo Carneiro, Ian Reid, and Tom Drummond. Smart mining for deep metric learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2821–2829, 2017.

[Haykin 1998] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR, USA, 2nd edition, 1998.

[He and Xiao 2022] Daijun He and Jing Xiao. Goal selection and feedback for solving math word problems. *Applied Intelligence*, pages 1–15, 2022.

[He *et al.* 2016] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[He *et al.* 2020a] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9729–9738, 2020.

[He *et al.* 2020b] Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. Deberta: Decoding-enhanced bert with disentangled attention. *ArXiv*, abs/2006.03654, 2020.

[Henaff 2020] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In *International conference on machine learning*, pages 4182–4192. PMLR, 2020.

[Hendrycks and Gimpel 2016] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[Hendrycks *et al.* 2021] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[Henighan *et al.* 2020] Tom Henighan, Jared Kaplan, Mor Katz, Mark Chen, Christopher Hesse, Jacob Jackson, Heewoo Jun, Tom B Brown, Prafulla Dhariwal, Scott Gray, et al. Scaling laws for autoregressive generative modeling. *arXiv preprint arXiv:2010.14701*, 2020.

[Hermans *et al.* 2017] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*, 2017.

[Ho 1995] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[Hochreiter and Schmidhuber 1997] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[Hong *et al.* 2021] Yining Hong, Qing Li, Daniel Ciao, Siyuan Huang, and Song-Chun Zhu. Learning by fixing: Solving math word problems with weak supervision. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4959–4967, 2021.

[Honnibal and Montani 2017] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[Horiguchi *et al.* 2019] Shota Horiguchi, Daiki Ikami, and Kiyoharu Aizawa. Significance of softmax-based features in comparison to distance metric learning-based features. *IEEE transactions on pattern analysis and machine intelligence*, 42(5):1279–1285, 2019.

[Huang *et al.* 2016] Danqing Huang, Shuming Shi, Chin-Yew Lin, Jian Yin, and Wei-Ying 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)*, pages 887–896, 2016.

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

[Inaguma *et al.* 2021a] Hirofumi Inaguma, Yosuke Higuchi, Kevin Duh, Tatsuya Kawahara, and Shinji Watanabe. Non-autoregressive end-to-end speech translation with parallel autoregressive rescoring. *arXiv preprint arXiv:2109.04411*, 2021.

[Inaguma *et al.* 2021b] Hirofumi Inaguma, Yosuke Higuchi, Kevin Duh, Tatsuya Kawahara, and Shinji Watanabe. Orthros: Non-autoregressive end-to-end speech translation with dual-decoder. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7503–7507. IEEE, 2021.

[Inan *et al.* 2016] Hakan Inan, Khashayar Khosravi, and Richard Socher. Tying word vectors and word classifiers: A loss framework for language modeling. *arXiv preprint arXiv:1611.01462*, 2016.

[Irie *et al.* 2016] Kazuki Irie, Zoltán Tüske, Tamer Alkhouli, Ralf Schlüter, Hermann Ney, et al. Lstm, gru, highway and a bit of attention: An empirical overview for language modeling in speech recognition. In *Interspeech*, pages 3519–3523, 2016.

[Jeschke and Richter 2007] Sabina Jeschke and Thomas Richter. Mathematics in virtual knowledge spaces: user adaptation by intelligent assistants. In *Intelligent Assistant Systems: Concepts, Techniques and Technologies*, pages 232–263. IGI Global, 2007.

[Jie *et al.* 2022] Zhanming Jie, Jierui Li, and Wei Lu. Learning to reason deductively: Math word problem solving as complex relation extraction. In *Annual Meeting of the Association for Computational Linguistics*, 2022.

[Joulin *et al.* 2016] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

[Kalchbrenner *et al.* 2014] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

[Kalchbrenner *et al.* 2016] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

[Kalyan *et al.* 2021] Katikapalli Subramanyam Kalyan, Ajit Rajasekharan, and Sivanesan Sangeetha. Ammus: A survey of transformer-based pretrained models in natural language processing. *arXiv preprint arXiv:2108.05542*, 2021.

[Kamath *et al.* 2020] Sanjay Kamath, Brigitte Grau, and Yue Ma. How to pre-train your model? comparison of different pre-training models for biomedical question answering. In *Machine Learning and Knowledge Discovery in Databases: International Workshops of ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part II*, pages 646–660. Springer, 2020.

[Kapralov *et al.* 2020] Stoyan N. Kapralov, P. K. Ivanova, and Stefka Bouyuklieva. The compmath competition: Solving math problems with computer algebra systems. *Engaging Young Students in Mathematics through Competitions — World Perspectives and Practices*, 2020.

[Kaya and Bilge 2019] Mahmut Kaya and Hasan Şakir Bilge. Deep metric learning: A survey. *Symmetry*, 11(9):1066, 2019.

[Kha Vu 2021] Chan Kha Vu. *Deep Metric Learning: A (Long) Survey*, 2021.

[Khan *et al.* 2022] Salman Khan, Muzammal Naseer, Munawar Hayat, Syed Waqas Zamir, Fahad Shahbaz Khan, and Mubarak Shah. Transformers in vision: A survey. *ACM computing surveys (CSUR)*, 54(10s):1–41, 2022.

[Khosla *et al.* 2020] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *Advances in Neural Information Processing Systems*, 33:18661–18673, 2020.

[Kim *et al.* 2020] Bugeun Kim, Kyung Seo Ki, Donggeon Lee, and Gahgene 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)*, pages 3768–3779, 2020.

[Kim 2022] Yunsu Kim. *Neural machine translation for low-resource scenarios*. PhD thesis, Dissertation, RWTH Aachen University, 2022, 2022.

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

[Koehn *et al.* 2007] Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th annual meeting of the association for computational linguistics companion volume proceedings of the demo and poster sessions*, pages 177–180, 2007.

[Koncel-Kedziorski *et al.* 2016] Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh 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*, pages 1152–1157, 2016.

[Kowsari *et al.* 2019] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[Kubota *et al.* 2022] Hazumi Kubota, Yuta Tokuoka, Takahiro G Yamada, and Akira Funahashi. Symbolic integration by integrating learning models with different strengths and weaknesses. *IEEE Access*, 10:47000–47010, 2022.

[Kudo and Richardson 2018] Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

[Kula 2007] Fulya Kula. Verschaffel, l., greer, b., and de corte, e. (2000). making sense of word problems. netherlands: Swets & zeitlinger. 2007.

[Kullback 1997] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[Kumar *et al.* 2016] Ankit Kumar, Ozan Irsoy, Peter Ondruska, Mohit Iyyer, James Bradbury, Ishaan Gulrajani, Victor Zhong, Romain Paulus, and Richard Socher. Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, pages 1378–1387. PMLR, 2016.

[Kwiatkowski *et al.* 2013] Tom Kwiatkowski, Eunsol Choi, Yoav Artzi, and Luke Zettlemoyer. Scaling semantic parsers with on-the-fly ontology matching. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1545–1556, 2013.

[Lamb *et al.* 2016] Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

[Lample and Charton 2019] Guillaume Lample and François Charton. Deep learning for symbolic mathematics. *arXiv preprint arXiv:1912.01412*, 2019.

[Lample and Conneau 2019] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019.

[Lan *et al.* 2019] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019.

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

[Lang and Schubert 2020] Andreas Lang and Erich Schubert. Betula: Numerically stable cf-trees for birch clustering. *ArXiv*, abs/2006.12881, 2020.

[LeCun *et al.* 1989] Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

[LeCun *et al.* 1998] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Lee *et al.* 2016] Chen-Yu Lee, Patrick W Gallagher, and Zhuowen Tu. Generalizing pooling functions in convolutional neural networks: Mixed, gated, and tree. In *Artificial intelligence and statistics*, pages 464–472. PMLR, 2016.

[Lee *et al.* 2018] Kuang-Huei Lee, Xi Chen, Gang Hua, Houdong Hu, and Xiaodong He. Stacked cross attention for image-text matching. In *Proceedings of the European conference on computer vision (ECCV)*, pages 201–216, 2018.

[Lee *et al.* 2020] Jongpil Lee, Nicholas J Bryan, Justin Salamon, Zeyu Jin, and Juhan Nam. Metric learning vs classification for disentangled music representation learning. *arXiv preprint arXiv:2008.03729*, 2020.

[Lee *et al.* 2021] Donggeon Lee, Kyung Seo Ki, Bugeun Kim, and Gahgene Gweon. Tm-generation model: a template-based method for automatically solving mathematical word problems. *The Journal of Supercomputing*, 77(12):14583–14599, 2021.

[Lewis *et al.* 2019] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*, 2019.

[Li *et al.* 2015] Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*, 2015.

[Li *et al.* 2018] Pengcheng Li, Yan Song, Ian Vince McLoughlin, Wu Guo, and Li-Rong Dai. An attention pooling based representation learning method for speech emotion recognition. 2018.

[Li *et al.* 2019] Jierui Li, Lei Wang, Jipeng Zhang, Yan Wang, Bing Tian Dai, and Dongxiang 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*, pages 6162–6167, 2019.

[Li *et al.* 2020] Shucheng Li, Lingfei Wu, Shiwei Feng, Fangli Xu, Fengyuan Xu, and Sheng 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.

[Li *et al.* 2021a] Jianquan Li, Xiaokang Liu, Wenpeng Yin, Min Yang, Liqun Ma, and Yaohong Jin. Empirical evaluation of multi-task learning in deep neural networks for natural language processing. *Neural Computing and Applications*, 33(9):4417–4428, 2021.

[Li *et al.* 2021b] Xiaoxu Li, Xiaochen Yang, Zhanyu Ma, and Jing-Hao Xue. Deep metric learning for few-shot image classification: A selective review. *arXiv preprint arXiv:2105.08149*, 2021.

[Li *et al.* 2021c] Zhongli Li, Wenxuan Zhang, Chao Yan, Qingyu Zhou, Chao Li, Hongzhi Liu, and Yunbo Cao. Seeking patterns, not just memorizing proce-

dures: Contrastive learning for solving math word problems. *arXiv preprint arXiv:2110.08464*, 2021.

[Li *et al.* 2022a] Ailisi Li, Xueyao Jiang, Bang Liu, Jiaqing Liang, and Yanghua Xiao. Tackling math word problems with fine-to-coarse abstracting and reasoning. *arXiv preprint arXiv:2205.08274*, 2022.

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

[Li 2017] Yuxi Li. Deep reinforcement learning: An overview. *arXiv preprint arXiv:1701.07274*, 2017.

[Liang and Zhang 2021] Zhenwen Liang and Xiangliang Zhang. Solving math word problems with teacher supervision. In *IJCAI*, pages 3522–3528, 2021.

[Liang *et al.* 2021] Zhenwen Liang, Jipeng Zhang, Lei Wang, Wei Qin, Yunshi Lan, Jie Shao, and Xiangliang Zhang. Mwp-bert: Numeracy-augmented pre-training for math word problem solving. *arXiv preprint arXiv:2107.13435*, 2021.

[Lin *et al.* 2017a] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[Lin *et al.* 2017b] Zhouhan Lin, Minwei Feng, Cicero Nogueira dos Santos, Mo Yu, Bing Xiang, Bowen Zhou, and Yoshua Bengio. A structured self-attentive sentence embedding. *arXiv preprint arXiv:1703.03130*, 2017.

[Lin *et al.* 2022] Tianyang Lin, Yuxin Wang, Xiangyang Liu, and Xipeng Qiu. A survey of transformers. *AI Open*, 2022.

[Lin 2021] Xin Lin. Investigating the unique predictors of word-problem solving using meta-analytic structural equation modeling. *Educational Psychology Review*, 33(3):1097–1124, 2021.

[Liu *et al.* 2016] Yang Liu, Chengjie Sun, Lei Lin, and Xiaolong Wang. Learning natural language inference using bidirectional lstm model and inner-attention. *arXiv preprint arXiv:1605.09090*, 2016.

[Liu *et al.* 2017] Jun Liu, Gang Wang, Ping Hu, Ling-Yu Duan, and Alex C Kot. Global context-aware attention lstm networks for 3d action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1647–1656, 2017.

[Liu *et al.* 2019a] Chunxiao Liu, Zhendong Mao, An-An Liu, Tianzhu Zhang, Bin Wang, and Yongdong Zhang. Focus your attention: A bidirectional focal attention network for image-text matching. In *Proceedings of the 27th ACM international conference on multimedia*, pages 3–11, 2019.

[Liu *et al.* 2019b] Qianying Liu, Wenyv Guan, Sujian Li, and Daisuke 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), pages 2370–2379, 2019.

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

[Liu *et al.* 2020a] Xiaodong Liu, Yu Wang, Jianshu Ji, Hao Cheng, Xueyun Zhu, Emmanuel Awa, Pengcheng He, Weizhu Chen, Hoifung Poon, Guihong Cao, et al. The microsoft toolkit of multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:2002.07972*, 2020.

[Liu *et al.* 2020b] Yinhan Liu, Jiatao Gu, Naman Goyal, Xian Li, Sergey Edunov, Marjan Ghazvininejad, Mike Lewis, and Luke Zettlemoyer. Multilingual denoising pre-training for neural machine translation. *Transactions of the Association for Computational Linguistics*, 8:726–742, 2020.

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

[Liu *et al.* 2022] Jiayu Liu, Zhenya Huang, Xin Lin, Qi Liu, Jianhui Ma, and Enhong Chen. A cognitive solver with autonomously knowledge learning for reasoning mathematical answers. In *2022 IEEE International Conference on Data Mining (ICDM)*, pages 269–278. IEEE, 2022.

[Logeswaran *et al.* 2020] Lajanugen Logeswaran, Ann Lee, Myle Ott, Honglak Lee, Marc'Aurelio Ranzato, and Arthur Szlam. Few-shot sequence learning with transformers. *arXiv preprint arXiv:2012.09543*, 2020.

[Loshchilov and Hutter 2017] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[Lukasik *et al.* 2020] Michal Lukasik, Srinadh Bhojanapalli, Aditya Menon, and Sanjiv Kumar. Does label smoothing mitigate label noise? In *International Conference on Machine Learning*, pages 6448–6458. PMLR, 2020.

[Luong *et al.* 2015] Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[Lyon 2022] Harper Lyon. Tracking xenophobic terminology on twitter using nlp. 2022.

[Magister *et al.* 2022] Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek,

Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. *arXiv preprint arXiv:2212.08410*, 2022.

[Maini *et al.* 2020] Pratyush Maini, Keshav Kolluru, Danish Pruthi, et al. Why and when should you pool? analyzing pooling in recurrent architectures. *arXiv preprint arXiv:2005.00159*, 2020.

[Malte and Ratadiya 2019] Aditya Malte and Pratik Ratadiya. Evolution of transfer learning in natural language processing. *arXiv preprint arXiv:1910.07370*, 2019.

[McCann *et al.* 2018] Bryan McCann, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. The natural language decathlon: Multitask learning as question answering. *arXiv preprint arXiv:1806.08730*, 2018.

[McFee and Lanckriet 2010] Brian McFee and Gert R Lanckriet. Metric learning to rank. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 775–782, 2010.

[Meister *et al.* 2020a] Clara Meister, Tim Vieira, and Ryan Cotterell. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809, 2020.

[Meister *et al.* 2020b] Clara Meister, Tim Vieira, and Ryan Cotterell. If beam search is the answer, what was the question? *arXiv preprint arXiv:2010.02650*, 2020.

[Meng and Rumshisky 2019] Yuanliang Meng and Anna Rumshisky. Solving math word problems with double-decoder transformer. *arXiv preprint arXiv:1908.10924*, 2019.

[Michelbacher 2013] Lukas Michelbacher. Multi-word tokenization for natural language processing. 2013.

[Mielke *et al.* 2021] Sabrina J Mielke, Zaid Alyafeai, Elizabeth Salesky, Colin Raffel, Manan Dey, Matthias Gallé, Arun Raja, Chenglei Si, Wilson Y Lee, Benoît Sagot, et al. Between words and characters: A brief history of open-vocabulary modeling and tokenization in nlp. *arXiv preprint arXiv:2112.10508*, 2021.

[Mikolov *et al.* 2013] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[Mohamed *et al.* 2021] Shereen A Mohamed, Ashraf A Elsayed, YF Hassan, and Mohamed A Abdou. Neural machine translation: past, present, and future. *Neural Computing and Applications*, 33:15919–15931, 2021.

[Morales and Escalante 2022] Eduardo F Morales and Hugo Jair Escalante. A brief introduction to supervised, unsupervised, and reinforcement learning. In *Biosignal Processing and Classification Using Computational Learning and Intelligence*, pages 111–129. Elsevier, 2022.

[Morton and Qu 2013] Kyle Morton and Yanzhen Qu. A novel framework for math

word problem solving. *International Journal of Information and Education Technology*, pages 88–93, 2013.

[Mosbach *et al.* 2020] Marius Mosbach, Anna Khokhlova, Michael A Hedderich, and Dietrich Klakow. On the interplay between fine-tuning and sentence-level probing for linguistic knowledge in pre-trained transformers. *arXiv preprint arXiv:2010.02616*, 2020.

[Muennighoff 2022] Niklas Muennighoff. Sgpt: Gpt sentence embeddings for semantic search. *arXiv preprint arXiv:2202.08904*, 2022.

[Muffo *et al.* 2022] Matteo Muffo, Aldo Cocco, and Enrico Bertino. Evaluating transformer language models on arithmetic operations using number decomposition. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 291–297, 2022.

[Mukherjee and Garain 2008] Anirban Mukherjee and Utpal Garain. A review of methods for automatic understanding of natural language mathematical problems. *Artif. Intell. Rev.*, 29:93–122, 04 2008.

[Müller *et al.* 2019] Rafael Müller, Simon Kornblith, and Geoffrey E Hinton. When does label smoothing help? *Advances in neural information processing systems*, 32, 2019.

[Musgrave *et al.* 2020] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. A metric learning reality check. In *European Conference on Computer Vision*, pages 681–699. Springer, 2020.

[Nair and Hinton 2010] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[Nederhof 2003] Mark-Jan Nederhof. Weighted deductive parsing and knuth's algorithm. *Computational Linguistics*, 29(1):135–143, 2003.

[Neelakantan *et al.* 2022] Arvind Neelakantan, Tao Xu, Raul Puri, Alec Radford, Jesse Michael Han, Jerry Tworek, Qiming Yuan, Nikolas Tezak, Jong Wook Kim, Chris Hallacy, et al. Text and code embeddings by contrastive pre-training. *arXiv preprint arXiv:2201.10005*, 2022.

[Nguyen *et al.* 2021] Minh-Tien Nguyen, Dung Tien Le, and Linh Le. Transformers-based information extraction with limited data for domain-specific business documents. *Engineering Applications of Artificial Intelligence*, 97:104100, 2021.

[Nogueira *et al.* 2021] Rodrigo Nogueira, Zhiying Jiang, and Jimmy Lin. Investigating the limitations of transformers with simple arithmetic tasks. *arXiv preprint arXiv:2102.13019*, 2021.

[Noorbakhsh *et al.* 2021] Kimia Noorbakhsh, Modar Sulaiman, Mahdi Sharifi, Kallol Roy, and Pooyan Jamshidi. Pretrained language models are symbolic mathematics solvers too! *ArXiv*, abs/2110.03501, 2021.

[Oh Song *et al.* 2017] Hyun Oh Song, Stefanie Jegelka, Vivek Rathod, and Kevin Murphy. Deep metric learning via facility location. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5382–5390, 2017.

[Oord *et al.* 2018] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[Parikh *et al.* 2016] Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*, 2016.

[Patel *et al.* 2021] Arkil Patel, S. Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *NAACL*, 2021.

[Pikos *et al.* 2021] Piotr Pikos, Henryk Michalewski, and Mateusz Malinowski. Measuring and improving bert's mathematical abilities by predicting the order of reasoning. *arXiv preprint arXiv:2106.03921*, 2021.

[Press and Wolf 2016] Ofir Press and Lior Wolf. Using the output embedding to improve language models. *arXiv preprint arXiv:1608.05859*, 2016.

[Prétet *et al.* 2020] Laure Prétet, Gaël Richard, and Geoffroy Peeters. Learning to rank music tracks using triplet loss. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 511–515. IEEE, 2020.

[Qin *et al.* 2020] Jinghui Qin, Lihui Lin, Xiaodan Liang, Rumin Zhang, and Liang Lin. Semantically-aligned universal tree-structured solver for math word problems. *arXiv preprint arXiv:2010.06823*, 2020.

[Qin *et al.* 2021] Jinghui Qin, Xiaodan Liang, Yining Hong, Jianheng Tang, and Liang Lin. Neural-symbolic solver for math word problems with auxiliary tasks. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5870–5881, Online, August 2021. Association for Computational Linguistics.

[Radford *et al.* 2018] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

[Radford *et al.* 2019] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[Raffel *et al.* 2020] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J Liu, et al. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(140):1–67, 2020.

[Reed and MarksII 1999] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[Reimers and Gurevych 2019] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[Ridgeway and Mozer 2018] Karl Ridgeway and Michael C Mozer. Learning deep disentangled embeddings with the f-statistic loss. *Advances in neural information processing systems*, 31, 2018.

[Robaidek *et al.* 2018] Benjamin Robaidek, Rik Koncel-Kedziorski, and Hannaneh Hajishirzi. Data-driven methods for solving algebra word problems. *arXiv preprint arXiv:1804.10718*, 2018.

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

[Roy and Roth 2017] Subhro Roy and Dan Roth. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

[Ruder 2017] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[Sagi and Rokach 2018] Omer Sagi and Lior Rokach. Ensemble learning: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4):e1249, 2018.

[Sahu and Anand 2018] Sunil Kumar Sahu and Ashish Anand. Drug-drug interaction extraction from biomedical texts using long short-term memory network. *Journal of biomedical informatics*, 86:15–24, 2018.

[Salehinejad *et al.* 2017] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

[Sathya *et al.* 2013] Ramadass Sathya, Annamma Abraham, et al. Comparison of supervised and unsupervised learning algorithms for pattern classification. *International Journal of Advanced Research in Artificial Intelligence*, 2(2):34–38, 2013.

[Saxton *et al.* 2019] David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*, 2019.

[Scheibling-Sève *et al.* 2020] Calliste Scheibling-Sève, Elena Pasquinelli, and Emmanuel Sander. Assessing conceptual knowledge through solving arithmetic word problems. *Educational Studies in Mathematics*, 103:293 – 311, 2020.

[Schmidhuber 2015] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks : the official journal of the International Neural Network Society*, 61:85–117, 2015.

[Schroff *et al.* 2015] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings*

of the IEEE conference on computer vision and pattern recognition, pages 815–823, 2015.

[Schubert and Gertz 2018] Erich Schubert and Michael Gertz. Numerically stable parallel computation of (co-) variance. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, pages 1–12, 2018.

[Schubert 2021] Erich Schubert. A triangle inequality for cosine similarity. In *Similarity Search and Applications: 14th International Conference, SISAP 2021, Dortmund, Germany, September 29–October 1, 2021, Proceedings*, pages 32–44. Springer, 2021.

[Sennrich *et al.* 2015] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2015.

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

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

[Shen *et al.* 2022] Yibin Shen, Qianying Liu, Zhuoyuan Mao, Fei Cheng, and Sadao Kurohashi. Textual enhanced contrastive learning for solving math word problems. *arXiv preprint arXiv:2211.16022*, 2022.

[Shieber *et al.* 1995] Stuart M Shieber, Yves Schabes, and Fernando CN Pereira. Principles and implementation of deductive parsing. *The Journal of logic programming*, 24(1-2):3–36, 1995.

[Shleifer and Rush 2020] Sam Shleifer and Alexander M Rush. Pre-trained summarization distillation. *arXiv preprint arXiv:2010.13002*, 2020.

[Shu *et al.* 2018] Bo Shu, Fuji Ren, and Yanwei Bao. Investigating lstm with k-max pooling for text classification. In *2018 11th International Conference on Intelligent Computation Technology and Automation (ICICTA)*, pages 31–34. IEEE, 2018.

[Sikaroudi *et al.* 2020] Milad Sikaroudi, Benyamin Ghojogh, Amir Safarpoor, Fakhri Karray, Mark Crowley, and Hamid R Tizhoosh. Offline versus online triplet mining based on extreme distances of histopathology patches. In *Advances in Visual Computing: 15th International Symposium, ISVC 2020, San Diego, CA, USA, October 5–7, 2020, Proceedings, Part I 15*, pages 333–345. Springer, 2020.

[Sohn 2016] Kihyuk Sohn. Improved deep metric learning with multi-class n-pair loss objective. *Advances in neural information processing systems*, 29, 2016.

[Speer 2019] Robyn Speer. *ftfy*. Zenodo, 2019. Version 5.5.

[Stahlberg 2020] Felix Stahlberg. Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418, 2020.

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

[Suárez-Paniagua and Segura-Bedmar 2018] Víctor Suárez-Paniagua and Isabel Segura-Bedmar. Evaluation of pooling operations in convolutional architectures for drug-drug interaction extraction. *BMC bioinformatics*, 19:39–47, 2018.

[Sun and Zhang 2021] Yan-Ping Sun and Min-Ling Zhang. Compositional metric learning for multi-label classification. *Frontiers of Computer Science*, 15:1–12, 2021.

[Sun *et al.* 2020] Yifan Sun, Changmao Cheng, Yuhan Zhang, Chi Zhang, Liang Zheng, Zhongdao Wang, and Yichen Wei. Circle loss: A unified perspective of pair similarity optimization. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6398–6407, 2020.

[Sun 2013] Shiliang Sun. A survey of multi-view machine learning. *Neural computing and applications*, 23:2031–2038, 2013.

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

[Tan *et al.* 2020] Zhixing Tan, Shuo Wang, Zonghan Yang, Gang Chen, Xuancheng Huang, Maosong Sun, and Yang Liu. Neural machine translation: A review of methods, resources, and tools. *AI Open*, 1:5–21, 2020.

[Tang *et al.* 2022] Tianyi Tang, Junyi Li, Wayne Xin Zhao, and Ji-Rong Wen. Mvp: Multi-task supervised pre-training for natural language generation. *arXiv preprint arXiv:2206.12131*, 2022.

[The MathWorks Inc. 2019] The MathWorks Inc. *Statistics and machine learning toolbox*, 2019.

[Tian *et al.* 2020] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XI 16*, pages 776–794. Springer, 2020.

[Ting and Witten 1997] Kai Ming Ting and Ian H Witten. Stacking bagged and dagged models. 1997.

[Tommasi *et al.* 2013] Tatiana Tommasi, Francesco Orabona, and Barbara Caputo. Learning categories from few examples with multi model knowledge transfer. *IEEE transactions on pattern analysis and machine intelligence*, 36(5):928–941, 2013.

[Torrey and Shavlik 2010] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global, 2010.

[Upadhyay *et al.* 2016] Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen

tau Yih. Learning from explicit and implicit supervision jointly for algebra word problems. In *Conference on Empirical Methods in Natural Language Processing*, 2016.

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

[Verma *et al.* 2014] Tanu Verma, Renu Renu, and Deepti Gaur. Tokenization and filtering process in rapidminer. *International Journal of Applied Information Systems*, 7(2):16–18, 2014.

[Verschaffel *et al.* 2020] Lieven Verschaffel, Stanislaw Schukajlow, Jon R. Star, and Wim Van Dooren. Word problems in mathematics education: a survey. *ZDM*, 52:1–16, 2020.

[Voita *et al.* 2019] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5797–5808, Florence, Italy, July 2019. Association for Computational Linguistics.

[Wallace *et al.* 2019] Eric Wallace, Yizhong Wang, Sujian Li, Sameer Singh, and Matt Gardner. Do nlp models know numbers? probing numeracy in embeddings. *arXiv preprint arXiv:1909.07940*, 2019.

[Wang and Chen 2020] Yu-An Wang and Yun-Nung Chen. What do position embeddings learn? an empirical study of pre-trained language model positional encoding. *CoRR*, abs/2010.04903, 2020.

[Wang and Jiang 2015] Shuohang Wang and Jing Jiang. Learning natural language inference with lstm. *arXiv preprint arXiv:1512.08849*, 2015.

[Wang *et al.* 2014] Jiang Wang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. Learning fine-grained image similarity with deep ranking. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1386–1393, 2014.

[Wang *et al.* 2016a] Faqiang Wang, Wangmeng Zuo, Liang Lin, David Zhang, and Lei Zhang. Joint learning of single-image and cross-image representations for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1288–1296, 2016.

[Wang *et al.* 2016b] Yequan Wang, Minlie Huang, Xiaoyan Zhu, and Li Zhao. Attention-based lstm for aspect-level sentiment classification. In *Proceedings of the 2016 conference on empirical methods in natural language processing*, pages 606–615, 2016.

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

[Wang *et al.* 2018a] Cheng Wang, Qian Zhang, Chang Huang, Wenyu Liu, and Xinggang Wang. Mancs: A multi-task attentional network with curriculum sampling for person re-identification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 365–381, 2018.

[Wang *et al.* 2018b] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu. Cosface: Large margin cosine loss for deep face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5265–5274, 2018.

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

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

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

[Wang *et al.* 2019b] Xun Wang, Xintong Han, Weilin Huang, Dengke Dong, and Matthew R Scott. Multi-similarity loss with general pair weighting for deep metric learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5022–5030, 2019.

[Wang *et al.* 2019c] Zihao Wang, Xihui Liu, Hongsheng Li, Lu Sheng, Junjie Yan, Xiaogang Wang, and Jing Shao. Camp: Cross-modal adaptive message passing for text-image retrieval. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 5764–5773, 2019.

[Wang *et al.* 2020] Changhan Wang, Kyunghyun Cho, and Jiatao Gu. Neural machine translation with byte-level subwords. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 9154–9160, 2020.

[Wang *et al.* 2022a] Bin Wang, Jiangzhou Ju, Yang Fan, Xin-Yu Dai, Shujian Huang, and Jiajun Chen. Structure-unified m-tree coding solver for mathword problem. *arXiv preprint arXiv:2210.12432*, 2022.

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

[Wang *et al.* 2022c] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Huai hsin Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171, 2022.

[Webster and Kit 1992] Jonathan J Webster and Chunyu Kit. Tokenization as the initial

phase in nlp. In *COLING 1992 volume 4: The 14th international conference on computational linguistics*, 1992.

[Wei *et al.* 2022] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. *ArXiv*, abs/2201.11903, 2022.

[Weiss *et al.* 2016] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3(1):1–40, 2016.

[Weisstein 2006] Eric W Weisstein. Pythagorean theorem. *https://mathworld. wolfram. com/*, 2006.

[Wen *et al.* 2015] Tsung-Hsien Wen, Milica Gasic, Nikola Mrksic, Pei-Hao Su, David Vandyke, and Steve Young. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. *arXiv preprint arXiv:1508.01745*, 2015.

[Weng 2021] Lilian Weng. Contrastive representation learning. *lilianweng.github.io*, May 2021.

[Wies *et al.* 2021] Noam Wies, Yoav Levine, Daniel Jannai, and Amnon Shashua. Which transformer architecture fits my data? a vocabulary bottleneck in self-attention. In *International Conference on Machine Learning*, pages 11170–11181. PMLR, 2021.

[Williams and Zipser 1989] Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

[Williams *et al.* 2017] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[Wolfram Research Inc. 2019] Wolfram Research Inc. *Mathematica, Version 13.2*, 2019. Champaign, IL, 2022.

[Wu *et al.* 2017] Chao-Yuan Wu, R Manmatha, Alexander J Smola, and Philipp Krahenbuhl. Sampling matters in deep embedding learning. In *Proceedings of the IEEE international conference on computer vision*, pages 2840–2848, 2017.

[Wu *et al.* 2020a] Chuhan Wu, Fangzhao Wu, Tao Qi, Xiaohui Cui, and Yongfeng Huang. Attentive pooling with learnable norms for text representation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2961–2970, 2020.

[Wu *et al.* 2020b] Qinzhuo Wu, Qi Zhang, Jinlan Fu, and Xuan-Jing 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)*, pages 7137–7146, 2020.

[Wu *et al.* 2020c] Yanfeng Wu, Chenkai Guo, Hongcan Gao, Xiaolei Hou, and Jing Xu.

Vector-based attentive pooling for text-independent speaker verification. In *Interspeech*, pages 936–940, 2020.

[Wu *et al.* 2021a] Junfeng Wu, Li Yao, Bin Liu, Zheyuan Ding, and Lei Zhang. Multi-task learning based encoder-decoder: A comprehensive detection and diagnosis system for multi-sensor data. *Advances in Mechanical Engineering*, 13(5):16878140211013138, 2021.

[Wu *et al.* 2021b] Qinzhuo Wu, Qi Zhang, and Zhongyu Wei. An edge-enhanced hierarchical graph-to-tree network for math word problem solving. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 1473–1482, 2021.

[Wu *et al.* 2021c] Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuan-Jing Huang. Math word problem solving with explicit numerical values. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5859–5869, 2021.

[Wu *et al.* 2021d] Qinzhuo Wu, Qi Zhang, Zhongyu Wei, and Xuanjing Huang. Math word problem solving with explicit numerical values. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 5859–5869, Online, August 2021. Association for Computational Linguistics.

[Wu *et al.* 2022] Chang-Yang Wu, Xin Lin, Zhen-Ya Huang, Yu Yin, Jia-Yu Liu, Qi Liu, and Gang Zhou. Clause-level relationship-aware math word problems solver. *Machine Intelligence Research*, 19(5):425–438, 2022.

[Xiao *et al.* 2018] Yawen Xiao, Jun Wu, Zongli Lin, and Xiaodong Zhao. A deep learning-based multi-model ensemble method for cancer prediction. *Computer methods and programs in biomedicine*, 153:1–9, 2018.

[Xie and Sun 2019] Zhipeng Xie and Shichao Sun. A goal-driven tree-structured neural model for math word problems. In *International Joint Conference on Artificial Intelligence*, 2019.

[Xiong *et al.* 2022a] Jing Xiong, Chengming Li, Min Yang, Xiping Hu, and Bin Hu. Expression syntax information bottleneck for math word problems. In *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 2166–2171, 2022.

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

[Xu *et al.* 2013] Chang Xu, Dacheng Tao, and Chao Xu. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634*, 2013.

[Xu *et al.* 2022] Jialiang Xu, Mengyu Zhou, Xinyi He, Shi Han, and Dongmei Zhang.

Towards robust numerical question answering: Diagnosing numerical capabilities of nlp systems. *arXiv preprint arXiv:2211.07455*, 2022.

[Xue *et al.* 2022] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022.

[Yang *et al.* 2016] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 1480–1489, 2016.

[Yang *et al.* 2020] Qiang Yang, Yu Zhang, Wenyuan Dai, and Sinno Jialin Pan. *Transfer learning*. Cambridge University Press, 2020.

[Yoo *et al.* 2020] Jin Yong Yoo, John X Morris, Eli Lifland, and Yanjun Qi. Searching for a search method: Benchmarking search algorithms for generating nlp adversarial examples. *arXiv preprint arXiv:2009.06368*, 2020.

[Yu *et al.* 2014] Dingjun Yu, Hanli Wang, Peiqiu Chen, and Zhihua Wei. Mixed pooling for convolutional neural networks. In *Rough Sets and Knowledge Technology: 9th International Conference, RSKT 2014, Shanghai, China, October 24-26, 2014, Proceedings 9*, pages 364–375. Springer, 2014.

[Yu *et al.* 2021] Weijiang Yu, Yingpeng Wen, Fudan Zheng, and Nong 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*, pages 3384–3394, 2021.

[Yuan *et al.* 2020] Li Yuan, Francis EH Tay, Guilin Li, Tao Wang, and Jiashi Feng. Revisiting knowledge distillation via label smoothing regularization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3903–3911, 2020.

[Yuhui *et al.* 2010] Ma Yuhui, Zhou Ying, Cui Guangzuo, Ren Yun, and Huang 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*, volume 2, pages 476–479. IEEE, 2010.

[Zafar *et al.* 2022] Afia Zafar, Muhammad Aamir, Nazri Mohd Nawi, Ali Arshad, Saman Riaz, Abdulrahman Alruban, Ashit Kumar Dutta, and Sultan Almotairi. A comparison of pooling methods for convolutional neural networks. *Applied Sciences*, 12(17):8643, 2022.

[Zell 1994] Andreas Zell. Simulation neuronaler netze. 1994.

[Zhang and Yang 2018] Yu Zhang and Qiang Yang. An overview of multi-task learning. *National Science Review*, 5(1):30–43, 2018.

[Zhang and Yang 2021] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021.

[Zhang *et al.* 2018] Ziqi Zhang, David Robinson, and Jonathan Tepper. Detecting hate speech on twitter using a convolution-gru based deep neural network. In *The Semantic Web: 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3–7, 2018, Proceedings 15*, pages 745–760. Springer, 2018.

[Zhang *et al.* 2020a] D. Zhang, Lei Wang, Nuo Xu, Bing Tian Dai, and Heng Tao Shen. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42:2287–2305, 2020.

[Zhang *et al.* 2020b] Jipeng Zhang, Roy Ka-Wei Lee, Ee-Peng Lim, Wei Qin, Lei Wang, Jie Shao, and Qianru Sun. Teacher-student networks with multiple decoders for solving math word problem. IJCAI, 2020.

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

[Zhang *et al.* 2022a] Wenqi Zhang, Yongliang Shen, Yanna Ma, Xiaoxia Cheng, Zeqi Tan, Qingpeng Nong, and Weiming Lu. Multi-view reasoning: Consistent contrastive learning for math word problem. *arXiv preprint arXiv:2210.11694*, 2022.

[Zhang *et al.* 2022b] Zhihan Zhang, Wenhao Yu, Mengxia Yu, Zhichun Guo, and Meng Jiang. A survey of multi-task learning in natural language processing: Regarding task relatedness and training methods. *arXiv preprint arXiv:2204.03508*, 2022.

[Zhao *et al.* 2017] Jing Zhao, Xijiong Xie, Xin Xu, and Shiliang Sun. Multi-view learning overview: Recent progress and new challenges. *Information Fusion*, 38:43–54, 2017.

[Zhao *et al.* 2018] Qi Zhao, Shuchang Lyu, Boxue Zhang, and Wenquan Feng. Multiactivation pooling method in convolutional neural networks for image recognition. *Wireless Communications and Mobile Computing*, 2018, 2018.

[Zhou and Zhou 2021] Zhi-Hua Zhou and Zhi-Hua Zhou. *Ensemble learning*. Springer, 2021.

[Zhou *et al.* 2021] Qiang Zhou, Zhong Qu, and Chong Cao. Mixed pooling and richer attention feature fusion for crack detection. *Pattern Recognition Letters*, 145:96–102, 2021.

[Zhuang *et al.* 2020] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2020.