Journal of Information & Optimization Sciences
ISSN 0252-2667 (Print), ISSN 2169-0103 (Online)
Vol. 45 (2024), No. 5, pp. 1359–1377
DOI : 10.47974/JIOS-1709

ExEqGen : An external symbolic solver for mathematical word problem solving

Michael Kurisappan *
S. Sundara Pandiyan †
Department of Computer Science and Engineering
CHRIST (Deemed to be University)
Bangalore 560074
Karnataka
India

Abstract

Mathematical word problem (MWP) solving is a complex domain that is characterized by a wide range of complex problems, ranging from rule-based systems to pre-trained language models. In this context, we introduce the External Equation Generator (ExEqGen), a novel model that combines SymPY, a sophisticated Python equation generator, with the state-of-the-art RoBERTa language model. Our approach leverages external symbolic solvers for expression generation and mathematical computations, demonstrating enhanced accuracy in solving math word problems. The study also demonstrates the positive impact of external symbols on improving numerical reasoning performance, particularly in mitigating false positives associated with longer reasoning programs. The findings provide valuable insights into enhancing the performance of Natural Language Understanding Models in complex tasks, contributing to the ongoing quest for more accurate, interpretable, and applicable natural language understanding models.

Subject Classification: *Primary 00A05, Secondary 97P40.*

Keywords: *Mathematical reasoning, Deep learning, Natural language processing, NLP datasets, Question answering, Semantic parsing, Large language models, RoBERTa, SymPY, FinQA, MathQA.*

1. Introduction

Scholars and students alike face a continuous difficulty when it comes to solving Mathematical Word Problems (MWPs). This is because MWPs

* *E-mail:* michael.k@res.christuniversity.in (Corresponding Author)
† *E-mail:* sundarapandiyan.s@christuniversity.in

demand a comprehensive comprehension of natural language phrasing and the skill to convert information into formal mathematical expressions. To demonstrate the issue, we will provide a straightforward illustration. In Section I, we examined a mathematical word problem that necessitates logical deduction and numerical calculations to construct an equation, which can then be utilized for problem resolution. Additionally, it is crucial to authenticate both the equation and the solutions in order to confirm the accuracy of the problem-solving process. We found that each step of the problem-solving process that involves several steps and is semantically related to the others makes the problem-solving process harder to complete. These problems draw attention to the process's practical issues.

Early attempts to solve MWP relied on rule-based systems, which, though effective for simple problems, struggled with complexity and ambiguity [1]. Pioneering work in the 1960s introduced rule-based architectures like the General Problem Solver (GPS), marking a step forward [2]. Subsequently, template-based techniques offered more adaptability, but required substantial manual effort [3]. Dolphin18K [4] dataset, for example, were primarily created to solve the MWP using a template-based approach [5-7].

In spite of the achievements of the Seq2Seq models, there is still ongoing research to enhance their performance. One potential approach is to utilize pre-trained language models [8][9], which offer the possibility of fine-tuning for specific tasks and show promise in improving performance.

In the field of Natural Language Processing (NLP), large language models (LLMs) are considered the most advanced and cutting-edge technological advancement. These models, also known as foundation models, have become an integral part of data work due to the increasing usage of generative AI tools in the modern era. LLMs possess extensive capabilities, not only in solving mathematical problems but also in handling multiple tasks simultaneously [10]. In this specific context, our study introduces the External Equation Generator (ExEqGen), a framework that combines SymPY [11], a sophisticated Python equation generator and parser, with RoBERTa [12], an advanced language model. Our approach leverages external symbolic solvers to generate expressions and conduct mathematical calculations, demonstrating enhanced accuracy in solving mathematical word problems.

Two versions of the ExEqGen model are used for training, one based on the RoBERTa base and the other on the RoBERTa large. These variants have different configuration parameters. The model architecture consists

of several steps, such as tokenization and positional embeddings, linear transformation, nonlinear activation, and symbolic expression parsing. We employ ExEqGen to solve math word problems and show its effectiveness in dealing with both simple and complex scenarios.

To assess the performance of our model, we use two datasets: FinQA [13], which is centered on financial analysis, and MathQA [14], a comprehensive compilation of mathematical word problems. By conducting a comparative analysis against state-of-the-art models such as FinQANet and ELASTIC, we can showcase the strengths of ExEqGen. Our implementation, which is constructed using PyTorch and Transformer libraries, achieves optimal performance by meticulously fine-tuning the hyperparameters.

The findings demonstrate the superiority of ExEqGen, specifically the RoBERTa-large variant, in terms of the precision of the program on both the FinQA and MathQA datasets. The research also explores the impact of external symbolic solvers, such as SymPY, showing their positive effect on numerical reasoning performance and their ability to reduce false positives in longer reasoning programs. The source code for the experiments is available on GitHub https://github.com/michaelvimal/ExEqGen

2. Background

Successful problem-solving in mathematics requires the ability to comprehend natural language phrasing, extract relevant information, and convert it into a formal mathematical representation. Table 1 illustrate the different stages involved in solving a math word problem. Initially, rule-based systems were employed to solve math word problems. These systems identified keywords and phrases in the problem text and applied predefined rules to translate them into mathematical expressions. Although effective for simple problems, these systems faced challenges when dealing with complex and ambiguous language. In the 1960s, Newell and Simon made significant advances in this field by introducing the General Problem Solver (GPS) [2], an architecture that decomposed problems into smaller steps and utilized symbolic manipulation techniques to find solutions.

The emergence of template-based techniques has provided a more adaptable substitute for rule-based systems. Instead of relying on rigid rules, templates offer a general framework for representing various types of problem. By incorporating identified keywords and entities, these

Table 1

Stages in Math word problem Solving

Math Word Problem: Emily and Alex are participating in a fundraising campaign for their school. Emily decides to donate $50 initially and then increase her donation by $10 every week. Alex starts with a donation of $30 and increases his donation by $15 every week. After a certain number of weeks, they realize that the total amount they've donated is the same. How many weeks did they participate in the campaign:
1.1 Logical Reasoning:
a) Define variables:
* w: number of weeks
* Etotal: Emily's total donation after w weeks
* Atotal: Alex's total donation after w weeks
b) Formulate equations:
* Etotal = 50 + 10w (start with initial amount + weekly increase)
* Atotal = 30 + 15w (similar for Alex)
c) Set the equations equal:
* Etotal = Atotal (Total amount becomes the same at some point)
1.2 Problem Solving:
* Substitute the above equations: 50 + 10w = 30 + 15w
* Combine like terms: 20 = 5w
* Divide both sides by 5: w = 4
1.3 Verification:
a) Check if the total amounts are equal after 4 weeks:
* Etotal = 50 + 10 · 4 = 90
* Atotal = 30 + 15 · 4 = 90
b) Both equations hold true, confirming the answer.

templates can handle a wider range of problems. However, the creation and maintenance of these templates still requires a considerable amount of manual effort. The field of solving mathematical word problems has been revolutionized by the introduction of machine learning. Statistical models such as Hidden Markov Models [5] and Conditional Random Fields [6] learn from large datasets of labelled problems, automatically identifying patterns and relationships within the text. These models have demonstrated significant improvements in accuracy and flexibility compared to rule-based and template-based approaches. This change

established the basis for stronger and more flexible systems capable of dealing with the inherent uncertainty and intricacies of human language. In recent years, deep learning has gained popularity, and Seq2Seq models [15][16][17] have taken the lead in this field. These models are trained to convert a sequence of words in a problem statement into a sequence of symbols in the corresponding mathematical expression. By employing an end- to-end approach, the model can grasp intricate connections directly from the data, eliminating the need for manually crafted features.

The introduction of deep learning in the 2010s marked a significant shift in the field of MWP solving. This period was characterized by the emergence of strong neural networks that had the ability to decipher intricate sentence structures, deduce implicit connections and generate solutions without relying on symbolic representations. These networks were trained on extensive datasets consisting of text and mathematical problems. Seq2Seq models equipped with attention mechanisms [7] demonstrated notable advances in directly producing numerical answers from natural language questions or converting them into symbolic expressions. Furthermore, the precision of these models was enhanced through the use of graph-based techniques such as Graph2Tree [18], which effectively captured the intricate relationships between concepts and entities in MWP.

Despite the impressive results achieved by the Seq2Seq models, researchers are continuously looking for ways to improve performance and address limitations. One promising avenue is the use of pre-trained language models (LLMs), which can be fine-tuned on large amounts of text data to enhance accuracy and generalization in tasks like solving math word problems [10] [19] [20]. Ongoing research is focused on developing more advanced models that not only solve problems, but also provide explanations and insights into the underlying mathematical concepts. An example of a hybrid approach is the combination of symbolic reasoning with deep learning, which can yield models that are more interpretable and robust.

3. Our Approach

We introduce a new model called External Equation Generator (ExEqGen), which combines SymPY [11], a sophisticated equation generator and parser module in Python, with the RoBERTa [12] model as its foundation. Our research demonstrates that incorporating an external symbolic solver for expression generation and mathematical computations

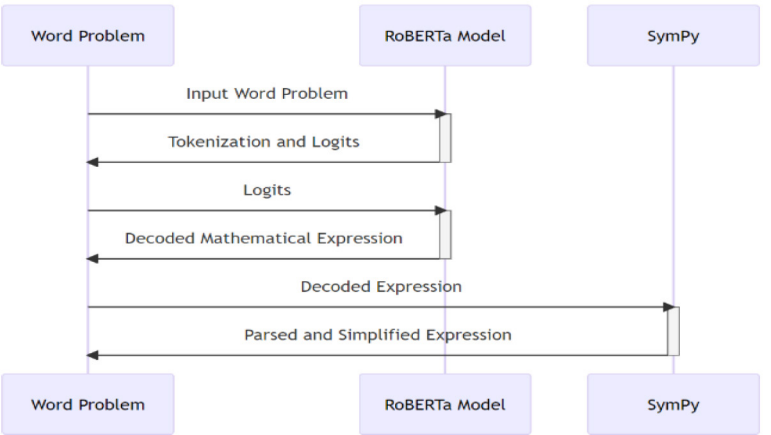

Figure 1

Architecture of ExEqGen Model

can significantly enhance the accuracy of solving math word problems. The general architecture of the ExEqGen model is illustrated in Figure 1.

In our study, we developed two versions of ExEqGen: one based on the RoBERTa base and the other on the RoBERTa-large. The primary configuration differences between these two variants lie in the number of encoders, self-attention heads, embedding dimensionality, hidden layers in the Feed Forward neural network, and total parameters used for pre-training the RoBERTa Model. The distinctive features of these configurations are summarized in Our approach initiates by combining token and positional embeddings (TE and PE) using RoBERTa. The embeddings are then subjected to a linear transformation, which involves matrix multiplication with weights (W) and addition of bias (b). Following this, a GELU (Gaussian Error Linear Unit) function, which is a crucial component of the non-linear activation step, is applied to learn patterns and relationships within the data. The resulting hidden states (H′) are utilized to compute logits through another linear transformation with weights (W′) and bias (b′). Softmax activation is then applied to the logits to obtain probabilities. The decoded output is generated from the logits, while simultaneously parsing the input expression using SymPy. The parsed expression is further simplified using SymPy's simplify function. The algorithm returns both the decoded output and the simplified expression. Algorithm 1 presents the various steps involved in executing the entire program. A comprehensive explanation of how the ExEqGen

model is employed to solve both simple and complex math word problems is provided below.

Algorithm 1: ExEqGen model (RoBERTa + SymPY)

Data: Token and Positional embeddings (TE and PE)
Result: Decoded output, Simplified expression

1. Combined embeddings : $\leftarrow TE + PE$;
2. Linear transformation : $H \leftarrow W \cdot (\text{Combined embeddings}) + \text{b}$;
3. Non-linear activation : $H' \leftarrow \text{GELU}(H)$;
4. Logits: $\leftarrow W' \cdot H' + b'$;
5. Softmax activation: Probabilities $\leftarrow$ softmax (Logits);
6. Decoded output : $\leftarrow$ decode (Logits)
7. Parsed expression : $\leftarrow$ SymPy.parse_expression (expression)
8. Simplified expression : $\leftarrow$ SymPy.simplify (parsed expression)

Return Decoded output, Simplified expression

3.1 *Embeddings*

In natural language processing (NLP), embedding [8] involves representing words or tokens as vectors in a continuous vector space.

3.1.1 Token Embeddings

Let's denote the input sequence as a set of token IDs: Token IDs = [id_1, id_2, ..., id_n], where each id_i corresponds to a token in the vocabulary. The embedding matrix (E) is used to systematically arrange the vectors generated. E[id_i] is the embedding for a token id_i. Let Token Embeddings[i] represent the Token embedding (TE):

$$\boldsymbol{TE} = [E[id_1], E[id_2], \ldots, E[id_n]] \tag{1}$$

3.1.2 Positional Embeddings

In order to enable the model to distinguish between tokens based on their relative positions in a sequence, positional embeddings [8] are incorporated into token embeddings by adding elements. By adding positional embeddings elementwise, the original information about the token itself is retained. Each dimension of the resulting vector encompasses both the positional and token-specific information, as they are combined

through element-wise addition. The model can adjust the weights during training to assign more importance to specific dimensions, depending on the task at hand. This flexibility is crucial for identifying intricate patterns in sequential data. The combination of token embeddings and positional embeddings generates a more comprehensive vector representation that assists the model in understanding not only the meaning of each token, but also its position within the sequence. This is particularly significant for language modelling tasks, where the arrangement of words largely determines meaning.

Let Position Embeddings[i] represent the positional embedding (PE) for the token at position 'i' in the sequence.

$$\boldsymbol{PE} = [PE[1], PE[2], \ldots, PE[n]] \tag{2}$$

3.2 *Linear Transformation*

The combined embeddings are linearly transformed [7] using a weight matrix W and a bias vector b. Let H represent the hidden states after the linear transformation.

$$\boldsymbol{H} = LayerNorm(TE + PE + SE) \tag{3}$$

$$\boldsymbol{H} = LayerNorm(E[id_1] + PE[1] + SE[1], E[id_2] + PE[2] + SE[2], \ldots, E[id_n] + PE[n] + SE[n]) \tag{4}$$

3.3 *Non-linear Activation*

The inclusion of a non-linear activation function, such as GELU (Gaussian Error Linear Unit), in the processing of hidden states (H) introduces non-linearity to the model [12], improving its ability to learn intricate patterns and relationships within the data. The expressiveness gained through nonlinear activation functions is crucial for modelling complex, non-linear relationships inherent in natural language. GELU, specifically chosen for its performance in certain contexts, helps transform the feature space effectively, contributing to the learning of meaningful representations in deep neural networks. By avoiding the vanishing gradient problem, non-linear activation functions facilitate more effective gradient propagation during backpropagation, addressing challenges associated with training deep networks.

$$\boldsymbol{H}' = GELU(H) \tag{5}$$

Within the RoBERTa model [12], these particular actions form part of a single layer, while the overall model is typically composed of multiple layers stacked on top of each other. Through-out the training process, the weights (W) and bias (b) are determined. The values obtained from the non-linear activation H′ are multiplied by the weights (W). The legits are then generated by adding the resulting value to the bias (b).

3.4 *Logits*

Logits refer to the unprocessed predictions produced by a machine learning model prior to the application of an activation function. These predictions are not normalized and serve as an indication of the model's certainty for each potential class. Logits [21] are instrumental in determining the final output of the model and serve as an important intermediate step in the decision-making process of machine learning algorithms.

In our approach the RoBERTa model produces logits Z for each class.

$$\boldsymbol{Z} = [z_1, z_2, \ldots, z_k] \tag{6}$$

3.5 *Softmax Activation*

The softmax activation function is applied to each logit z_i to obtain the probability P_i.

$$\boldsymbol{P_i} = e_i^z \,/\, \Sigma_{(j=1)}^{(k)} e_j^z \tag{7}$$

The softmax operation [22][8] is applied independently to each element in Z, resulting in the probability distribution P:

$$\boldsymbol{P} = [e_1^z \,/\, \Sigma_{(j=1)}^{(k)} e_j^z, e_2^z \,/\, \Sigma_{(j=1)}^{(k)} e_j^z, \ldots, e_k^z \,/\, \Sigma_{(j=1)}^{(k)} e_j^z] \tag{8}$$

3.6 *Logits Decoding*

The predicted class is often determined by taking the argmax of the probabilities [23]:

$$\boldsymbol{P'} = argmax\,(P) \tag{9}$$

This means selecting the class with the highest probability as the predicted class P′.

3.7 *Parse the Expression*

During our experiment, after determining the predicted class index, we decode the class value by mapping the index to its corresponding class label. This mapping is usually established during the training phase, where each class is given a unique label [24]. By associating the predicted index with the predefined labels, we can easily interpret the model output in a human-readable format. The decoded expression, represented as a string, is denoted as exp = P′.

$$\textbf{\textit{Parsed Expression}} = \textit{SymPy.expression}(\textit{exp}) \tag{10}$$

3.8 *Simplify the Expression*

In this phase of our investigation, we employ the "Simplify" function in SymPy [11], which is a robust tool utilized to parse a string expression and convert it into a SymPy expression. The primary purpose of the "Simplify" function is to take a string that represents a mathematical expression and convert it into a SymPy expression, allowing symbolic computation. Once the input string is parsed using "Simplify", it becomes feasible to carry out various operations and manipulations on the symbolic expression. This procedure proves to be highly advantageous in the mathematical and scientific domains where symbolic representations are critical for analysis [25][26].

$$\textbf{\textit{Simplified Expression}} = \textit{simplify}(\textit{Parsed expression}) \tag{11}$$

To ensure the accuracy of the conversion, we compare the original input string with the final output obtained after parsing and manipulation. This verification step guarantees that the "Simplify" function correctly captures the intended mathematical expression. It enhances the dependability of SymPy's symbolic computations, making it a valuable resource for users dealing with symbolic mathematical expressions.

4. Experimental Setup

4.1 *Dataset*

We perform evaluation experiments on two datasets, namely FinQA [13] and MathQA [14]. The FinQA and MathQA datasets have distinct advantages in conducting experiments on numerical reasoning. FinQA focuses on real-world finance, necessitating intricate reasoning using financial data and domain knowledge. Its transparent reasoning programs

provide insight into how models solve problems, highlighting the challenges that existing models often struggle with. On the other hand, MathQA builds on the popular AQuA dataset [27] and includes formally specified operations to facilitate interpretable analysis of model reasoning. The diverse range of problems in MathQA enhances the generalizability of models and serves as a benchmark for interpretable models.

4.1.1 FinQA

The field of financial analysis is revolutionized by the FinQA dataset. It tackles the challenge of automating insights from vast amounts of financial documents by offering a unique resource of question- answering pairs authored by financial experts. These pairs not only provide factual answers but also involve complex numerical reasoning that requires multiple calculations and an understanding of financial concepts. To evaluate the dataset, we employ the program accuracy matrix from the original paper. The program accuracy measures the correctness of the generated expressions, which include operators and logic, by comparing them to the original expressions. FinQA grants access to over 8,000 QA pairs and their corresponding "gold" reasoning programs, which are divided into sections containing 6,251, 883, and 1,147 examples for the training, evaluation, and testing phases, respectively.

4.1.2 MathQA

The MathQA dataset consists of a large collection of over 37,200 math word problems and their corresponding operation programs. This dataset was created to advance the field of interpretable neural math problem solving. Unlike previous datasets [28][29][15], MathQA provides a comprehensive set of problems with detailed operation sequences, allowing the development and evaluation of models that can solve problems and provide explanations for their reasoning. The dataset covers a wide range of topics including arithmetic, algebra, geometry, and word problems of varying difficulty levels. Each problem in the data set is accompanied by a hand-written operation program that outlines the mathematical procedures necessary to solve it. Additionally, clear explanations are provided for the solutions, providing valuable insights into the model's methodology. Due to its large size, diverse problem set, and detailed operation annotations, MathQA serves as a challenging and useful benchmark for developing models that can understand mathematical problems and generate human-readable solutions. The accuracy of the

operating programs serves as the performance evaluation metric for the dataset, similar to FinQA.

4.2 *Baselines*

In this paper, we conduct a comparative analysis of our ExEqGen model with two state-of-the-art models, namely FinQANet [13] and ELASTIC [30]. FinQANet uses an encoder-decoder architecture with a cache update mechanism to generate the program. However, it is limited to producing operators with only two operands. To train and evaluate FinQANet, we utilized the MathQA data set and removed operators involving more than two operands. On the other hand, ELASTIC utilizes RoBERTa as both an encoder and a compiler to generate symbolic expressions. The numerical reasoning process in ELASTIC involves two phases. In the first phase, the encoder transforms the question and problem

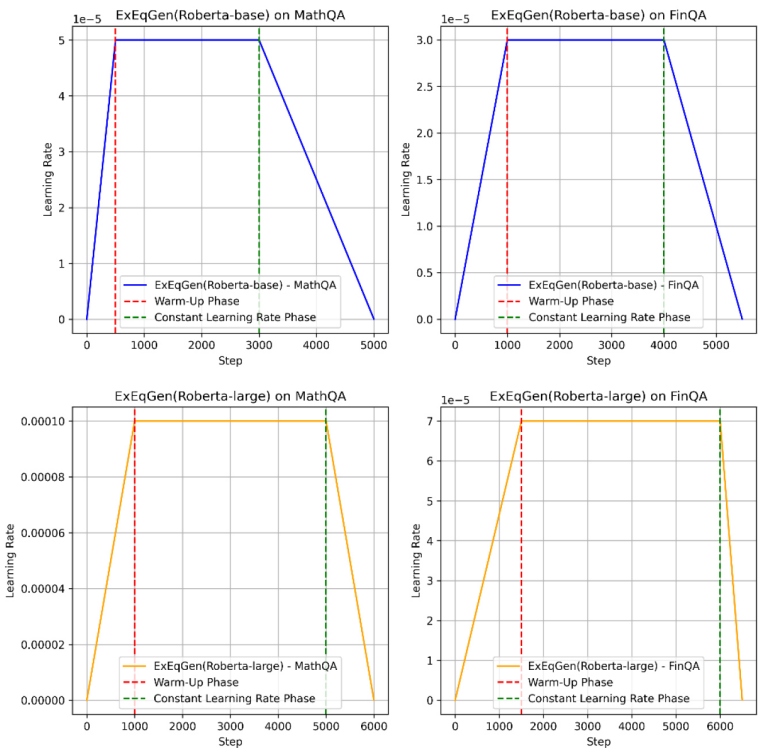

Figure 2

ExEqGen (RoBERTa-Base) and (RoBERTa-Large) learning rates: FinQA vs MathQA datasets

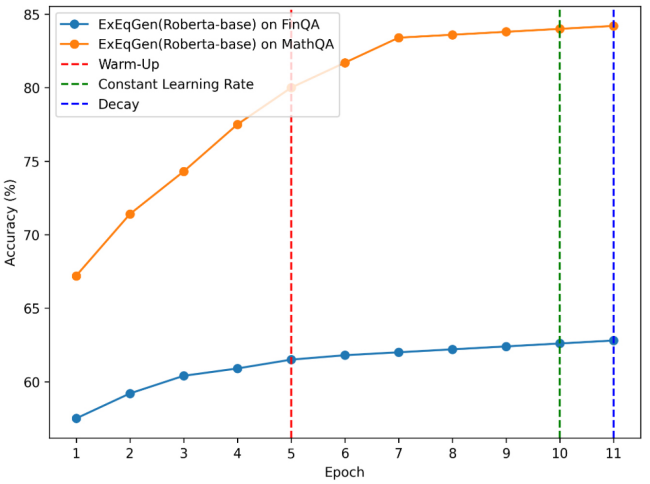

Figure 3

ExEqGen (RoBERTa-Base : Accuracy vs Epoch)

text into easily understandable information. Then, the compiler, which consists of four modules, is deployed to construct the solution. The Reasoning Manager oversees the process and guides the Operator and Operand Generators in creating the necessary mathematical operations and values. The Memory Register serves as temporary storage for intermediate results, facilitating the construction of complex programs. It is worth noting that ELASTIC's flexibility allows it to generate subprograms in any order, and its efficient cache system optimizes the problem-solving process by reusing previous results.

4.3 *Implementation*

Our ExEqGen model was implemented using the PyTorch and Transformer libraries. The model was trained on a Google Cloud server with an NVIDIA H100 80GB GPU. The training process involved fine-tuning a pre-trained RoBERTa model. The duration of training varied depending on the model size (RoBERTa-base or -large) and the task (FinQA or MathQA), typically ranging from 12 to 16 epochs. The learning rate models were used to compute the warm-up, decay, and constant-rate phases independently. The learning rate versus the steps involved in each iteration of the program execution is visualized in Figure 2. For MathQA, the ExEqGen-base model used fewer steps and a lower peak rate compared to FinQA, while the ExEqGen-large model used more steps and a higher

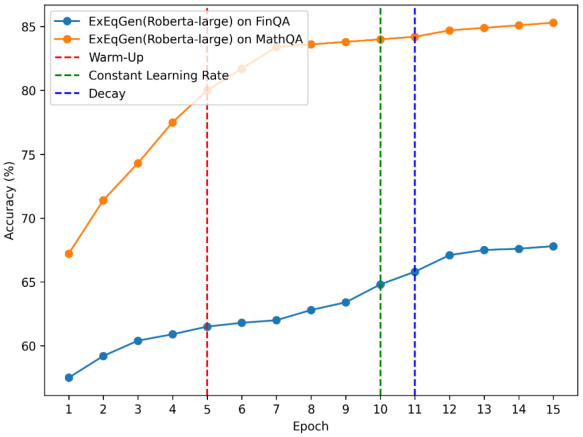

Figure 4

ExEqGen (RoBERTa-Large : Accuracy vs Epoch)

peak rate for MathQA. These variations emphasize the importance of fine-tuning hyperparameters for specific models and tasks. The precision of our model during the training phase is depicted in Figure 3 and Figure 4, showing the accuracy plotted against the number of epochs for RoBERTa-Base and RoBERTa-Large, respectively.

During each training iteration, we processed 10 batches of data in our experiment. The model's parameters were continuously adjusted by the Adam optimizer, which initially had a learning rate that gradually decreased over the training process. To prevent overfitting, we also used dropout (at a rate of 0.1) and weight decay (at a rate of 1e-5) in the training regime. These techniques ensured that the model learned general patterns instead of just memorizing the training data. Overall, we carefully balanced various hyperparameters to achieve the best performance for specific tasks and model configurations.

5. Results

In this section, we present the main findings of our study, focusing on the performance of the ExEqGen model and its variations on the FinQA and MathQA datasets. The performance metrics of our ExEqGen model and different baselines in both data sets are shown in Table 2.

It is worth noting that the ExEqGen (RoBERTa-large) model outperforms all other models, achieving the highest scores on both

Table 2
Stages in Math word problem Solving

Datasets & Metrics	FinQA Prog Accuracy	MathQA Prog Accuracy
FinQANet (RoBERTa-base)	58.38	74.12
FinQANet (RoBERTa-large)	63.52	79.2
ELASTIC (RoBERTa-base)	59.28	82.27
ELASTIC (RoBERTa-large)	65.21	83
RoBERTa-base + SymPy	62.75	84.19
RoBERTa-large + SymPy	68.18	85.29

datasets. We also observe that the choice between RoBERTa-large and RoBERTa-base has an impact on the model's performance. Although RoBERTa-large excels in capturing linguistic details, it requires significant computational resources. On the other hand, RoBERTa-base has fewer parameters, resulting in improved speed and accessibility.

5.1 *Dataset Analysis*

In the FinQA dataset, ExEqGen (RoBERTa-large) performs better than ELASTIC (RoBERTa- large), with a 2.97 increase in program accuracy. However, when transitioning to ExEqGen (RoBERTa-base), there is a decrease in performance by 2.46 points, indicating that ELASTIC (RoBERTa-large) still outperforms ExEqGen (RoBERTa-base). ExEqGen (RoBERTa-large) emerges as the best performing model on the MathQA dataset, surpassing ELASTIC (RoBERTa-large) by 2.29 points in program accuracy and FinQANet (RoBERTa-large) by 6.09 points. ExEqGen (RoBERTa-large) and ExEqGen (RoBERTa-base) show a marginal difference in performance, indicating that the contextual semantic information extracted from the passage and the question is sufficient for efficient performance.

5.2 *Influence of SymPY and Program Steps*

To showcase the strength of ExEqGen, we investigate the importance of the external symbolic solver SymPY. The study demonstrates the positive impact of SymPY on improving numerical reasoning performance, particularly in mitigating false positives associated with longer reasoning programs. ExEqGen exhibits superior performance in generating longer numerical reasoning programs, showing resilience to cascading errors. The program steps up to four produce the best results, while programs

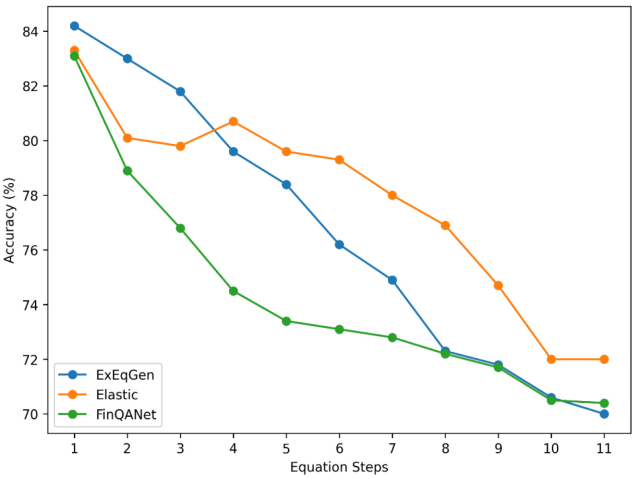

Figure 5

Accuracy vs Equation steps between ExEqGen, ELASTIC and FinQANet Models

beyond five show a notable reduction in improvement. The influence of reasoning steps versus performance on our research against the three models is shown in Figure 5. This underscores the advantage of separating the expression generation procedure for math operations, revealing that incorrect predictions stem from the model's selection of incorrect operands, potentially influenced by contextual noise. Finally, the results indicate that ExEqGen, especially with RoBERTa-large, excels at tackling numerical reasoning tasks, showcasing the importance of external symbolic solvers, and demonstrating proficiency in generating longer program steps. The findings provide valuable insights into enhancing the performance of natural language understanding models in complex tasks.

6. Future Research Directions

In this research, we delved into the intricate domain of mathematical word problem (MWP) solving, tracing the evolution from rule-based systems to recent advancements in deep learning and pre-trained language models. The culmination of our exploration is the introduction of the External Equation Generator (ExEqGen), a novel model seamlessly integrating SymPY, a sophisticated Python equation generator, with the state-of-the-art RoBERTa language model. ExEqGen represents a significant leap forward in addressing the challenges inherent in MWPs,

demonstrating enhanced numerical reasoning performance, especially in scenarios involving longer reasoning programs. Through rigorous comparative analysis with contemporary models, ExEqGen consistently outperformed its counterparts, showcasing superior program accuracy, particularly with the RoBERTa-large variant. Our research has also shed light on the nuanced interplay between model architecture and performance. Looking ahead, the success of ExEqGen opens avenues for further exploration and enhancement.

7. Discussion and conclusion

Future research endeavors may include investigating and optimizing fine-tuning strategies for other LLMs, evaluating the model generalizability to additional datasets and domains, exploring hybrid models by integrating external solvers, investigating scalability considerations for handling larger datasets and more complex problems, and exploring user interface integration for seamless interaction. In conclusion, ExEqGen represents a significant advancement in MWP solving, and the proposed future work areas aim to address existing challenges, extend the model's capabilities, and ensure its practical utility across a wide range of applications, contributing to the ongoing quest for more accurate, interpretable, and applicable natural language understanding models.

References

[1] Zhang, Dongxiang, et al. "The gap of semantic parsing: A survey on automatic math word problem solvers." IEEE transactions on pattern analysis and machine intelligence 42.9 : 2287-2305 (2019).

[2] Newell, Allen, and Herbert Alexander Simon. "GPS, a program that simulates human thought." (1961).

[3] Kushman, Nate, et al. "Learning to automatically solve algebra word problems." Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (2014).

[4] Huang, Danqing, et al. "How well do computers solve math word problems? large-scale dataset construction and evaluation." Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (2016).

[5] Baum, Leonard E., et al. "A maximization technique occurring in the statistical analysis of probabilistic functions of Markov chains." *The Annals of Mathematical Statistics* 41.1 : 164-171 (1970).

[6] Lafferty, John, Andrew McCallum, and Fernando Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." Icml. Vol. 1. No. 2. (2001).

[7] Vaswani, Ashish, et al. "Attention is all you need." *Advances in Neural Information Processing Systems* 30 (2017).

[8] Devlin, Jacob, et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." arXiv preprint arXiv:1810.04805 (2018).

[9] Liang, Zhenwen, et al. "Mwp-bert: Numeracy-augmented pretraining for math word problem solving." arXiv preprint arXiv:2107.13435 (2021).

[10] Bommasani, Rishi, et al. "On the opportunities and risks of foundation models." arXiv preprint arXiv:2108.07258 (2021).

[11] Meurer, Aaron, et al. "SymPy: symbolic computing in Python." PeerJ Computer Science 3 : e103 (2017).

[12] Liu, Yinhan, et al. "Roberta: A robustly optimized bert pretraining approach." arXiv preprint arXiv:1907.11692 (2019).

[13] Chen, Zhiyu, et al. "Finqa: A dataset of numerical reasoning over financial data." arXiv preprint arXiv:2109.00122 (2021).

[14] Amini, Aida, et al. "Mathqa: Towards interpretable math word problem solving with operation-based formalisms." arXiv preprint arXiv:1905.13319 (2019).

[15] Wang, Yan, Xiaojiang Liu, and Shuming Shi. "Deep neural solver for math word problems." Proceedings of the 2017 conference on empirical methods in natural language processing (2017).

[16] Zhang, Biao, et al. "Variational neural machine translation." arXiv preprint arXiv:1605.07869 (2016).

[17] Wang, Lei, et al. "Mathdqn: Solving arithmetic word problems via deep reinforcement learning." Proceedings of the AAAI conference on artificial intelligence. Vol. 32. No. 1 (2018).

[18] Zhang, Jipeng, et al. "Graph-to-tree learning for solving math word problems." Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (2020).

[19] OpenAI, R. "Gpt-4 technical report. arxiv 2303.08774." View in Article 2.5 (2023).

[20] Anil, Rohan, et al. "Palm 2 technical report." arXiv preprint arXiv:2305.10403 (2023).

[21] Kumar, Ankit, et al. "An improved quantum key distribution protocol for verification." *Journal of Discrete Mathematical Sciences and Cryptography* 22.4 : 491-498 (2019).

[22] Saini, Garima, et al. "Structural equation based model to investigate the moderating effect of fear of COVID using partial least square method." *Journal of Interdisciplinary Mathematics* 25.3 : 703-720 (2022).

[23] Kumar, Ankit, et al. "An enhanced quantum key distribution protocol for security authentication." *Journal of Discrete Mathematical Sciences and Cryptography* 22.4 : 499-507 (2019).

[24] Kipf, Thomas N., and Max Welling. "Semi-supervised classification with graph convolutional networks." arXiv preprint arXiv:1609.02907 (2016).

[25] Bhatnagar, Vaibhav, et al. "Descriptive analysis of COVID-19 patients in the context of India." *Journal of Interdisciplinary Mathematics* 24.3 : 489-504 (2021).

[26] Huot, Mathieu, and Amir Shaikhha. "Denotationally correct, purely functional, efficient reverse-mode automatic differentiation." arXiv preprint arXiv:2212.09801 (2022).

[27] Garcia, Noa, et al. "A dataset and baselines for visual question answering on art." Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16. Springer International Publishing (2020).

[28] Upadhyay, Shyam, and Ming-Wei Chang. "Annotating derivations: A new evaluation strategy and dataset for algebra word problems." arXiv preprint arXiv:1609.07197 (2016).

[29] Koncel-Kedziorski, Rik, et al. "MAWPS: A math word problem repository." Proceedings of the 2016 conference of the north american chapter of the association for computational linguistics: human language technologies (2016).

[30] Zhang, Jiaxin, and Yashar Moshfeghi. "ELASTIC: numerical reasoning with adaptive symbolic compiler." *Advances in Neural Information Processing Systems* 35 : 12647-12661 (2022).

Received April, 2024