Point to the Expression: Solving Algebraic Word Problems using the Expression-Pointer Transformer Model

Bugeun Kim **Kyung Seo Ki** **Donggeon Lee** **Gahgene Gweon**
Department of Transdisciplinary Studies,
Seoul National University,
Seoul, Republic of Korea
{cd4209,kskee88,lsw5835,ggweon}@snu.ac.kr

Abstract

Solving algebraic word problems has recently emerged as an important natural language processing task. To solve algebraic word problems, recent studies suggested neural models that generate solution equations by using 'Op (operator/operand)' tokens as a unit of input/output. However, such a neural model suffered two issues: expression fragmentation and operand-context separation. To address each of these two issues, we propose a pure neural model, Expression-Pointer Transformer (EPT), which uses (1) 'Expression' token and (2) operand-context pointers when generating solution equations. The performance of the EPT model is tested on three datasets: ALG514, DRAW-1K, and MAWPS. Compared to the state-of-the-art (SoTA) models, the EPT model achieved a comparable performance accuracy in each of the three datasets; 81.3% on ALG514, 59.5% on DRAW-1K, and 84.5% on MAWPS. The contribution of this paper is two-fold; (1) We propose a pure neural model, EPT, which can address the expression fragmentation and the operand-context separation. (2) The fully automatic EPT model, which does not use hand-crafted features, yields comparable performance to existing models using hand-crafted features, and achieves better performance than existing pure neural models by at most 40%.

1 Introduction

Solving algebraic word problems has recently become an important research task in that automatically generating solution equations requires understanding natural language. Table 1 shows a sample algebraic word problem, along with corresponding solution equations that are used to generate answers for the problem. To solve such problems with deep learning technology, researchers recently suggested neural models that generate solution equations automatically (Huang et al., 2018; Amini et al., 2019; Chiang and Chen, 2019; Wang et al., 2019). However, suggested neural models showed a fairly large performance gap compared to existing state-of-the-art models based on hand-crafted features in popular algebraic word problem datasets, such as ALG514 (44.5% for pure neural model vs. 83.0% for using hand-crafted features) (Huang et al., 2018; Upadhyay and Chang, 2016). To address the large performance gap in this study, we propose a larger unit of input/output (I/O) token called "*Expressions*" for a pure neural model. Figure 1 illustrates conventionally used "Op (operator/operands)" versus our newly proposed "Expression" token.

Problem	**One** number is **eight** more than **twice** another and their sum is **20**. What are their numbers?
Numbers	1('one'), 8('eight'), 2('twice'), 20.
Equations	$x_0 - 2x_1 = 8, x_0 + x_1 = 20$
Answers	$(16, 4)$

Table 1: A sample algebraic word problem

To improve the performance of pure neural models that can solve algebraic word problems, we identified two issues that can be addressed using Expression tokens, which are shown in Figure 1: (1) *expression fragmentation* and (2) *operand-context separation*. First, the expression fragmentation issue is a segmentation of an expression tree, which represents a computational structure of equations that are used to generate a solution. This issue arises when Op, rather than the whole expression tree, is used as an input/output unit of a problem-solving model. For example, as shown in Figure 1 (a), using Op tokens as an input to a problem-solving model disassembles a tree structure into operators ("×") and operands ("x_1" and "2"). Meanwhile, we propose using the

Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, pages 3768–3779,
November 16–20, 2020.

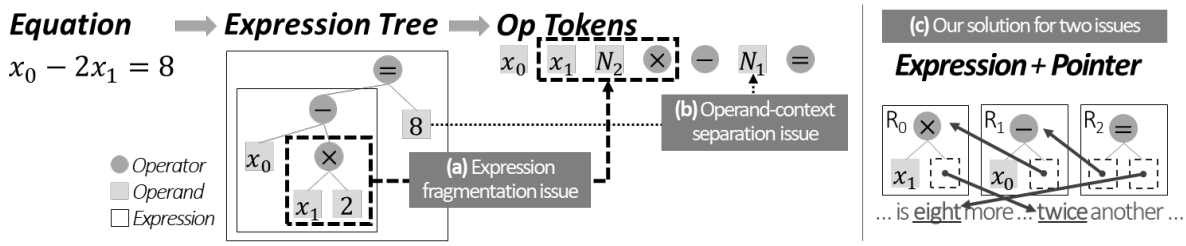

Figure 1: Illustration using the word problem in Table 1 for the (a) expression fragmentation issue, (b) operand-context separation issue, and (c) our solution for these two issues.

"Expression" ($\times(x_1, 2)$) token, which can explicitly capture a tree structure as a whole, as shown in Figure 1 (c).

The second issue of operand-context separation is the disconnection between an operand and a number that is associated with the operand. This issue arises when a problem-solving model substitutes a number stated in an algebraic word problem into an abstract symbol for generalization. As shown in Figure 1 (b), when using an Op token, the number 8 is changed into an abstract symbol 'N_1'. Meanwhile, when using an Expression token, the number 8 is not transformed into a symbol. Rather a pointer is made to the location where the number 8 occurred in an algebraic word problem. Therefore, using such an "operand-context pointer" enables a model to access contextual information about the number directly, as shown in Figure 1 (c); thus, the operand-context separation issue can be addressed.

In this paper, we propose a pure neural model called Expression-Pointer Transformer (EPT) to address the two issues above. The contribution of this paper is two-fold;

1. We propose a pure neural model, Expression-Pointer Transformer (EPT), which can address the expression fragmentation and operand-context separation issues.

2. The EPT model is the first pure neural model that showed comparable accuracy to the existing state-of-the-art models, which used hand-crafted features. Compared to the state-of-the-art pure neural models, the EPT achieves better performance by about 40%.

In the rest of the paper, we introduce existing approaches to solve algebraic word problems in Section 2. Next, Section 3 introduces our proposed model, EPT, and Section 4 reports the experimental settings. Then in Section 5, results of two studies are presented. Section 5.1 presents a performance comparison between EPT and existing SoTA models. Section 5.2 presents an ablation study examining the effects of Expression tokens and applying operand-context pointers. Finally, in Section 6, a conclusion is presented with possible future directions for our work.

2 Related work

Our goal is to design a pure neural model that generates equations using 'Expression' tokens to solve algebraic word problems. Early attempts for solving algebraic word problems noted the importance of Expressions in building models with hand-crafted features (Kushman et al., 2014; Roy et al., 2015; Roy and Roth, 2015; Zhou et al., 2015; Upadhyay et al., 2016). However, recent neural models have only utilized 'Op (operator/operand)' tokens (Wang et al., 2017; Amini et al., 2019; Chiang and Chen, 2019; Huang et al., 2018; Wang et al., 2019), resulting in two issues: (1) the expression fragmentation issue and (2) the operand-context separation issue. In the remaining section, we present existing methods for tackling each of these two issues.

To address the expression fragmentation issue, researchers tried to reflect relational information between operators and operands either by using a two-step procedure or a single step with sequence-to-sequence models. Earlier attempts predicted operators and their operands by using a two-step procedure. Such early models selected operators first by classifying a predefined template (Kushman et al., 2014; Zhou et al., 2015; Upadhyay et al., 2016), then in the second step, operands were applied to the template selected in the first step. Other models selected operands first before constructing expression trees with operators in the second step (Roy et al., 2015; Roy and Roth, 2015). However, such two-step procedures in these early attempts

Input position	Output index	Expression token			Meaning
		Operator (f_i)	Operand 0 (a_{i0})	Operand 1 (a_{i1})	
0	-	BEGIN			(Start an equation)
1	R_0	VAR			(Generate variable x_0)
2	R_1	VAR			(Generate variable x_1)
3	R_2	$\times$	2	R_1	$2x_1$
4	R_3	$-$	R_0	R_2	$x_0 - 2x_1$
5	R_4	$=$	R_3	8	$x_0 - 2x_1 = 8$
6	R_5	$+$	R_0	R_1	$x_0 + x_1$
7	R_6	$=$	R_5	20	$x_0 + x_1 = 20$
-	R_7	END			(Gather all equations)

Table 2: The Expression token sequence for $x_0 - 2x_1 = 8$ and $x_0 + x_1 = 20$

can be performed via a single-step procedure with neural models. Specifically, recent attempts have utilized sequence-to-sequence (seq2seq) models as a single-step procedure to learn the implicit relationship between operators and operands (Amini et al., 2019; Chiang and Chen, 2019; Wang et al., 2019). For example, to capture the operator-operand relationship, Chiang and Chen (2019) constructed a seq2seq model that used push/pop actions on a stack for generating operator/operand tokens. Similarly, Amini et al. (2019) built a seq2seq model to generate an operator token right after producing required operand tokens. However, although these seq2seq approaches consider relational information of operands when generating operators, the approach still does not address the problem of lacking relation information of operators when generating operands. On the other hand, by using Expression token, our model can consider relational information when generating both operator and operands.

Secondly, there were efforts to address the operand-context separation issue. To utilize contextual information of an operand token, researchers built hand-crafted features that capture the semantic content of a word, such as the unit of a given number (Roy and Roth, 2015; Koncel-Kedziorski et al., 2015; Zhou et al., 2015; Upadhyay et al., 2016; Roy and Roth, 2017) or dependency relationship between numbers (Kushman et al., 2014; Zhou et al., 2015; Upadhyay et al., 2016). However, devising hand-crafted input features was time-consuming and required domain expertise. Therefore, recent approaches have employed distributed representations and neural models to learn numeric context of operands automatically (Wang et al., 2017; Huang et al., 2018; Chiang and Chen, 2019; Amini et al., 2019). For example, Huang et al. (2018) used a pointer-generator network that can point to the context of a number in a given math problem. Although Huang's model can address the operand-context separation issue using pointers, their pure neural model did not yield a comparable performance to the state-of-the-art model using hand-crafted features (44.5% vs. 83.0%). In this paper, we propose that by including additional pointers that utilize the contextual information of operands and neighboring Expression tokens, performance of pure neural models can improve.

3 EPT: Expression-Pointer Transformer

Figure 2 shows the proposed Expression-Pointer Transformer (EPT)[1] model, which adopts the encoder-decoder architecture of a Transformer model (Vaswani et al., 2017). The EPT utilizes the ALBERT model (Lan et al., 2019), a pretrained language model, as the encoder. The encoder input is tokenized words of the given word problem, and encoder output is the encoder's hidden-state vectors that denote numeric contexts of the given problem.

After obtaining the encoder's hidden-state vectors from the ALBERT encoder, the transformer decoder generates 'Expression' tokens. The two decoder inputs are Expression tokens and the ALBERT encoder's hidden-state vectors, which are used as memories. For the given example problem, the input is a list of 8 Expression tokens shown in Table 2. We included three special commands in the list: VAR (generate a variable), BEGIN (start an equation), and END (gather all equations). Following the order specified in the list of Table 2, the EPT receives one input Expression

[1]The code is available on https://github.com/snucclab/ept.

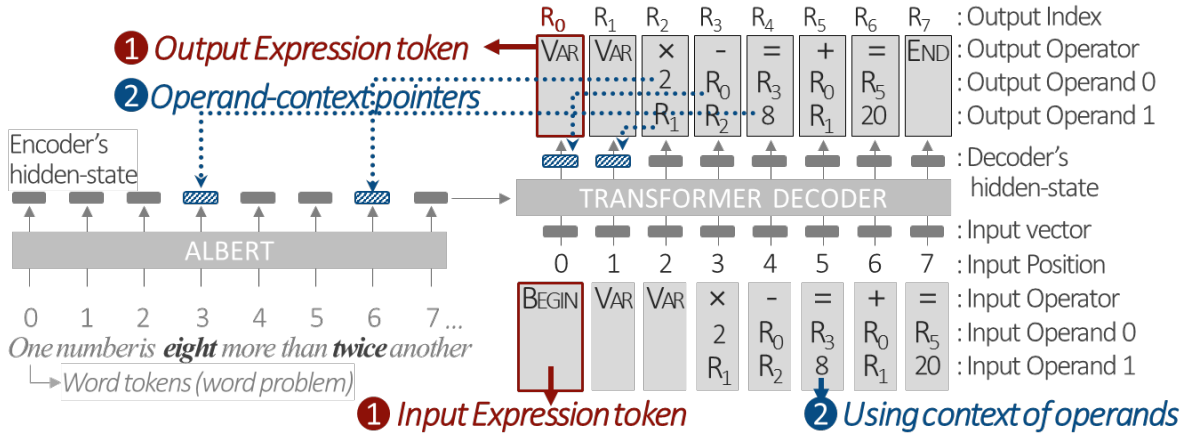

Figure 2: The architecture of Expression-Pointer Transformer (EPT) where two ideas applied: (1) Expression token and (2) operand-context pointer.

at a time. For the ith Expression input, the model computes an input vector $\mathbf{v}_i$. The EPT's decoder then transforms this input vector to a decoder's hidden-state vector $\mathbf{d}_i$. Finally, the EPT predicts the next Expression token by generating the next operator and operands simultaneously.

To produce 'Expression' tokens, two components are modified from the vanilla Transformer: input vector and output layer. In the following subsections, we explain the two components.

3.1 Input vector of EPT's decoder

The input vector $\mathbf{v}_i$ of ith Expression token is obtained by combining operator embedding $\mathbf{f}_i$ and operand embedding $\mathbf{a}_{ij}$ as follows:

$$\mathbf{v}_i = \mathrm{FF}_{\mathrm{in}}\left(\mathrm{Concat}\left(\mathbf{f}_i, \mathbf{a}_{i1}, \mathbf{a}_{i2}, \cdots, \mathbf{a}_{ip}\right)\right), \quad (1)$$

where FF_* indicates a feed-forward linear layer, and $\mathrm{Concat}(\cdot)$ means concatenation of all vectors inside the parentheses. All the vectors, including $\mathbf{v}_i$, $\mathbf{f}_i$, and $\mathbf{a}_{ij}$, have the same dimension D. Formulae for computing the two types of embedding vectors, $\mathbf{f}_i$ and $\mathbf{a}_{ij}$ are stated in the next paragraph.

For the operator token f_i of ith Expression, the EPT computes the operator embedding vector $\mathbf{f}_i$ as in Vaswani et al. (2017)'s setting:

$$\mathbf{f}_i = \mathrm{LN}_{\mathrm{f}}\left(c_{\mathrm{f}}\mathrm{E}_{\mathrm{f}}(f_i) + \mathrm{PE}(i)\right), \quad (2)$$

where $\mathrm{E}_*(\cdot)$ indicates a look-up table for embedding vectors, c_* denotes a scalar parameter, and $\mathrm{LN}_*(\cdot)$ and $\mathrm{PE}(\cdot)$ represent layer normalization (Ba et al., 2016) and positional encoding (Vaswani et al., 2017), respectively.

The embedding vector $\mathbf{a}_{ij}$, which represents the jth operand of ith Expression, is calculated differently according to the operand a_{ij}'s source. To reflect contextual information of operands, three possible sources are utilized: problem-dependent numbers, problem-independent constants, and the result of prior Expression tokens. First, problem-dependent numbers are numbers provided in an algebraic problem (e.g., '20' in Table 1). To compute $\mathbf{a}_{ij}$ of a number, we reuse the encoder's hidden-state vectors corresponding to such number tokens as follows:

$$\mathbf{a}_{ij} = \mathrm{LN}_{\mathrm{a}}\left(c_{\mathrm{a}}\mathbf{u}_{\mathrm{num}} + \mathbf{e}_{a_{ij}}\right), \quad (3)$$

where $\mathbf{u}_*$ denotes a vector representing the source, and $\mathbf{e}_{a_{ij}}$ is the encoder's hidden-state vector corresponding to the number a_{ij}.[2] Second, problem-independent constants are predefined numbers that are not stated in the problem (e.g., 100 is often used for percentiles). To compute $\mathbf{a}_{ij}$ of a constant, we use a look-up table E_{c} as follows:

$$\mathbf{a}_{ij} = \mathrm{LN}_{\mathrm{a}}\left(c_{\mathrm{a}}\mathbf{u}_{\mathrm{const}} + \mathrm{E}_{\mathrm{c}}(a_{ij})\right). \quad (4)$$

Note that LN_{a}, c_{a} are shared across different sources. Third, the result of the prior Expression token is an Expression generated before the ith Expression (e.g., R_0). To compute $\mathbf{a}_{ij}$ of a result, we utilize the positional encoding as follows[3]:

$$\mathbf{a}_{ij} = \mathrm{LN}_{\mathrm{a}}\left(c_{\mathrm{a}}\mathbf{u}_{\mathrm{expr}} + \mathrm{PE}(k)\right), \quad (5)$$

[2] When two or more tokens form a number in the problem, we averaged all related hidden-state vectors.

[3] Since we want to sustain simultaneous decoding, which is one of the strengths in the Transformer, we use $\mathrm{PE}(k)$ for the kth prior Expression, although it is possible to use decoder hidden state $\mathbf{d}_k$.

where k is the index where the prior Expression a_{ij} generated.

3.2 Output layer of EPT's decoder

The output layer of the EPT's decoder predicts the next operator f_{i+1} and operands $a_{i+1,j}$ simultaneously when the ith Expression token is provided. First, the next operator, f_{i+1}, is predicted as follows:

$$f_{i+1} = \arg\max_{f} \sigma(f|\mathrm{FF}_{\mathrm{out}}(\mathbf{d}_i)), \quad (6)$$

where $\sigma(k|\mathbf{x})$ is the probability of selecting an item k under a distribution following the output of softmax function, $\sigma(\mathbf{x})$.

Second, to utilize the context of operands when predicting an operand, the output layer applies 'operand-context pointers,' inspired by the pointer networks (Vinyals et al., 2015). In the pointer networks, the output layer predicts the next token using attention over candidate vectors. The EPT collects candidate vectors for the next $(i+1)$th Expression in three different ways depending on the source of operands:

$$\begin{array}{ll} \mathbf{e}_k & \text{for the } k\text{th number in the problem,} \\ \mathbf{d}_k & \text{for the } k\text{th Expression output,} \\ \mathrm{E_c}(x) & \text{for a constant } x \end{array} \quad (7)$$

Then the EPT predicts the next jth operand $a_{i+1,j}$, as follows. Let $\mathbf{A}_{ij}$ be a matrix whose row vectors are such candidates. Then, the EPT predicts $a_{i+1,j}$ by computing attention of a query vector Q_{ij} on a key matrix K_{ij}, as follows.

$$Q_{ij} = \mathrm{FF}_{\mathrm{query},j}(\mathbf{d}_i), \quad (8)$$

$$K_{ij} = \mathrm{FF}_{\mathrm{key},j}(\mathbf{A}_{ij}), \quad (9)$$

$$a_{i+1,j} = \arg\max_{a} \sigma\left(a \left| \frac{Q_{ij}K_{ij}^{\top}}{\sqrt{D}} \right.\right). \quad (10)$$

As the output layer is modified to predict an operator and its operands simultaneously, we also modified the loss function. We compute the loss of an Expression by summing up the loss of an operator and the loss of required arguments. All loss functions are computed using cross-entropy with the label smoothing approach (Szegedy et al., 2016).

4 Experimental Setup

4.1 Metric and Datasets

The metric for measuring the EPT model's performance is answer accuracy, which is the proportion of correctly answered problems over the entire set of problems. We regard a problem is correctly answered if a solution to the generated equations matches the correct answer without considering the order of answer-tuple, as in Kushman et al. (2014). To obtain a solution to the generated equations, we use `SymPy` (Meurer et al., 2017) at the end of the training phase.

	ALG514	DRAW-1K	MAWPS
Dataset size			
Problems	514	1,000	2,373
Splits	5-fold	Train 600 Dev., Test 200	5-fold
Complexity of generating equations (per problem)			
Unknown	1.82	1.75	1.00
Op tokens	13.08	14.16	6.20
Complexity of selecting an operand (per problem)			
Numbers	4.26	3.88	2.72
Expressions	7.45	7.95	3.60

Table 3: Characteristics of datasets used in the experiment

For the datasets, we use three publicly available English algebraic word problem datasets[4]: ALG514 (Kushman et al., 2014)[5], DRAW-1K (Upadhyay and Chang, 2016)[6], and MAWPS (Koncel-Kedziorski et al., 2016)[7]. The three datasets differ in terms of size and complexity, as shown in Table 3. The high-complexity datasets, ALG514 and DRAW-1K, require more expressions and unknowns when solving the algebraic problems than the low-complexity dataset, MAWPS. For DRAW-1K, we report the accuracy of a model on the development and test set since training and development sets are provided. For the other two datasets — MAWPS and ALG514, — we report the average accuracy and standard error using 5-fold cross-validation.

4.2 Baseline and ablated models

We examine the performance of EPT against five existing state-of-the-art (SoTA) models. The five models are categorized into three types; model using hand-crafted features, pure neural models,

[4] We provide a preprocessed version of these datasets on https://github.com/snucclab/ept/tree/master/dataset.

[5] http://groups.csail.mit.edu/rbg/code/wordprobs/

[6] https://www.microsoft.com/en-us/download/details.aspx?id=52628

[7] http://lang.ee.washington.edu/MAWPS

and a hybrid of these two types.

- Models using hand-crafted features use expert-defined input features without using a neural model: MixedSP (Upadhyay et al., 2016). Upadhyay et al. (2016) designed a model using a set of hand-crafted features similar to those used by Zhou et al. (2015). Using a data augmentation technique, they achieved the SoTA on ALG514 (83.0%) and DRAW-1K (59.5%).

- Pure neural models take algebraic word problems as the raw input to a neural model and do not require the use of a rule-based model: CASS-RL (Huang et al., 2018) and T-MTDNN (Lee and Gweon, 2020). The CASS-RL, which applied pointer-generator networks to generate Op tokens, achieved the best-performing neural model on ALG514 (44.5%). The T-MTDNN is the SoTA model on MAWPS (78.88%) dataset. T-MTDNN utilized multi-task learning for training a template classification model and a number aligning model.

- Hybrid models are models that are neither purely hand-crafted nor pure neural models: CASS-hybrid (Huang et al., 2018) and DNS (Wang et al., 2019). The CASS-hybrid is the best-performing hybrid model of the CASS-RL and Huang et al. (2017)'s model using hand-crafted features. The DNS is a hybrid model of a sequence-to-sequence model and a model using hand-crafted features. We copied the accuracy of DNS on DRAW-1K from Zhang et al. (2019).

After examining the EPT model performance, we conducted an ablation study to analyze the effect of using two main components of EPT; Expression tokens and operand-context pointers. We compared three types of models to test each of the components: (1) the vanilla Transformer model, (2) the Transformer with Expression token model, which investigates the effect of using Expression tokens, and (3) the EPT, which investigates the effect of using pointers in addition to Expression tokens. Additional details on the input/output of the vanilla Transformer and the Transformer with Expression token models are provided in Appendix A.

4.3 Implementation details

The implementation details of EPT and its ablated models are as follows. To build encoder-decoder models, we used `PyTorch` 1.5 (Paszke et al., 2019). For the encoder, three different sizes of ALBERT models in the `transformers` library (Wolf et al., 2019) are used: `albert-base-v2`, `albert-large-v2`, and `albert-xlarge-v2`. We fixed the encoder's embedding matrix during the training since such fixation preserves the world knowledge embedded in the matrix and stabilizes the entire learning process. For the decoder, we stacked six decoder layers and shared the parameters across different layers to reduce memory usage. We set the dimension of input vector D as the same dimension of encoder hidden-state vectors. To train and evaluate the entire model, we used teacher forcing in the training phase and beam search with 3 beams in the evaluation phase.

For the hyperparameters of the EPT, parameters follow the ALBERT model's parameters except for training epoch, batch size, warm-up epoch, and learning rate. First, for the training epoch T, a model is trained in 500, 500, and 100 epochs on ALG514, DRAW-1K, and MAWPS, respectively. For batch sizes, we used 2,048 (`albert-base-v2` and `albert-large-v2`) and 1,024 (`albert-xlarge-v2`) in terms of Op or Expression tokens. To acquire a similar effect of using 4,096 tokens as a batch, we also employed gradient accumulation technique on two types of consecutive mini-batches; two (`base` and `large`) and four (`xlarge`). Then, for the warm-up epoch and learning rate, we conduct the grid-search algorithm for each pair of a dataset and the size of the ALBERT model. For the grid search, we set the sampling space as follows: $\{0.00125, 0.00176, 0.0025\}$ for the learning rates and $\{0, 0.005T, 0.01T, 0.015T, 0.02T, 0.025T\}$ for the warm-up. The resulting parameters are listed in Appendix B. During each grid search, we only use the following training/validation sets and keep other sets unseen: the fold-0 training/test split for ALG514 and MAWPS and the training/development set for DRAW-1K. For the unstated hyperparameters, the parameters follow those of the ALBERT. These parameters include the optimizer and warm-up scheduler; we used LAMB (You et al., 2019) optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 10^{-12}$; and we

Model	ALG 514	DRAW-1K (Dev.)	DRAW-1K (Test)	MAWPS
State of the art (SOTA)				
Hand-crafted	**83.0**[M]		**59.5**[M]	—
Pure neural	44.5[C]		—	**78.9**[T]
Ensembles	82.5[H]		31.0[D]	—
Expression-Pointer Transformer				
EPT (B)	75.46	55.5	51.5	83.41
(Std. Err)	(2.23)			(0.32)
EPT (L)	**81.31**	63.5	59.0	**84.51**
(Std. Err)	(1.88)			(1.37)
EPT (XL)	—*	60.5	**59.5**	—*

Note: [M]MixedSP, [C]CASS-RL, [T]T-MTDNN, [H]CASS-hybrid, [D]DNS. *Overfitted on some folds.

Table 4: Accuracy(%) of the EPT and existing models. (B), (L), and (XL) indicate albert-base-v2, albert-large-v2, and albert-xlarge-v2.

employed linear decay with warm-up scheduling. All the experiment, including hyperparameter search, was conducted on a local computer with 64GB RAM and two GTX1080 Ti GPUs.

5 Result and Discussion

In section 5.1, we first present a comparison study, which examines the EPT's performance. Next, in section 5.2, we present an ablation study, which analyzes the two main components of EPT; Expression tokens and operand-context pointers.

5.1 Comparison study

As shown in Table 4, the performance of EPT is comparable or better in terms of performance accuracy compared to existing state-of-the-art (SoTA) models when tested on the three datasets of ALG514, DRAW-1K, and MAWPS. The fully automatic EPT model, which does not use hand-crafted features, yields comparable performance to existing models using hand-crafted features. Specifically, on the ALG514 dataset, the EPT outperforms the best-performing pure neural model by about 40% and shows comparable performance accuracy to the SoTA model that uses hand-crafted features. On the DRAW-1K dataset, which is harder than ALG514 dataset, a similar performance trend to ALG514 is found. The EPT model outperforms the hybrid model by about 30% and achieved comparable accuracy to the SoTA model that uses hand-crafted features. On the MAWPS dataset, which is only tested on pure neural models in existing studies, the EPT achieves SoTA accuracy.

Model	ALG 514	DRAW-1K (Dev.)	DRAW-1K (Test)	MAWPS
Vanilla Transfo.	27.52	14.5	24.0	79.83
(Std. Err)	(4.39)			(1.03)
+ Expression	42.03	32.0	32.5	80.46
(Std. Err)	(1.97)			(1.09)
+ Pointer (EPT)	75.46	55.5	51.5	83.41
(Std. Err)	(2.23)			(0.32)

Table 5: Accuracy(%) of the EPT and its ablated models (albert-base-v2).

One possible explanation for EPT's outstanding performance over the existing pure neural model is the use of operand's contextual information. Existing neural models solve algebraic word problems by using symbols to provide an abstraction of problem-dependent numbers or unknowns. For example, Figure 1 shows that existing methods used Op tokens, such as x_0 and N_1. However, treating operands as symbols only reflects 2 out of 4 means in which symbols are used in humans' mathematical problem-solving procedures (Usiskin, 1999). The 4 means of symbol usage are; (1) generalizing common patterns, (2) representing unknowns in an equation, (3) indicating an argument of a function, and (4) replacing arbitrary marks. By applying template classification or machine learning techniques, (1) and (2) were successfully utilized in existing neural models. However, the existing neural models could not consider (3) and (4). Therefore, in our suggested EPT model, we dealt with (3) by using Expression tokens and (4) by using operand-context pointers. We suspect that the EPT's performance, which is comparable to existing models using hand-crafted features, comes from dealing with (3) and (4) explicitly when solving algebraic word problems.

5.2 Ablation study

From the ablation study, our data showed that the two components of generating 'Expression' token and applying operand-context pointer, each improved the accuracy of the EPT model in different ways. Specifically, as seen in Table 5, adding Expression token to the vanilla Transformer improved the performance accuracy by about 15% in ALG514 and DRAW-1K and about 1% in MAWPS. In addition, applying operand-context pointer to the Transformer with Expression token

Case 1. Effect of using Expression tokens	Problem	The sum of two numbers is 90. Three times the smaller is 10 more than the larger. Find the larger number.			
	Expected	$3x_0 - x_1 =$	10,	$x_0 + x_1 =$ 90	
	Vanilla Transformer	$x_0 + x_1 =$	3,	$x_0 - x_1 =$ 10	(**Incorrect**)
	+ Expression	$3x_0 - x_1 =$	10,	$x_0 + x_1 =$ 90	(Correct)
	+ Pointer (EPT)	$3x_0 - x_1 =$	10,	$x_0 + x_1 =$ 90	(Correct)
Case 2. Effect of using pointers	Problem	A minor league baseball team plays 130 games in a season. If the team won 14 more than three times as many games as they lost, how many wins and losses did the team have?			
	Expected	$x_0 - 3x_1 =$	14,	$x_0 + x_1 =$130	
	Vanilla Transformer	$14x_0 - 3x_1 =$	0,	$x_0 + x_1 =$130	(**Incorrect**)
	+ Expression	$x_0 - 3x_1 =$	14,	$130x_0 - x_1 =$ 0	(**Incorrect**)
	+ Pointer (EPT)	$x_0 - 3x_1 =$	14,	$x_0 + x_1 =$130	(Correct)
Case 3. Comparative error	Problem	One number is 6 more than another. If the sum of the smaller number and 3 times the larger number is 34, find the two numbers.			
	Expected	$x_0 + 3x_1 =$	34,	$x_1 - x_0 =$ 6	
	Vanilla Transformer	$x_0 + 3x_1 =$	34,	$x_1 - x_0 =$ 6	(Correct)
	+ Expression	$3x_0 + 34x_1 =$	2,	$x_1 - x_0 =$ 6	(**Incorrect**)
	+ Pointer (EPT)	$3x_0 + x_1 =$	34,	$x_1 - x_0 =$ 6	(**Incorrect**)
Case 4. Temporal order error	Problem	The denominator of a fraction exceeds the numerator by 7. if the numerator is increased by three and the denominator increased by 5, the resulting fraction is equal to half. Find the original fraction.			
	Expected	$x_0 - \frac{1}{2}x_1 = \frac{1}{2} \cdot 5 - 3$,		$x_0 - x_1 =$ 7	
	Vanilla Transformer	$3x_0 + 5x_1 =$	$\frac{1}{2}N_4$,		(**Incorrect**)
	+ Expression	$3x_0 - 5x_1 =$	0,	$x_0 + x_1 =$ 7	(**Incorrect**)
	+ Pointer (EPT)	$5x_0 - 3x_1 =$	$\frac{1}{2}$,	$x_1 - x_0 =$ 7	(**Incorrect**)

Table 6: Sample incorrect problems (`albert-base-v2`) from the DRAW-1K development dataset.

model enhanced the performance by about 30% in ALG514 and DRAW-1K and about 3% in MAWPS.

Table 6 shows the result of an error analysis. The cases 1 and 2 show how the EPT model's two components contributed to performance improvement. In case 1, the vanilla Transformer yields an incorrect solution equation by incorrectly associating $x_0 + x_1$ and 3. However, using an Expression token, the explicit relationship between operator and operands is maintained, enabling the distinction between $x_0 + x_1$ and $3x_0 - x_1$. The case 2 example shows how adding an operand-context pointer can help distinguish between different expressions, in our example, x_0, $130x_0$, and $14x_0$. As the operand-context pointer directly points to the contextual information of an operand, the EPT could utilize the relationship between unknown (x_0) and its multiples ($130x_0$ or $14x_0$) without confusion.

We observed that the existing pure neural model's performance on low-complexity dataset of MAWPS was relatively high at 78.9%, compared to that of high-complexity dataset of ALG514 (44.5%). Therefore, using Expression tokens and operand-context pointers contributed to higher performance when applied to high-complexity datasets of ALG514 and DRAW-1K, as shown in Table 5. We suspect two possible explanations for such a performance enhancement.

First, using Expression tokens in high-complexity datasets address the expression fragmentation issue when generating solution equations, which is more complex in ALG514 and DRAW-1K than MAWPS. Specifically, Table 3 shows that on average the number of unknowns in ALG514 and DRAW-1K is almost twice (1.82 and 1.75, respectively) than MAWPS (1.0). Similarly, the number of Op tokens is also twice in ALG514 and DRAW-1K (13.08 and 14.16, respectively) than that of MAWPS (6.20). As the expression fragmentation issue can arise for each token, probability of fragmentation issues' occurrence increases exponentially as the number of unknowns/Op tokens in a problem increases.

Therefore, the vanilla Transformer model, which could not handle the fragmentation issue, yields low accuracy on high-complexity datasets.

Second, using operand-context pointers in high-complexity datasets addresses the operand-context separation issue when selecting an operand, which is more complex in ALG514 and DRAW-1K than MAWPS. Specifically, Table 3 shows that on average the amount of Expression tokens is also twice in ALG514 and DRAW-1K (7.45 and 7.95, respectively) than that of MAWPS (3.60). As numbers and Expression tokens are candidates for selecting an operand, probability of separation issues' occurrence increases linearly as the amount of numbers/Expressions in an equation increases. Since a Transformer with Expression token could not handle the separation issue, the model showed lower accuracy on high-complexity datasets.

In addition to the correctly solved problem examples, Table 6 also shows cases 3 and 4, which were incorrectly answered by the EPT model. The erroneous examples can be categorized into two groups; 'Comparative' error and 'Temporal order' error. 'Comparative' occurs when an algebraic problem contains comparative phrases, such as '6 more than,' as in case 3. 49.3% of incorrectly solved problems contained comparatives. When generating solution equations for the comparative phrases, the order of arguments is a matter for an equation that contains non-commutative operators, such as subtractions or divisions. Therefore, errors occurred when the order of arguments for comparative phrases with non-commutative operators was mixed up. Another group of error is 'Temporal order' error that occurs when a problem contains phrases with temporal orders, such as 'the numerator is increased by three,' as in case 4. 44.5% of incorrectly solved problems contained temporal orders. We suspect that these problems occur when co-referencing is not handled correctly. In a word problem with temporal ordering, a same entity may have two or more numeric values that change over time. For example, in case 4, the denominator has two different values of x_1 and $x_1 + 7$. The EPT model failed to assign a same variable for the denominators. The model assigned x_0 in the former expression and x_1 in the latter.

6 Conclusion

In this study, we proposed a neural algebraic word problem solver, Expression-Pointer Transformer (EPT), and examined its characteristics. We designed EPT to address two issues: expression fragmentation and operand-context separation. The EPT resolves the expression fragmentation issue by generating 'Expression' tokens, which simultaneously generate an operator and required operands. In addition, the EPT resolves the operand-context separation issue by applying operand-context pointers. Our work is meaningful in that we demonstrated a possibility for alleviating the costly procedure of devising hand-crafted features in the domain of solving algebraic word problems. As future work, we plan to generalize the EPT to other datasets, including non-English word problems or non-algebraic domains in math, to extend our model.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2020R1C1C1010162).

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Ting-Rui Chiang and Yun-Nung Chen. 2019. Semantically-aligned equation generation for solving and reasoning math word problems. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2656–2668, Minneapolis, Minnesota. Association for Computational Linguistics.

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

Danqing Huang, Shuming Shi, Chin-Yew Lin, and Jian Yin. 2017. Learning fine-grained expressions to

solve math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 805–814, Copenhagen, Denmark. Association for Computational Linguistics.

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

Rik Koncel-Kedziorski, Subhro Roy, Aida Amini, Nate Kushman, and Hannaneh Hajishirzi. 2016. MAWPS: A math word problem repository. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1152–1157, San Diego, California. Association for Computational Linguistics.

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

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations.

D. Lee and G. Gweon. 2020. Solving arithmetic word problems with a templatebased multi-task deep neural network (t-mtdnn). In *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pages 271–274.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, AMiT Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, Sean Vig, Brian E. Granger, Richard P. Muller, Francesco Bonazzi, Harsh Gupta, Shivam Vats, Fredrik Johansson, Fabian Pedregosa, Matthew J. Curry, Andy R. Terrel, Štěpán Roučka, Ashutosh Saboo, Isuru Fernando, Sumith Kulal, Robert Cimrman, and Anthony Scopatz. 2017. Sympy: symbolic computing in python. *PeerJ Computer Science*, 3:e103.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d' Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Subhro Roy and Dan Roth. 2015. Solving general arithmetic word problems. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1743–1752, Lisbon, Portugal. Association for Computational Linguistics.

Subhro Roy and Dan Roth. 2017. Unit dependency graph and its application to arithmetic word problem solving. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 3082–3088. AAAI Press.

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

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Shyam Upadhyay and Ming-Wei Chang. 2016. Annotating derivations: A new evaluation strategy and dataset for algebra word problems. *CoRR*, abs/1609.07197.

Shyam Upadhyay, Ming-Wei Chang, Kai-Wei Chang, and Wen-tau Yih. 2016. Learning from explicit and implicit supervision jointly for algebra word problems. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 297–306, Austin, Texas. Association for Computational Linguistics.

Zalman Usiskin. 1999. *Algebraic Thinking, Grades K-12: Readings from NCTM's School-Based Journals and Other Publications*, chapter Conceptions of School Algebra and Uses of Variables. National Council of Teachers of Mathematics.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. 2015. Pointer networks. In *Advances in Neural Information Processing Systems 28*, pages 2692–2700. Curran Associates, Inc.

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

Yan Wang, Xiaojiang Liu, and Shuming Shi. 2017. Deep neural solver for math word problems. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 845–854, Copenhagen, Denmark. Association for Computational Linguistics.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771.

Yang You, Jing Li, Jonathan Hseu, Xiaodan Song, James Demmel, and Cho-Jui Hsieh. 2019. Reducing BERT pre-training time from 3 days to 76 minutes. *CoRR*, abs/1904.00962.

D. Zhang, L. Wang, L. Zhang, B. T. Dai, and H. T. Shen. 2019. The gap of semantic parsing: A survey on automatic math word problem solvers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1.

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

A Input/output of ablation models

In this section, we describe how we compute the input and output of the two ablation models: (1) a vanilla Transformer and (2) a vanilla Transformer with 'Expression' tokens. Figure 3 shows the two models.

The first ablation model is a vanilla Transformer. The model generates an 'Op' token sequence and does not use operand-context pointers. The model manages an 'Op' token vocabulary that contains operators, constants, variables, and number placeholders (e.g., N_0). So the input of this model's decoder only utilizes a look-up table for embedding vectors. For the decoder's output, the vanilla Transformer uses a feed-forward softmax layer to output the probability of selecting an Op token. In summary, the input vector $\mathbf{v}_i$ of a token t_i and the output t_{i+1} can be computed as follows.

$$\mathbf{v}_i = \mathrm{LN}_{\mathrm{in}}\left(c_{\mathrm{in}}\mathrm{E}_{\mathrm{in}}(t_i) + \mathrm{PE}(i)\right), \quad (11)$$

$$t_{i+1} = \arg\max_t \sigma\left(\mathrm{FF}_{\mathrm{out}}(\mathbf{d}_i)\right)_t. \quad (12)$$

The second ablation model is a vanilla Transformer model that uses 'Expression' tokens as a unit of input/output. This model generates an 'Expression' token sequence but does not apply operand-context pointers. Instead of using operand-context pointers, this model uses an operand vocabulary that contains constants, placeholders for numbers, and placeholders of previous Expression token results (e.g., R_0). The input of this model's decoder is similar to that of EPT's decoder, but we replaced the equations 3 and 5 with the following formulae.

$$\mathbf{a}_{ij} = \mathrm{LN}_{\mathrm{a}}\left(c_{\mathrm{a}}\mathbf{u}_{\mathrm{num}} + \mathrm{E}_{\mathrm{c}}(a_{ij})\right), \quad (13)$$

$$\mathbf{a}_{ij} = \mathrm{LN}_{\mathrm{a}}\left(c_{\mathrm{a}}\mathbf{u}_{\mathrm{expr}} + \mathrm{E}_{\mathrm{c}}(a_{ij})\right). \quad (14)$$

For the output of this model's decoder, we used a feed-forward softmax layer to output the probability of selecting an operand. Since the softmax output can select the unavailable operand, we set the probability of such unavailable tokens as zeros to mask them. So, we replace equation 10 with the following formula.

$$a_{i+1,j} = \arg\max_a \sigma\left(a \,|\mathrm{M}\left(\mathrm{FF}_{\mathrm{j}}(\mathbf{d}_i)\right)\right), \quad (15)$$

where M is a masking function to set zero probability on unavailable tokens when generating ith Op token. The other unstated equations 1, 2, 4, and 6 remain the same.

B Hyperparameters used for this study

Table 7 shows the best parameters and performances on the development set, which are found using grid search.

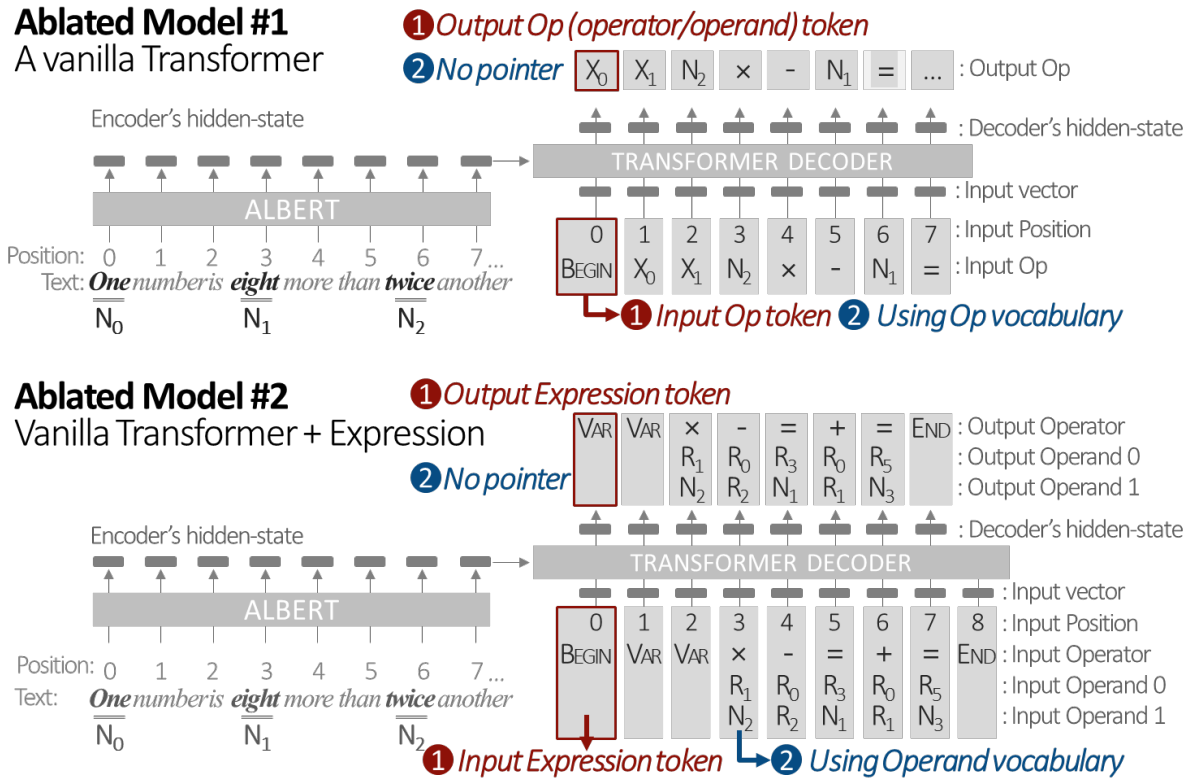

Figure 3: The architecture of two ablated model of EPT: a vanilla Transformer and a Transformer using Expression tokens

Model	# of Params	Hyper-parameters Learning Rate	Warm-up	Performance on Dev.
ALG514 dataset				
EPT (B)	25.29M	.00176	2.0% (10.0 epochs)	78.43
(L)	41.85M	.0025	2.5% (12.5 epochs)	81.37
(XL)	155.30M	.00176	1.0% (5.0 epochs)	83.33
DRAW-1K dataset				
EPT (B)	25.29M	.00176	2.5% (12.5 epochs)	58.5
(L)	41.86M	.00176	0.5% (2.5 epochs)	63.5
(XL)	155.31M	.00176	1.0% (5.0 epochs)	60.5
MAWPS dataset				
EPT (B)	25.30M	.0025	1.0% (1.0 epoch)	83.33
(L)	41.87M	.0025	0.0% (0.0 epoch)	83.97
(XL)	155.33M	.00176	2.5% (2.5 epochs)	83.97

Table 7: Best performing hyperparameters for each pair of a model and a dataset. (B), (L), and (XL) indicate albert-base-v2, albert-large-v2, and albert-xlarge-v2.