Mathematical Formula Representation via Tree Embeddings

Zichao Wang[1], Andrew Lan[2], and Richard Baraniuk[1]

[1]Rice University [2]University of Massachusetts Amherst
{jzwang, richb}@rice.edu andrewlan@cs.umass.edu

Abstract. We propose a new framework for learning formula representations using tree embeddings to facilitate search and similar content retrieval in textbooks containing mathematical (and possibly other types of) formula. By representing each symbolic formula (such as math equation) as an *operator tree*, we can explicitly capture its inherent structural and semantic properties. Our framework consists of a tree encoder that encodes the formula's operator tree into a vector and a tree decoder that generates a formula from a vector in operator tree format. To improve the quality of formula tree generation, we develop a novel *tree beam search* algorithm that is of independent scientific interest. We validate our framework on a formula reconstruction task and a similar formula retrieval task on a new real-world dataset of over 770k formulae collected online. Our experimental results show that our framework significantly outperforms various baselines.

1 Introduction

Recent years have seen increasing proliferation of *mathematical language* such as equations and formulae. Table 1 shows a few examples of formulae not only in mathematics but also in other scientific subjects that often appear in science, technology, engineering, and math (STEM) textbooks.

A number of practical tasks have recently gained traction because of the ubiquitous presence of mathematical language. For example, one common task is *similar formula retrieval*, i.e., finding relevant formulae similar to a query formula (e.g., [5]). This task arises in a wide range of scenarios, such as when students look for relevant math content in a textbook when doing algebra homework. Another common task is automatic formula generation, which arises in scenarios such as formula auto-completion and math summary and headline generation [24]. Both these tasks are potentially labor-intensive and time-consuming; an automatic method that tackles these tasks would be of great benefit. In this work, we focus on *mathematical language processing* (MLP), which involves the formula representation problem, i.e., processing a formula into an appropriate format for downstream tasks such as similar formula retrieval and formula generation.

$N = \left\lfloor 0.5 - \log_2 \left(\frac{\text{Frequency of this item}}{\text{Frequency of most common item}} \right) \right\rfloor$	(physics)
${}^{238}_{92}\mathrm{U} + {}^{64}_{28}\mathrm{Ni} \rightarrow {}^{302}_{120}\mathrm{Ubn}^{*} \rightarrow$ *fission only*	(chemistry)
$ax^2 + bx + c = 0$	(algebra)

Table 1: A few examples of mathematical language in our context.

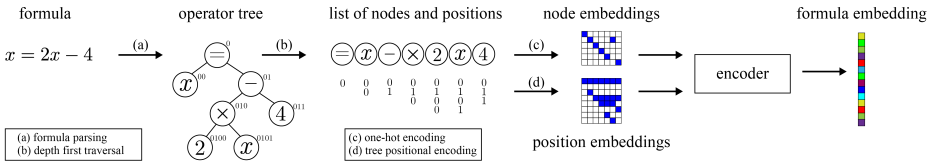
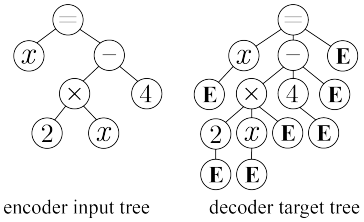


Fig. 1: Illustration of FORTE's encoding process of a formula.

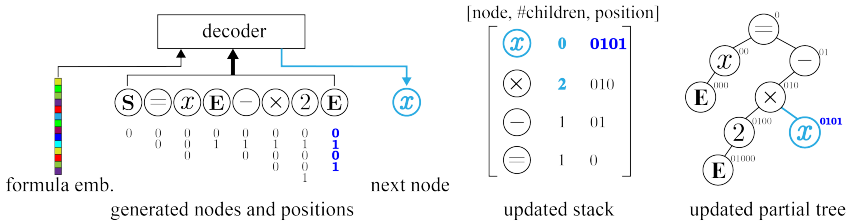
Existing research mostly focuses on either the retrieval or the generation task but rarely both. In terms of formula retrieval, an emerging line of research explores the idea of *symbolic tree* representation. Indeed, mathematical formulae are inherently hierarchical and tree structures are appropriate for organizing the math symbols in a formula. Compared to representing a formula simply as a *sequence* of math symbols, the symbolic tree representation has the advantage to encode both the semantics and the inherent hierarchical structure of a formula. Similar to works in natural language processing (NLP) that leverage inherent language structure (i.e., [20,16,22,12,18]), a number of recent works in formula retrieval exploit formula's unique structural properties, leading to improved results [5,28,11,27] compared to other formula representations, e.g., [6]. However, none of the aforementioned works is capable of *generating* a formula.

In terms of formula generation, existing research usually combines processing mathematical and natural language. For example, [23] trains a topic model on scientific documents and learns the keywords (topics) of a formula. [24] generates a headline from mathematical questions. However, these works treat formulae as sequences of math symbols and thus neither leverage formula's inherent tree structure. Some other works focus on solving math problems, i.e., generating a solution for an input formula [9,15]. However, these works are fully supervised, i.e., they rely on large, labeled datasets that are difficult to collect. To overcome the data issue, these works design methods to artificially generate formulae. However, such synthetic data is simple and does not cover many complicated formulae in practice (e.g., matrices), limiting the generalization capability of models trained on such data.

Contributions. We propose **FORTE**, a novel *unsupervised* framework for Mathematical **FO**rmula **R**epresentation learning via **T**ree **E**mbeddings. Our framework fully exploits the tree structure of math formulae to enable both effective formula encoding and formula generation. FORTE consists of 2 key components.



(a) Input and output formula tree.



(b) The decoding process.

Fig. 2: (**2a**) Illustration of FORTE's input and output operator tree of the same formula. The "E" nodes represents the special "end" node attached as the last child to every node. (**2b**) Illustration of FORTE's decoding process at a particular time step. First, the position of the next node to be generated is computed (dark blue). Next, the next node (light blue) is generated by the decoder using already generated nodes and positions and the newly computed position. Finally, the partial tree and the stack are updated.

First, a *tree encoder* encodes a formula tree into an embedding that can benefit various downstream tasks, i.e., formula retrieval. Second, a *tree decoder* generates a formula tree from an embedding. We also propose a novel *tree beam search* algorithm that extends the beam search in sequence-to-sequence models (e.g., [19,1]) to improve the generation quality for tree-structured data. To evaluate our framework, we have collected a dataset of over 770k formulae, the largest to date to our knowledge, from professional, real-world sources such as Wikipedia and arXiv articles. On a formula autoencoding task and a formula retrieval task, we show that our framework (sometimes significantly) outperforms existing methods.

2 The FORTE Framework

We now present our FORTE framework. We first describe the tree representation of a formula (operator tree), which forms the foundation of our framework. We then set up the MLP problem and introduce the various FORTE components, including the tree encoder, tree decoder, and tree beam search. Figures 1–?? together provide a high-level overview of our framework.

2.1 Formulae As Operator Trees

Every formula X is inherently tree structured [26,5] and can be represented as a symbolic *operator tree* (OT):

$$X = (U, <), \quad u \in U, \quad U \subseteq V \tag{1}$$

where u is a math symbol (a node in OT),[1] U is the set of math symbols in the operator tree X, and V is the "vocabulary", i.e., all unique math symbols in the data set. $<$ represents partial binary parent-child relation $\forall u \in U$ [8]. Procedure (a) in Fig. 1 illustrates the conversion from a formula to its OT. Intuitively, the OT organizes the math symbols in a formula, such as operators, variables, and numerical values, as *nodes* in an explicit, hierarchical tree structure. We choose OT because of its intuitive interpretation and rich semantics. We emphasize that our FORTE framework is agnostic to the underlying tree representation; other tree representations such as symbol layout tree [5] can also be used.

Math Symbol Vocabulary. The size of the math symbol vocabulary V may be unbounded (e.g., every element in the real number set $\mathbb{R}$, which is uncountably infinite, could be an element in V); however, most symbols rarely appear. We thus propose the following truncation method in order to work with a finite vocabulary in practice. First, we partition the vocabulary V into five disjoint sub-vocabulary according to symbol *types*, including numeric V_{num} (numbers, decimals), functional V_{fun} (multiplication, subtraction etc.), variable V_{var}, textual V_{txt} and others V_{o}. We do so because different types of math symbols carry different semantic meanings. Then, we retain only the most frequent K symbols in each sub-vocabulary and convert others to an "unknown" symbol *specific to each type*. This setup guarantees that the semantics of symbols that do not occur frequently are preserved.

2.2 The MLP Problem Formulation

We set up the MLP problem as an unsupervised "autoencoding" task (see e.g., Ch.14 in [7]), motivated by the downstream tasks that we envision our framework will perform. Specifically, our framework aims to reconstruct the input formula in its OT representation through an encoder-decoder bottleneck model design. This problem setup allows us to use the latent embedding from the encoder output for many downstream tasks, i.e., formula retrieval, and the generated formula from the decoder output for generation-related tasks.

Concretely, the training objective of our framework is

$$\mathcal{L}(\theta, \phi) = -\frac{1}{N} \sum_{i=1}^{N} \mathcal{P}_x(X_{\text{out}}^{(i)}) \log f_d(f_e(X_{\text{in}}^{(i)}; \theta); \phi) \,, \tag{2}$$

[1] We will refer to "math symbol" and "node" interchangeably depending on context.

where N is the number of formulae, f_e is the encoder function with parameter θ, f_d is the decoder function with parameter ϕ and $\mathcal{P}_x$ is the empirical data distribution. $X_{\text{in}}^{(i)}$ and $X_{\text{out}}^{(i)}$ are the input and output representations of the i-th formula tree in our new dataset, which we will introduce in Sec. 3. We will drop the data point index i in the remainder of the paper for simplicity of exposition.

2.3 Formula Tree Encoder

Our *tree encoder* takes a formula tree as input and outputs an embedding of this formula. The key idea is to properly encode all information underlying the formula tree. To this end, we use two methods including *tree traversal*, which extracts content (node) information, and *tree positional encoding*, which extracts structural (relative positions of nodes) information. Figure 1 provides an overview of our formula tree encoder.

Formula Tree Traversal and Node Embedding. To obtain the content in a formula tree, we employ tree traversal, which visits each node in the tree in a particular order and extracts its content. Process (b) in Fig. 1 illustrates this process. In this work, we consider traversal using depth-first search (DFS), although other traversal orders can also be used. This step returns a DFS-ordered list of nodes $\{u_t\}_{t=1}^{T}$ where t is the position of node u in the DFS order and T is the number of nodes in the formula tree. Each node is then represented as a trainable embedding $\widetilde{\boldsymbol{x}}_t \in \mathbb{R}^M$ with dimension M.

Tree Positional Embedding. To extract the structure of a formula tree, we propose a two-step method that first retains and then embeds the relative positions of nodes in the tree. First, we recursively encode the position t of node u as $q_t \in \mathbb{R}^{\ell_t}$ where $\ell_t = 0, \ldots, T$ is the depth of node u in the tree. q_t is composed of the node's parent's position appended with its relative positions to its siblings. $q_t = [0]$ for the root node. The formula tree in Fig. 1 illustrates this step. For example, the position $[0, 1, 1]$ of the numeric node "4" is composed of $[0, 1]$ which is its parent's position and $[1]$ because it is the second child of its parent. Second, we propose a *binary tree positional embedding* to convert the encoded position q_t of each node to a fixed-dimensional vector $\boldsymbol{p}_t \in \mathbb{R}^D$ where

$$\boldsymbol{p}_t\Big[\lceil\log_2(C)\rceil j{:}\lceil\log_2(C)\rceil j{+}\lceil\log_2(q_t[j])\rceil\Big]{=}\text{bin}(q_t[j]) \qquad \forall j = 0, \ldots, \ell_t\,. \tag{3}$$

C is the maximum degree (i.e., number of children) of all trees in the dataset, $\lceil\cdot\rceil$ is the ceiling function, $\text{bin}(\cdot)$ is the binarization operator (e.g., $\text{bin}(5) = 101$) and $\boldsymbol{p}_t[j]$ selects the j-th index of the vector $\boldsymbol{p}_t$. The resulting dimension of the tree positional embedding $\boldsymbol{p}_t$ is $D = L\log_2(C)$ where L is the maximum depth of all trees in the dataset.

Formula Tree Embedding. To transform the formula tree into its embedding, we utilize an embedding function $f_e : \mathbb{R}^{(M+D)\times T} \to \mathbb{R}^K$ where K is the dimension of the formula tree embedding and T is the total number of nodes in the formula tree. Because M is not necessarily the same as D, we concatenate the node and tree positional embeddings. Concretely, the formula tree embedding is computed as

$$\boldsymbol{h} = f_e(\{\boldsymbol{x}_t\}_{t=1}^T; \theta), \qquad \boldsymbol{x}_t = \left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_t^\top\right]^\top. \tag{4}$$

There are many options for instantiating the embedding function f_e because, thanks to our node and tree positional embedding methods, the tree content and structure are fully preserved in the encoded input sequence. In this work, we use the gated recurrent unit network (GRU) [4] for f_e, but one can freely choose other appropriate models.

Relation to Prior Work. Our tree encoder design differs from existing approaches in 2 regards. First, compared to [20,3], which perform tree traversal *during* training and thus only allow a single data point per iteration, our encoder performs traversal *before* training, which enables mini-batch processing during training. As a result, our approach removes this computationally expensive traversal step from the training process and significantly speeds up training. Second, compared to [17], which uses a onehot-style tree positional embedding, our encoder employs a different binary tree positional embedding which reduces the space complexity from $\mathcal{O}(LC)$ to $\mathcal{O}(L\log_2(C))$. This reduction is especially significant for trees with a large degree.

2.4 Formula Tree Decoder

The decoder takes a formula embedding vector, i.e., the output from our tree encoder, as input and generates a formula tree as output. We face two main challenges when we design the decoder. First, the terminating condition of tree generation is unclear because the formula tree can have multiple leaf nodes, each of which terminates the generation in a single branch but not necessarily the entire tree. Second, the order of generation, i.e., which node to generate next, is unclear because there are at least two directions at each node: its siblings (horizontal) and its children (vertical).

We now present our decoder, which tackles these two challenges by modifying the decoder target from the encoder input (recall that in a usual autoencoding task, the input and target are exactly the same) and generating by traversing the tree. We also present our novel tree beam search generation algorithm, which improves formula tree generation quality during test time.

Modified Decoder Target Tree. To address the challenge of unclear generation termination condition, we propose to slightly modify the decoder target from the input formula tree by attaching an additional special "end" node to each node as its last child. Doing so informs the decoder when a node has no more

Methods	ACC top-1 ↑	ACC top-5 ↑	TED-structural ↓	TED-overall ↓
seq2seqRNN	92.60% (0.19%)	95.56% (0.07%)	0.084 (0.004)	0.176 (0.003)
tree2treeRNN [3]	71.73% (0.91%)	-	0.507 (0.055)	0.709 (0.055)
tree2treeTF [17]	77.20% (3.30%)	-	0.476 (0.069)	0.507 (0.069)
FORTE (binary, greedy)	94.51% (0.13%)	-	0.053 (0.004)	0.125 (0.008)
FORTE (binary, beam)	**94.67% (0.13%)**	**97.38% (0.14%)**	**0.048 (0.004)**	**0.116 (0.007)**
FORTE (onehot, greedy)	94.28% (0.18%)	-	0.058 (0.005)	0.130 (0.008)
FORTE (onehot, beam)	94.42% (0.17%)	97.22% (0.05%)	0.054 (0.006)	0.124 (0.009)

Table 2: Formula reconstruction results. FORTE outperforms all other methods.

children to generate and provides a clear generation termination signal for each node. Figure 2a compares the encoder's input and decoder's target of the same formula tree.

Generation Via Tree Traversal. To address the challenge of the unclear generation order, we propose to traverse the tree during generation in the same order as the encoder. A data structure is required to track the traversal process. Therefore, for an encoder using DFS traversal, We propose to use a *stack* which maintains nodes whose generation is unfinished in the DFS order. In addition to the node itself, the stack also maintains the number of children and the position of each node in the stack.

During generation, we use the current node's embedding $\tilde{\boldsymbol{x}}_t$ together with the *next* node's position $\boldsymbol{p}_{t+1}$ to generate the next node. We do so because a node can have multiple children; therefore, we use the next node's position to inform the model which node to generate next. This is a major difference from typical sequence generation models, i.e., Transformers, in which the input position is the input token's position itself. Concretely, the next node is generated as

$$\begin{aligned} \widehat{u}_{t+1} = \underset{u \in V}{\operatorname{argmax}}\ \operatorname{softmax}(f_d(\{\boldsymbol{x}'_s\}_{s=1}^{t}; \phi))\,, \\ s = 1, \ldots, t \quad \text{and} \quad \boldsymbol{x}'_s = [\widetilde{\boldsymbol{x}}_s; \boldsymbol{p}_{s+1}; \boldsymbol{h}]\,, \end{aligned} \tag{5}$$

where we also concatenate the formula tree embedding $\boldsymbol{h}$ to the decoder's input at each generation step. The next node's position $\boldsymbol{p}_{s+1}$ is computed using the number of children and the position of the current input node; see Section 2.3 for details. When a node's generation finishes, as signaled by the generation of an "end" node, this node is removed from the stack. The "end" node itself is never added to the stack. The entire tree generation is finished when the stack is empty, i.e., no more nodes to expand further. We use a special "start" token to mark the beginning of the generation process. Figure 2b illustrates the generation process.

Tree Beam Search for Tree Generation. The above generation process is a *greedy algorithm* which may be sub-optimal. To improve tree generation quality, we propose the *tree beam search* (TBS). The idea is to maintain a *stack* for each beam, which records the node generation order. During the generation process, the decoder generates the top B most probable next nodes from the current

generated tree of each beam, resulting in a total of B^2 candidate trees to expand. We then select B most probable candidate trees to expand further. This process continues until B trees finish generation or until a preset maximum number of steps have been reached.

TBS extends beam search in NLP (e.g., in [19,1]) which only concerns sequential data and is incapable of generating tree-structured data. Similar to beam search in NLP, by expands the search space of candidate trees by a factor of B, TBS enables more flexible and higher quality generation during the decoding process compared to greedy generation.

Relation to Prior Work. To our knowledge, this is the first beam search algorithm for generating tree structured data. Our work differs from [29] which proposed a tree beam search algorithm for recommender systems that treats the entire dataset as a tree, where each node is a data point (user, item); in our work, we treat each data point (formula) as a tree. Our work also differs from [17] which specifies the number of children each node must have in the tree, significantly constrains the generation process, unnecessarily increases the node vocabulary and leads to worse generation quality; in our work, there are no constraints on the number of children that each node must have, resulting in flexible and varied generated formula tree.

3 Experiments

We conduct two experiments to validate FORTE. In the first experiment, we demonstrate the advantage of FORTE compared to other tree and sequence generation methods for formulae in the formula reconstruction task. In the second experiment, we demonstrate an application of FORTE in the context of the formula retrieval task and show the advantage of FORTE over existing formula retrieval systems. For our framework in all experiments, we use a 2-layer bidirectional GRU for the encoder and a 2-layer unidirectional GRU for the decoder.

Dataset. We collected a large real-world dataset of more than 770k formulae from a subset of articles on Wikipedia and arXiv. We extracted formulae from these articles and processed them into OT representations.

3.1 Formula Reconstruction

In this experiment, we test FORTE's ability to reconstruct a formula. Because some baselines only works on binary trees [3,17], we select a subset of 170k formulae whose operator trees are binary.

Baselines. We consider the following baselines: **seq2seqRNN** which implements the same encoder and decoder as our framework but processes formulae as sequences of math symbols; **tree2treeRNN** [3] which is an RNN-based method capable of encoding and decoding only binary trees; **treeTransformer** [17]

which is a Transformer-based method that shows success only on binary trees. The latter two baselines were originally developed and evaluated on a very different task (program translation) than MLP. We also include four variants of our framework to evaluate the utility of (1) binary against onehot tree positional embedding and (2) TBS against greedy search for tree generation. We construct training, validation, and test sets by splitting the 170k dataset 80%-10%-10%. We train each model 5 times for 50 epochs, record the model with the best performance on the validation set. We then perform formula reconstruction on the test set using beam size $B = 10$ for applicable methods and report the average values of the following metrics on the test set.

Evaluation Metrics. We use two groups of metrics. The first group of metrics measures the reconstruction accuracy, i.e., the percentage of the generated formulae that are exactly the same as the ground-truth. We compute both **ACC top-1**, using only the most probable generated formulae, and **ACC top-5**, using the five most probable generated formulae. The second group of metrics measures how much the generated formula *tree* differs from the ground truth formula *tree*. We use *tree edit distance* (TED) which measures the distance of two trees by computing the minimum number of operations needed, including changing nodes and node connections, to convert one tree to the other. See [14,13] for an overview of the TED algorithm. We compute both **TED-overall** which considers both node and connection editing and **TED-structural** which only considers connection editing.

Results. Table 2 presents the formula reconstruction results comparing our framework with baselines. Comparing to the two tree2tree baselines that struggle at this task, the encoder and decoder designs in FORTE enable near-perfect formulae reconstruction. Comparing to the seq2seqRNN baseline, FORTE shows improvement when processing formulae as OT against as sequences. Moreover, the results from the 4 FORTE variants clearly demonstrates the benefits of binary tree positional embedding and TBS, leading to improvements in all 4 metrics compared to onehot tree positional embedding and greedy search, respectively. We repeat this experiment on the full dataset comparing only seq2seqRNN and FORTE since the tree-based baselines cannot process non-binary trees. FORTE achieves 85.87% compared to seq2seqRNN's 84.30% on the TOP-1 ACC and 90.30% compared to seq2seqRNN's 88.52% on TOP-5 ACC, respectively. These results further show the benefits of representing formulae as OT against as sequences.

3.2 Formula Retrieval

In this qualitative experiment, we evaluate FORTE's capabilities in a formula retrieval application. Given an input formula (query), a retrieval method aims to return the top relevant formulae (retrievals) from a collection of formulae. We use the entire 770k formula dataset to train our framework and then use the trained encoder to obtain an embedding for each formula. For each query,

Methods	Metrics	
	map	**bpref**
Approach0	0.486	0.507
Tangent-S	0.461	0.472
TangentCFT	0.462	0.464
FORTE	0.475	0.485
FORTE-App	**0.509**	**0.513**

Table 3: Formula retrieval results.

we compute the cosine similarity between its embedding and the embedding of each formula in the dataset. Finally, we choose the formulae with the highest similarity scores as the retrievals. We use the queries from the NTCIR-12 formula retrieval task [25]. Table 1 shows a few examples of the queries.

Model and Baselines. We consider three state-of-the-art baselines designed specifically for the formula retrieval task including **Tangent-CFT** [11], which is one of the few data-driven formula retrieval systems to date, and **Tangent-S** [5] and **Approach0** [28], both of which are based on symbolic sub-tree matching and are data independent. We train Tangent-CFT on the same dataset as FORTE.

Evaluation Metrics. Because it is difficult to algorithmically judge the relevance of a retrieval to a query, we perform a human evaluation for this task as follows. First, for each method and each query, we choose the top 25 retrieved formulae and mix them into a single pool of retrievals. Second, for each query, three human evaluators independently provide a ternary rating for each retrieval in the pool, i.e., whether the retrieval is relevant, half-relevant, or irrelevant to the query. The above evaluation procedure is consistent with [25], including the number of evaluators involved. We then use the mean average precision (**MAP**) [21] and **bpref** [2] as the evaluation metrics. Compared to other retrieval evaluation metrics, Both MAP and bpref are easy to interpret and appropriate for evaluating multiple queries and for comparing multiple retrieval systems.

Results. Table 3 presents the quantitative evaluation results, averaged over the three evaluators' scores. Both metrics indicate that our framework achieves superior performance than the other data-driven baseline, TangentCFT, and one of the data-independent baselines, Tangent-S. Our method falls slightly behind Approach0. This may be caused by the fact that Approach0 uses a more explicit symbolic matching using directly subtrees whereas we rely on the more abstract vector representations of formulae which may lose information during similarity computation. This result is consistent with existing literature [11]. We then propose a new model, FORTE-App that combines the strengths of FORTE and Approach0. The last row of Table 3 shows that the combined method achieves new state-of-the-art performance on the formula retrieval experiment. In particular, the bpref score implies that, on average, in FORTE's retrieved formulae,

relevant formulae (as judged by evaluators) rank higher than irrelevant ones more often than in those retrieved by baselines.

4 Conclusions and Future Work

In this work, we propose FORTE, a novel, unsupervised mathematical language processing framework by leveraging tree embeddings. By encoding formulae as operator trees, we can explicitly capture the inherent structure and semantics of a formula. We propose an encoder and a decoder capable of embedding and generating formula trees, respectively, and a novel tree beam search algorithm to improve generation quality at test time. We evaluate our framework on formula reconstruction and demonstrate our framework's superior performance in both experiments compared to baselines. There are many avenues of future work. One direction is to combine our framework's dedicated capability to encode and generate formulae with state-of-the-art NLP methods to enable cross-modality applications that involve both mathematical and natural language. For example, our framework can serve as a drop-in replacement for the formulae processing part in a number of existing works to potentially improve performance, i.e., in [23] for joint text and math retrieval, in [24] for math headline generation, in [10] for grading students' math homework solutions, and in [15,9] for neural math reasoning. We also look forward to conducting a thorough human evaluation for relevant formula retrieval.

References

1. Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. arXiv e-prints (Sep 2014)
2. Buckley, C., Voorhees, E.M.: Retrieval evaluation with incomplete information. In: Prof. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval. p. 25–32. SIGIR '04 (2004)
3. Chen, X., Liu, C., Song, D.: Tree-to-tree neural networks for program translation. In: Proc. Intl. Conf. Neural Info. Process. Syst. p. 2552–2562 (2018)
4. Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder–decoder for statistical machine translation. In: Proc. Conf. Empirical Methods Natural Lang. Process. pp. 1724–1734 (Oct 2014)
5. Davila, K., Zanibbi, R.: Layout and semantics: Combining representations for mathematical formula search. In: Prof. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval. p. 1165–1168 (2017)
6. Gao, L., Jiang, Z., Yin, Y., Yuan, K., Yan, Z., Tang, Z.: Preliminary Exploration of Formula Embedding for Mathematical Information Retrieval: can mathematical formulae be embedded like a natural language? arXiv e-prints (Jul 2017)
7. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
8. Kunen, K.: Set theory - an introduction to independence proofs, Studies in logic and the foundations of mathematics, vol. 102. North-Holland (1983)

9. Lample, G., Charton, F.: Deep learning for symbolic mathematics. In: Proc. Intl. Conf. Learn. Representations (2020), https://openreview.net/forum?id=S1eZYeHFDS
10. Lan, A.S., Vats, D., Waters, A.E., Baraniuk, R.G.: Mathematical language processing: Automatic grading and feedback for open response mathematical questions. In: Proc. ACM Conf. Learn. @ Scale. p. 167–176 (2015)
11. Mansouri, B., Rohatgi, S., Oard, D.W., Wu, J., Giles, C.L., Zanibbi, R.: Tangent-cft: An embedding model for mathematical formulas. In: Proc. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval. p. 11–18 (2019)
12. Nguyen, X.P., Joty, S., Hoi, S., Socher, R.: Tree-structured attention with hierarchical accumulation. In: Proc. Intl. Conf. Learn. Representations (2020), https://openreview.net/forum?id=HJxK5pEYvr
13. Pawlik, M., Augsten, N.: Efficient computation of the tree edit distance. ACM Trans. Database Syst. **40**(1) (Mar 2015)
14. Pawlik, M., Augsten, N.: Tree edit distance: Robust and memory-efficient. Info. Syst. **56**, 157 – 173 (2016)
15. Saxton, D., Grefenstette, E., Hill, F., Kohli, P.: Analysing mathematical reasoning abilities of neural models. In: Proc. Intl. Conf. Learn. Representations (2019), https://openreview.net/forum?id=H1gR5iR5FX
16. Shen, Y., Tan, S., Sordoni, A., Courville, A.: Ordered neurons: Integrating tree structures into recurrent neural networks. In: Proc. Intl. Conf. Learn. Representations (2019), https://openreview.net/forum?id=B1l6qiR5F7
17. Shiv, V., Quirk, C.: Novel positional encodings to enable tree-based transformers. In: Proc. Intl. Conf. Neural Info. Process. Syst. pp. 12081–12091 (2019)
18. Sun, J., Han, P., Cheng, Z., Wu, E., Wang, W.: Transformer based multi-grained attention network for aspect-based sentiment analysis. IEEE Access **8**, 211152–211163 (2020)
19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Proc. Intl. Conf. Neural Info. Process. Syst. p. 3104–3112 (2014)
20. Tai, K.S., Socher, R., Manning, C.D.: Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. arXiv e-prints (Feb 2015)
21. Voorhees, E.M., Harman, D.K., et al.: TREC: Experiment and evaluation in information retrieval, vol. 63. MIT press Cambridge (2005)
22. Wang, Y., Lee, H.Y., Chen, Y.N.: Tree transformer: Integrating tree structures into self-attention. In: Proc. Conf. Empirical Methods Natural Lang. Process. and Intl. Joint Conf. Natural Lang. Process. pp. 1061–1070 (Nov 2019)
23. Yasunaga, M., Lafferty, J.: TopicEq: A Joint Topic and Mathematical Equation Model for Scientific Texts. In: Proc. AAAI conf. Artificial Intell. (2019)
24. Yuan, K., He, D., Jiang, Z., Gao, L., Tang, Z., Giles, C.L.: Automatic generation of headlines for online math questions. In: Proc. AAAI conf. Artificial Intell. pp. 9490–9497 (2020)
25. Zanibbi, R., Aizawa, A., Kohlhase, M., Ounis, I., Topic, G., Davila, K.: Ntcir-12 mathir task overview. In: Proc. NTCIR Conf. Eval. Info. Access (2016)
26. Zanibbi, R., Blostein, D.: Recognition and retrieval of mathematical expressions. Intl. J. Document Anal. Recognit. **15**(4), 331–357 (Dec 2012)
27. Zhong, W., Rohatgi, S., Wu, J., Giles, C., Zanibbi, R.: Accelerating substructure similarity search for formula retrieval. In: Proc. European Conf. Info. Retrieval. pp. 714–727 (2020)
28. Zhong, W., Zanibbi, R.: Structural similarity search for formulas using leaf-root paths in operator subtrees. In: Azzopardi, L., Stein, B., Fuhr, N., Mayr, P., Hauff,

C., Hiemstra, D. (eds.) Proc. Intl. Conf. Neural Info. Process. Syst. pp. 116–129 (2019)
29. Zhuo, J., Xu, Z., Dai, W., Zhu, H., Li, H., Xu, J., Gai, K.: Learning optimal tree models under beam search. In: Proc. Intl. Conf. Mach. Learn. vol. 119, pp. 11650–11659 (13–18 Jul 2020)