Scientific Formula Retrieval via Tree Embeddings

Zichao Wang
Dept. Electrical and Computer Engineering
Rice University
Houston, TX, USA
zw16@rice.edu

Mengxue Zhang
College of Information and Computer Sciences
University of Massachusetts, Amherst
Amherst, MA, USA
mengxuezhang@cs.umass.edu

Richard G. Baraniuk
Dept. Electrical and Computer Engineering
Rice University
Houston, TX, USA
richb@rice.edu

Andrew S. Lan
College of Information and Computer Sciences
University of Massachusetts, Amherst
Amherst, MA, USA
andrewlan@cs.umass.edu

***Abstract*—Exploiting the ever-growing corpus of scientific content calls for new ways and means to effectively organize, search, and retrieve scientific formulae. We propose a new data-driven framework for retrieving similar scientific formulae via learned formula representations based on tree embeddings. FORTE (for FOrmula Representation learning via Tree Embeddings) leverages operator tree representations of symbolic scientific formulae (such as math equations) to explicitly capture their inherent structural and semantic properties. FORTE employs i) a tree encoder that encodes the formula's operator tree into an embedding vector and ii) a tree decoder that directly generates a formula's operator tree from the embedding vector. We also develop a novel tree beam search algorithm that improves the quality of the decoded operator trees. We demonstrate that FORTE (sometimes significantly) outperforms various baseline methods on formula reconstruction and retrieval using a real-world dataset comprising 770k scientific formulae collected online.**

***Index Terms*—representation learning, scientific formulae understanding, information retrieval, generative models, tree-structured data**

I. Introduction

Recent years have seen increasing proliferation of *scientific formulae* as a data format, c.f. Table I. With its unique set of symbols and language structure, scientific formulae complement natural language in concisely and precisely communicating essential scientific knowledge. These formulae are also an indispensable part of an ever-growing scientific corpus. However, the large quantity of these formulae also poses challenges for effectively organizing and synthesizing scientific formulae in order to derive new knowledge and insights. An important and common real-world use case is *formula retrieval*, i.e., finding relevant formulae similar to a query formula (e.g., [1]). This scenario arises when, for example, researchers search for related work in a large collection of scientific papers or when students look for relevant practice problems in a textbook when doing algebra homework. Scientific formula retrieval is labor-intensive and time-consuming for humans, making an automatic method highly beneficial.

978-1-6654-3902-2/21/$31.00 ©2021 IEEE

TABLE I: Examples of scientific formulae in various domains.

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

An emerging line of research for automatic scientific formula retrieval leverages the *symbolic tree* representation of scientific formula since they often have an inherent hierarchical structure that can be well-captured by trees [2]. Compared to representing a formula simply as a *sequence* of symbols, the symbolic tree representation has the advantage to encode both the semantic and structural properties of a formula. several recent works incorporate symbolic tree representations, leading to improved performance on the formula retrieval task [1, 3, 4] over other formula representations, e.g., [5]. However, most of the above approaches are data-independent, i.e., they perform retrieval based on a set of user-defined rules and are thus not capable of *learning* representations from the large collection of scientific formulae to further improve retrieval performance. [6] is one of the few data-driven formula retrieval methods to date, which demonstrate the benefit of *learned* formula representations for formula retrieval compared to data-independent methods. However, [6] does not fully leverage the tree structure since it linearizes formula trees into sequences and then trains a sequential language model on them.

An additional desirable property of a learned formula representation is that it enables interesting applications beyond formula retrieval such as automatically generating content that involves formulae. For example, [7] uses a learned formula representation to generate a textual description for this formula. [8] automatically generates a headline that summarizes the substance of a mathematical question. However, these approaches treat scientific formulae simply as sequences of symbols and thus ignore their inherent hierarchical structure. [9]

validates their rule-based representations for discrete data using a math formula generation task. However, their experiment involves only simple, synthetically-generated math equations with up to one variable, two math functions (sinusoid and exponential), and three numbers. This level of complexity is dwarfed by that of real-world scientific formulae [9]; therefore, it is unclear whether their approach is applicable in this setting. Furthermore, the above methods are *supervised* and therefore must depend on large, *labeled* datasets which are challenging to construct. Therefore, a desirable scientific formula representation learning method should be unsupervised, take advantage of the inherent formula structure, and enable both retrieval and generation.

A. Contributions

We propose **FORTE**, a novel *unsupervised* framework for scientific **FO**rmula **R**epresentation learning via **T**ree **E**mbeddings, capable of both similar formula retrieval and formula generation. Our framework fully exploits the tree structure of scientific formulae in an autoencoding model design to learn an effective representation. FORTE consists of two key components. First, a *tree encoder* encodes a formula in operator tree format into an embedding vector that can be used for various downstream tasks, including similar formula retrieval. Second, a *tree decoder* reconstructs a formula tree from its encoded embedding vector. The decoder can also generate novel, unseen formula trees from any input vector. To improve generation quality, we also propose a novel *tree beam search* method that extends the classic beam search method for sequential data to improve formula generation by keeping multiple trees (possibly with different structures) at every search step. To evaluate our framework, we combine and parse several existing datasets collected from real-world sources (such as Wikipedia and arXiv articles) into a dataset with over 770k scientific formulae. To the best of our knowledge, this dataset is the largest one to date. We conduct extensive quantitative and qualitative experiments on this dataset for both formula reconstruction and similar formula retrieval. Experimental results show that FORTE (sometimes significantly) outperforms various existing methods.

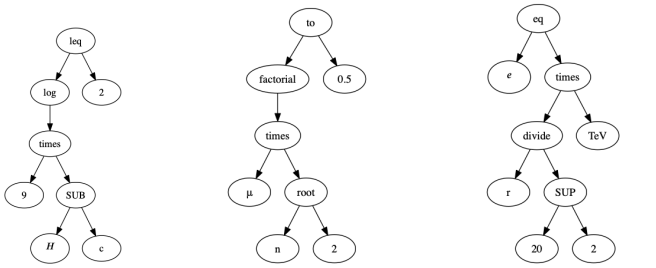
II. The FORTE Framework

We now detail the FORTE framework. We first introduce tree representations of formulae, especially operator trees, which is a crucial pre-processing step in our framework. We then set up the formula representation learning problem and introduce the various FORTE components, including the tree encoder, the tree decoder, and the tree beam search algorithm for formula generation. Figures 2–4 together provide a high-level overview of our framework.

A. Preliminary: Formulae As Operator Trees

A scientific formula is inherently tree structured [1, 10] and can be represented as a symbolic *operator tree* (OT) that we denote as X:

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



(a) $\log(9 \times H_c) \leq 2$ (b) $(\mu \times \sqrt[2]{n}) \to 0.5$ (c) $e = \frac{r}{20^2} \times \text{TeV}$

Fig. 1: Examples of simple formulae and their corresponding OTs. These examples are randomly generated by our proposed FORTE framework; see Section III-B5.

where u is a scientific (most often math) symbol that corresponds to a node in the OT. Throughout the paper, we will refer to "symbol", "scientific symbol", and "node" interchangeably depending on context. U is the set of symbols in X, and V is the "vocabulary", i.e., all unique scientific symbols in the data set. $<$ represents partial binary parent-child relation $\forall u \in U$ [11]. S is the set of all valid OTs. Intuitively, the OT organizes the scientific symbols in a formula, such as math operators, variables, and numerical values, as *nodes* in an explicit tree structure. We remark that OT is not the only tree representation of scientific formulae and there exist other formula tree representations such as the symbol layout tree [1]. In principle, our FORTE framework is agnostic to the underlying tree representation; we choose OT because of its intuitive interpretation and ability to preserve the semantic and structural information in scientific formulae. Figure 1 gives a few simple examples of formulae and their corresponding OTs.

B. Problem Formulation

We set up the formula representation learning problem as an unsupervised "autoencoding" task, motivated by the downstream tasks that we envision our framework will perform, including formula generation and retrieval. Specifically, our framework aims to reconstruct the input formula in its OT representation through an encoder-decoder design with a bottleneck embedding vector. This problem setup enables us to use the latent embedding obtained from the encoder for many downstream tasks, i.e., formula retrieval, and the generated formula from the decoder for generation-related tasks.

Given our autoencoding task formulation, we use a model structure consisting of an encoder and a decoder, where the encoder encodes a scientific formula into a vector representation and the decoder reconstructs a scientific formula from its vector representation. Concretely, we have

$$\begin{aligned} \text{encode}: &\ \boldsymbol{e} = f_{\text{enc}}(X; \Theta), \\ \text{decode}: &\ X_{\text{out}} \sim p_{\text{dec}}(\boldsymbol{e}; \Phi), \end{aligned}$$

where $\boldsymbol{e} \in \mathbb{R}^M$ is the M-dimensional vector representation of a formula. f_{enc} and p_{dec} are the encoder and decoder models parametrized by a set of parameters Θ and Φ, respectively. We

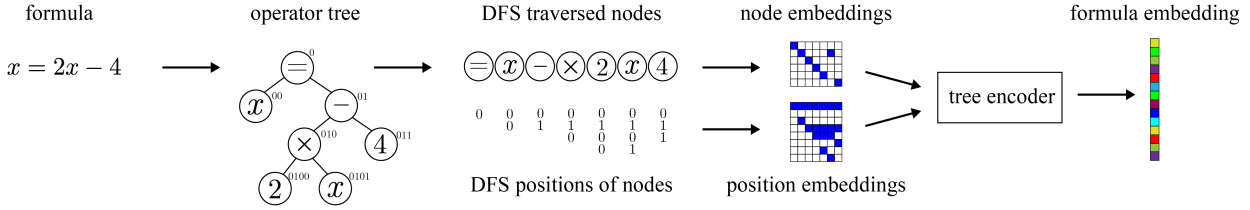


Fig. 2: Illustration of FORTE's encoding process of a formula.

will detail their specific forms in Sections II-C and II-D. X_{out} denotes the formula tree for the decoder, which is slightly modified from X. The reason that we need a different tree format for the decoder will become clear when we detail our decoder design in Section II-D.

1) Optimization: Given our formulation and a dataset of N scientific formulae, we train f_{enc} and p_{dec} jointly to maximize the scientific formula reconstruction accuracy by minimizing the cross entropy loss

$$\mathcal{L}(\Theta, \Phi) = \frac{1}{N} \sum_{i=1}^{N} -\log p_{\text{dec}}(f_{\text{enc}}(X^{(i)}; \Theta); \Phi) \,. \tag{2}$$

i is the index of formulae, which we will drop for notation simplicity whenever we discuss a single formula.

2) Formula Retrieval and Generation: In formula retrieval, we are given a query formula X_{q} and a collection $\mathcal{D}$ of candidate formulae $X_{\text{r}} \in \mathcal{D}$ for retrieval. We will first compute the embeddings of the query and the candidate formulae and then select a subset $\mathcal{G} \subset \mathcal{D}$ consisting of those that are most similar (e.g., measured by cosine similarity scores). Concretely, we have:

$$\mathcal{G} = \Big\{ X_r^{(i)} \in \mathcal{D} \;\Big|\; |\{X_r^{(i)'} \in \mathcal{D} : \text{sim}(\boldsymbol{e}_r, \boldsymbol{e}_q) < \text{sim}(\boldsymbol{e}'_r, \boldsymbol{e}_q)\}| < R \Big\}$$

where $\boldsymbol{e}_r$ and $\boldsymbol{e}_q$ are the embeddings for candidate retrieval X_r and query X_q, respectively. $\text{sim}(\boldsymbol{e}_r, \boldsymbol{e}_q)$ is the cosine similarity function

$$\text{sim}(\boldsymbol{e}_r, \boldsymbol{e}_q) = \frac{\boldsymbol{e}_r^\top \boldsymbol{e}_q}{\|\boldsymbol{e}_r\| \|\boldsymbol{e}_q\|} \,.$$

In formula generation, given an input vector $\boldsymbol{e}$ (not necessarily a formula embedding), the decoder generates a formula X' with the highest log-likelihood:

$$X' = \underset{X_{\text{out}}}{\text{argmax}} \, \log p_{\text{dec}}(X_{\text{out}} \,|\, \boldsymbol{e}; \Phi) \,.$$

C. Formula Tree Encoder

Recall that our *tree encoder* takes a formula tree X as input and outputs an embedding of this formula $\boldsymbol{e}$. 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 information (the symbol each node corresponds to), and *tree positional encoding*, which extracts structural information (the relative positions of nodes). These methods are inspired by prior works in program translation [12, 13] and natural language processing [14, 15] that achieve state-of-the-art performance. These works involve traversing structured data (e.g., the compiler stack of a program, the parse tree of a sentence) and keeping the traversal order, which resembles what we employ in our work. Figure 2 provides an overview of our formula tree encoder.

1) Formula Tree Traversal and Node Embedding.: We traverse each formula tree in the depth first search (DFS) order to extract the node symbols. This step returns a DFS-ordered list of nodes $\{u_t\}_{t=1}^{T}$ where t indexes the nodes u's in the DFS order and T is the total number of nodes in the formula tree. Each node u_t is then represented as a trainable embedding $\widetilde{\boldsymbol{x}}_t \in \mathbb{R}^M$.

2) Tree Positional Encoding.: To extract the structure of a formula tree, we propose a two-step method that first computes and then embeds the relative positions of nodes in the tree.

Let v be the parent of node u. Let q_v and q_u be the positions of v and u, respectively. Let n_u denote that u is the n-th child of v from left to right; n starts with 0. In the first step, we compute the position q_u of each node as follows:

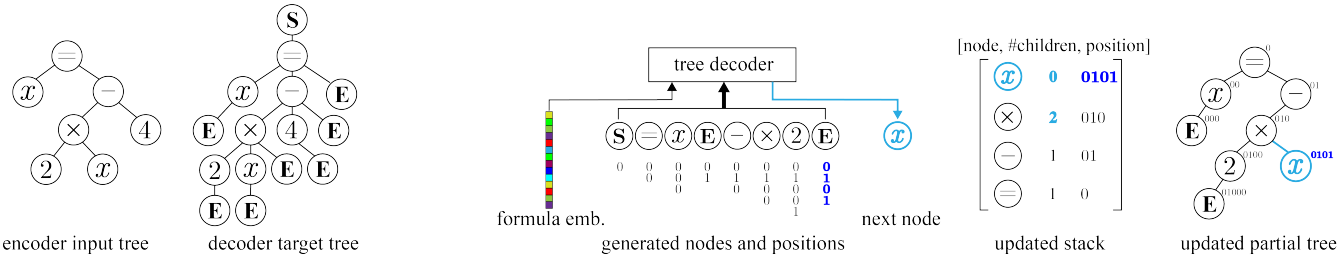
$$q_u = 10 q_v + n_u \,. \tag{3}$$

When u is the root node, we set q_u = 0. The above construction is intuitive and informative because i) the number of digits in the biggest q_u in a tree represents the depth of this tree and ii) the largest number of all q_u in a tree represents the maximum degree of this tree. The formula OT in Figure 2 illustrates the above computation. For example, the position 011 of the numeric node "4" is composed of 01 which is its parent's position and 1 because it is the second child of its parent.

The numeric values of different positions p_u may differ significantly, i.e., between 0 of the root node and tens of thousands of a leaf node in trees that are deep. Therefore, in the second step, we embed these positions p_u into fixed-dimensional *tree positional embeddings*. We propose a binary tree positional embedding method where each digit in p_u is converted to its corresponding binary number in the base-2 numeric system and then concatenated together. Concretely, let $p_u = [p_{u,1}, \ldots, p_{u,l}]_{10}$ be the base-10 representation of p_u where $l = \lfloor \log_{10}(p_u) \rfloor$ is the number of digits in p_u and $p_{u,j}$ is the j-th digit in p_u from left to right. Let $\boldsymbol{q}_u$ be the embedding of q_u. Then:

$$\widetilde{\boldsymbol{q}}_u = \left[\text{bin}(p_{u,1})^\top, \ldots, \text{bin}(p_{u,l})^\top \right]^\top \in \mathbb{R}^{D_u} \,, \tag{4}$$

$$\boldsymbol{q}_u = \left[\widetilde{\boldsymbol{q}}_u^\top, \mathbf{0}_{D-D_u}^\top \right]^\top \in \mathbb{R}^{D} \,, \tag{5}$$



(a) Input and output formula trees.

(b) FORTE's decoding process.

Fig. 3: (**3a**) Illustration of FORTE's input and output operator trees of the same formula. The "S" node represents the special "<start>" node at the root of the tree. The "E" nodes represent the special "<end>" node attached as the last child to every node. (**3b**) 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.

where $\mathrm{bin}(\cdot)$ is the binarization operator (e.g., $\mathrm{bin}(5) = 101$). $D = L\log_2(C)$ where L and C are the maximum depth and maximum degree of all formula trees under consideration. $\mathbf{0}_{D-D_u} \in \mathbb{R}^{D-D_u}$ is an all-zero vector to make the dimension of every $\boldsymbol{q}_u$ the same.

3) Formula Tree Embedding.: To transform the formula tree into its embedding, we utilize an embedding function $f_{\text{enc}} : \mathbb{R}^{(M+D)\times T} \rightarrow \mathbb{R}^K$ where K is the dimension of the formula tree embedding and T is the total number of nodes in a formula tree. We concatenate the node and tree positional embeddings such that the encoder is aware of both the nodes and their positions. The concatenation setup enables more modeling flexibility because we do not need to enforce $M = D$. Concretely, the formula tree embedding is:

$$\boldsymbol{e} = f_{\text{enc}}(\{\boldsymbol{x}_t\}_{t=1}^{T}; \Theta), \qquad \text{where } \boldsymbol{x}_t = \left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_t^\top\right]^\top. \quad (6)$$

In this work, we use a bidirectional gated recurrent unit network (bi-GRU) [16] that recurrently computes a hidden state $\boldsymbol{h}_t$ for each $\boldsymbol{x}_t$. The forward direction is computed as follows:

$$\overrightarrow{\boldsymbol{z}_t} = \sigma(\mathbf{W}_z\left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_t^\top\right]^\top + \mathbf{U}_z\overrightarrow{\boldsymbol{h}}_{t-1} + \boldsymbol{b}_z), \quad (7)$$

$$\overrightarrow{\boldsymbol{r}_t} = \sigma(\mathbf{W}_r\left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_t^\top\right]^\top + \mathbf{U}_r\overrightarrow{\boldsymbol{h}}_{t-1} + \boldsymbol{b}_r), \quad (8)$$

$$\overrightarrow{\boldsymbol{c}_t} = \sigma(\mathbf{W}_c\left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_t^\top\right]^\top + \overrightarrow{\boldsymbol{r}_t} \cdot \mathbf{U}_c\overrightarrow{\boldsymbol{h}}_{t-1} + \boldsymbol{b}_c), \quad (9)$$

$$\overrightarrow{\boldsymbol{h}_t} = \overrightarrow{\boldsymbol{z}_t} \cdot \overrightarrow{\boldsymbol{h}}_{t-1} + (1 - \overrightarrow{\boldsymbol{z}_t}) \cdot \overrightarrow{\boldsymbol{c}_t}, \quad (10)$$

where $\mathbf{W}$, $\mathbf{U}$, and b are part of the set of parameters Θ. The backward direction $\overleftarrow{\boldsymbol{h}_t}$ is computed similarly, with the same Θ. The final formula embedding $\boldsymbol{e}$ is a simple weighted combination of the latent states $\overrightarrow{\boldsymbol{h}_t}$ and $\overleftarrow{\boldsymbol{h}_t}$:

$$\boldsymbol{e} = \sum_{t=1}^{T} a_t\boldsymbol{h}_t, \qquad \boldsymbol{a} = \mathrm{softmax}(\mathbf{W}_a^\top[\boldsymbol{h}_1, \ldots, \boldsymbol{h}_T]),$$

where $\boldsymbol{h}_t = [\overrightarrow{\boldsymbol{h}_t}^\top, \overleftarrow{\boldsymbol{h}_t}^\top]^\top$ and $\mathbf{W}_a$ is also part of Θ.

D. Formula Tree Decoder

The decoder takes a formula embedding vector, i.e., the output from our tree encoder, as input and generates a formula in OT format. In contrast to decoders often used in NLP that generate a sequence of symbols as output, we develop a decoder that generates symbols laid out in a tree. Our tree generation strategy leverages the fact that one only needs to know all symbols in the tree and all symbols' positions in the tree to perform reconstruction. Using this insight, our decoder first computes the next node's position and then generates the next node symbol at a given time step. We first describe node symbol generation and node position computation in the context of greedy tree generation. We then propose a tree beam search algorithm that extends and improves greedy tree generation.

1) Computing the Node Positions: The key difference between the decoder and the encoder regarding the tree positional embedding: in the decoder, the tree position embedding at time step t is *not* for the node at time step t (recall Eqs. 7–10) but rather for the node at time step $t+1$. The reason for this design is that, unlike the encoder that has access to all positions for all nodes in the input formula tree, the decoder has no positional information and needs to compute the positions for all nodes during the generation process in addition to generating the node symbols themselves. The $t+1$-th node's position is computed as Eq. 3, using its parent's position, and its parent's current number of children. Knowing a node's parent during generation requires maintaining the structure of the current, partially generated tree, which we detail in Sections II-D3 and II-D4.

2) Generating the Node Symbols: For node symbol generation, we use a causal, uni-directional GRU network, in which the hidden state at each recurrent step is computed as:

$$\boldsymbol{z} = \sigma(\mathbf{W}_z\left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_{t+1}^\top, \boldsymbol{x}^\top\right]^\top + \mathbf{U}_z\boldsymbol{s}_t + b_z),$$

$$\boldsymbol{r} = \sigma(\mathbf{W}_r\left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_{t+1}^\top, \boldsymbol{x}^\top\right]^\top + \mathbf{U}_r\boldsymbol{s}_t + b_r),$$

$$\boldsymbol{c} = \sigma(\mathbf{W}_c\left[\widetilde{\boldsymbol{x}}_t^\top, \boldsymbol{p}_{t+1}^\top, \boldsymbol{x}^\top\right]^\top + r \cdot \mathbf{U}_c\boldsymbol{s}_t + b_c),$$

$$\boldsymbol{s}_{t+1} = z \cdot \boldsymbol{s}_t + (1 - z) \cdot \boldsymbol{c}.$$

$$\boldsymbol{y}_{t+1} = \mathrm{softmax}\left(\mathbf{W}_y\boldsymbol{s}_t + b_y\right).$$

Here, t denotes the time step in DFS order, which is consistent with the encoder's node traversal order. $\widetilde{\boldsymbol{x}}_t$ and $\boldsymbol{s}_t$ are the

embedding of the generated node and the decoder hidden state at the t-th time step, respectively. We concatenate the embedding of the input formula tree $\boldsymbol{e}$ at each step of the generation to inform the decoder and guide the generation towards the formula tree corresponding to $\boldsymbol{e}$, similar to [12]. $\boldsymbol{p}_{t+1}$ is the tree positional embedding of the *next* node, which is decided given the DFS order and the symbol generated for the last node (see Section II-D1 for details). $\boldsymbol{y}_{t+1}$ is the probability distribution over all symbols to be generated at time step $t+1$. To generated the next node symbol, a simple strategy is greedy search, i.e., the decoder selects the next symbol by choosing the one with the highest probability:

$$u_{t+1} = \underset{i}{\operatorname{argmax}}\, \boldsymbol{y}_{t+1}\,.$$

3) Maintaining the Tree Structure Using a Stack: As mentioned in Section II-D1, computing the positions of node u_t requires knowing the node's parent v_t, which then requires the decoder to keep track of the structure of the partial tree generated so far. To do so, we employ a stack S to keep track of node positions in the DFS order. Each element (implemented as a `struct`) in the stack records three items: the node symbol u_t, its position q_t, and its current number of children a_t (recall Section II-C2 for definitions of these variables). This way, the decoder knows that the next node u_{t+1} will be attached to the node v_{t+1} at the top of the stack as its next child. The next node's position can then be computed from q_t and a_t (See Eq. 3).

4) Updating the Stack: Generally speaking, when a new node is generated, we i) increment the number of children of this node's parent and ii) push the element containing this node, its position, and its number of children (which is 0 when the node is just generated) to the stack. When the generation of all children of a node finishes, we pop the element containing this node from the stack. Because the decoder generates nodes on the fly and does not have access to the entire tree structure, we need to know when to finish generating children for a parent node. To do so, we introduce an additional special node "`<end>`", which is attached as the last child of every node. Therefore, whenever the `<end>` node is generated, we finish generating the children of a parent node v and pop the top element of the stack that contains v. The next generation step will use whichever element at the top of the stack now to determine the position of the next node. In addition, to initialize the generation process, we introduce another special node "`<start>`" as the parent of the root of every formula tree. `<start>` and `<end>` nodes modify the encoder input tree X, resulting in X_{out}, which is the target for the decoder. Figure 3a illustrates the modified decoder target formula tree with these additional special nodes and compares it to the encoder input formula tree. The termination condition of tree generation is when the stack is empty, i.e., when there are no more node symbols for which we need to generate children. The stack update process is illustrated in Figure 3b.

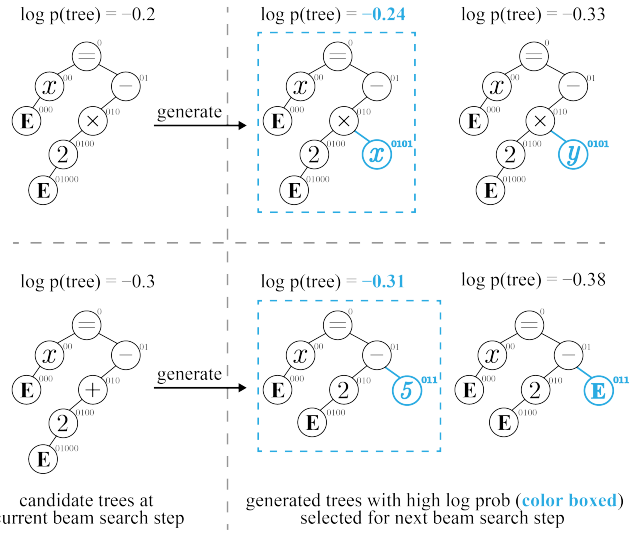


Fig. 4: Illustration of the tree beam search algorithm (TBS) at a particular time step with a beam size of 2. TBS enables search over different formula tree structures.

E. Tree Beam Search for Tree Generation

The generation process detailed above is a *greedy* process that generates each node optimally but not necessarily the optimal tree. To generate higher-quality formula trees, we develop a tree beam search (TBS) algorithm that extends the classic beam search algorithm commonly used for sequence data [17]. The intuition is to maintain a set of B candidate *trees* during the generation process where B is the beam size that controls the size of the search space. Because of the expanded search space, TBS can keep trees with potentially different structures at each time step in the beam. Therefore, TBS improves over greedy search which only keeps one tree that may become suboptimal after more nodes are generated. When we implement TBS in practice, we need to keep B different stacks for each tree in the beam. During the generation process, for each beam, the decoder first decides the position of the next node and then generates the top B most probable symbols for the node, using the current generated tree (symbols, positions) as input. This process results in a total of B^2 candidate trees to select from (B beams and B possible next nodes for each beam), from which we select B most probable candidate trees to keep in the beam. This process continues until B trees finish their generation process (e.g., when their stacks are empty) or until a preset maximum number of steps is reached. The entire TBS algorithm is illustrated in Figure 4 and Algorithm 1.

F. Relation to Prior Work

We now conduct a review of prior works on learning representations for tree/graph-structured data and provide a detailed comparison of the technical components in FORTE with prior works.

1) Tree-Structured Data Analysis: Many data types, including computer programs [12, 13], scientific formulae [2], molecule structures [9], neural network architectures [18], and

the syntax of natural language sentences [15], have inherent structure such as trees and directed acyclic graphs (DAGs). To better leverage these structures in downstream tasks, an important line of work focuses on learning representations for these structured data types.

Our work differs from prior works that deal with tree/graph-structured data in three ways. First, some prior works only consider tree encoding[15, 19, 20, 21, 22], whereas we consider both tree encoding and generation. Second, our encoder design enables more efficient input processing. Compared to [12, 19], which perform tree traversal *during* training and thus can only process 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 intensive traversal step from the training process and significantly speeds up training. Compared to [13], 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 $\mathcal{O}(L\log_2(C))$, resulting in faster computation and less memory usage. This reduction is especially significant for trees with a large degree C. Third, our decoder design, i.e., the use of the special `<end>` node to signal generation termination, enables flexible formula tree generation, which is suitable for real-world scientific formulae. In contrast, many prior works resort to constrained generation processes by enforcing the generated data to have a rigid structure. For example, [9] and [18] propose methods that extensively leverage the limited data diversity in their applications by pre-specifying possible nodes and edges. Some works also restrict the number of children each node must have [12, 13]. The scientific formulae that we work with are much more complex: they contain numerous distinct nodes and edges and many nodes can also have varying numbers of children. It is thus unclear whether these methods generalize to our problem. Section III-B performs a quantitative comparison between FORTE and some of these applicable baselines and demonstrates FORTE's superior generation performance.

2) Beam Search for Tree Generation: Beam search has been extensively applied for natural language generation tasks [23, 24, 25] but it is only applicable for sequential rather than tree-structured data. Our TBS method thus builds on and significantly extends sequential beam search. [26] is the only prior work that involves beam search over trees to our knowledge, which appears to resemble our proposed TBS at first glance. However, our TBS is fundamentally different from the "tree beam search" in [26]. The tree beam search in [26] is used for *retrieval*, i.e., for searching nodes deemed as relevant to a query by some scoring function. In contrast, our TBS is used for *tree generation* and thus needs to address issues such as generation termination conditions and partial tree structure maintenance, none of which is an issue in [26]. Moreover, the tree construction in [26] differs from ours: they treat the *entire dataset* as a tree in which each node corresponds to a data point, whereas we treat *each data point* (formula) as a tree in which each node corresponds to a scientific symbol. Thus, even though [26] may resemble a single TBS step, our full TBS method (Algorithm 1) is still first-of-its-kind to the best of our knowledge.

Algorithm 1: Tree Beam Search

Require: Decoder p_{dec}, maximum generation step T, beam size B
Input : tree embedding $\boldsymbol{e}$
Output : (node, position) tuples $F = \{(U_b, Q_b)\}_{b=1}^{B}$ where $U_b = \{u_{t,b}\}_{t=1}^{T'}$ and $Q_b = \{q_{t,b}\}_{t=1}^{T'}$

Initialize stacks $S_1, \ldots, S_B$;
Initialize $U_1, \ldots, U_B$, $Q_1, \ldots, Q_B$;
$u_0 =$ `<start>`, $q_1 = 0$;

Generate next B nodes $u_{1,1}, \ldots, u_{1,B}$;
Add $u_{1,b}$ to U_b and $q_{1,b}$ to Q_b for $b \in [1, B]$;
for $b = 1, \ldots, B$ **do**
  if $u_{1,b}$ *is not* `<end>` **then**
    Initialize struct $A_{1,b}$;
    $A_{1,b}.u = u_{1,b}$, $A_{1,b}.a = 0$, $A_{1,b}.q = q_{1,b}$;
    Push A_{1b} onto S_b;
    Compute $q_{2,b}$ via Eq. 3;
  else
    Add (U_b, Q_b) to F;

for $t = 1, \ldots, T$ **do**
  Generate next nodes $u_{t+1,b}$ and update stacks S_b;
  for $b = 1, \ldots, B$ **do**
    Add $u_{t+1,b}$ to U_b and $q_{t+1,b}$ to Q_b;
    if Q_b *not empty* **then**
      Update stack S_b;
      Compute $q_{t+2,b}$ via Eq. 3;
    else
      Add (U_b, Q_b) to F;
  if $\text{card}(F) \geq B$ *or* $S_b = \emptyset\ \forall b$ **then**
    Break;
Return F;

3) Other applications Involving Formulae: There are other applications that involve scientific/mathematical content. For example, some works focus on automatically *solving* math problems, i.e., generating a solution to a given math problem that consists of numbers or math expressions [27, 28]. Other works focus on the interplay between formulae and natural language. Some examples include learning topic words for a formula [7] and automatic summary generation for a math question post [29]. Our work does not consider these applications but rather focuses on the fundamental research question of how to represent scientific formula. Nevertheless, our work can potentially be integrated into some of these applications; we leave these extensions for future work.

III. Experiments

We conduct two experiments to validate FORTE's effectiveness. In the first experiment, we demonstrate the advantage of FORTE over other tree and sequence generation methods in a formula reconstruction task. In the second experiment, we demonstrate the advantage of FORTE over existing methods in a similar formula retrieval task. In all experiments, we implement FORTE using a 2-layer bidirectional GRU as the encoder and a 2-layer unidirectional GRU for the decoder.

TABLE II: Formula reconstruction results. FORTE outperforms all other methods.

Methods	ACC-1 ↑	ACC-5 ↑	TED-struct ↓	TED-full ↓
GVAE [9]	30.10%	-	-	-
DVAE [18]	50.29%	-	1.178	1.791
tree2treeRNN [12]	71.73%	-	0.507	0.709
tree2treeTF [13]	77.20%	-	0.476	0.507
seq2seqRNN	92.60%	95.56%	0.084	0.176
FORTE (binary, greedy)	94.51%	-	0.053	0.125
FORTE (binary, beam)	**94.67%**	**97.38%**	**0.048**	**0.116**
FORTE (onehot, greedy)	94.28%	-	0.058	0.130
FORTE (onehot, beam)	94.42%	97.22%	0.054	0.124

A. Dataset

We collected a large real-world dataset of more than 770k scientific formulae from existing sources including scientific articles on Wikipedia [1] and papers on arXiv.[2,3]

a) Dataset Preprocessing: The formulae in the raw source are in either MathML or LaTeX format. We employ a parser [1] to convert them to the same OT format. We retain formulae that do not incur any errors during the parsing process and whose depth is below 20, degree is below 10, and the number of nodes is below 250. We set these thresholds to remove rare formulae with extremely complicated structures that significantly slow down the training and evaluation process. For some baselines that operate on LaTeX format of the formulae (see Section III-B1), we use the tokenizer in [30] to process each LaTeX formula into a sequence of symbols.

b) Scientific Symbol Vocabulary.: Recall from Section II-A that the scientific symbol vocabulary V contains all unique scientific symbols in all the formula OTs in our dataset. The size of this vocabulary V may be unbounded (e.g., every element in the real number set $\mathbb{R}$, which is uncountably infinite, could appear in V as a separate symbol); however, most symbols rarely appear. We thus propose the following truncation method to work with a finite vocabulary in practice. First, we partition the vocabulary V into five disjoint sub-vocabularies according to five 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.

B. Formula Reconstruction

In this experiment, we test FORTE's ability to reconstruct a formula. Because some baselines only work on binary trees [12, 13], for this experiment we select a subset of 170k formulae from our dataset whose operator trees are binary.

[1]https://www.cs.rit.edu/~rlaz/NTCIR-12_MathIR_Wikipedia_Corpus.zip

[2]https://sigmathling.kwarc.info/resources/arxmliv-dataset-082019/

[3]https://nlp.stanford.edu/projects/myasu/topiceq/context_eq_data_20190220.zip

1) 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** [12] which is an RNN-based method capable of encoding and decoding only binary trees; **treeTransformer** [13] which is a Transformer-based method that shows success only on binary trees; **GVAE** [9] and **DVAE** [18], both of which are variational auto-encoders (VAEs) suitable for tree-structured data. More discussion on these baselines are in Section II-F. We also include four variants of our framework to evaluate the utility of i) binary against onehot tree positional encoding and ii) TBS against greedy search for tree generation.

2) Evaluation Metrics.: We use two groups of metrics. The first group of metrics measure formula reconstruction accuracy (ACC), i.e., the percentage of the decoder outputs that are exactly the same as the ground-truth decoder target formulae. We compute both **ACC-1**, using only the output formula with the highest likelihood, and **ACC-5**, using the five formulae with highest likelihood. Specifically, let $X_{\text{out}}^{(i)}$ be the i-th ground-truth output tree in the test set and $A = \{\widehat{X}_{\text{out}}^{(ij)}\}_{j=1}^{J}$ be the set of J generated trees for $X_{\text{out}}^{(i)}$. Then

$$\text{ACC} = \frac{1}{N_{\text{test}}} \sum_{i=1}^{N_{\text{test}}} \mathbf{1}_A(X_{\text{out}}) \,,$$

where

$$\mathbf{1}_A(X_{\text{out}}) = \begin{cases} 1 & \text{if } X_{\text{out}} \in A \\ 0 & \text{if } X_{\text{out}} \notin A \end{cases}$$

and N_{test} is the total number of formula trees in the test set. ACC-1 uses $J = 1$ and greedy search for generation whereas ACC-5 uses $J = 5$ and beam search for generation. For the seq2seqRNN baseline, X_{out} and $\widehat{X}_{\text{out}}$ are both in the format of a sequence of math symbols instead of a tree.

The second group of metrics measures how much the generated formula differs from the ground-truth decoder target formula under the tree format. We use *tree edit distance* (TED) which measures the distance between two trees by computing the minimum number of operations needed, including changing nodes and node connections, to convert one tree to the other. We implement the TED metric using the `apted` package [31] and refer to [32, 33] for an overview of TED. We compute both **TED-full** which considers both node and structural differences and **TED-Struct** which only considers structural differences. For the seq2seqRNN baseline that does not output formula in the OT format, we first use the formula tree parser [1] to convert generated formulae to the corresponding OT and then compute TED. Note that some generated formulae from the seq2seqRNN baseline may incur errors when we convert them to OTs. We do not count these cases in our TED computation, which gives an advantage to the seq2seqRNN baseline; nevertheless, seq2seqRNN still underperforms our method.

TABLE III: Formula reconstruction visualizations comparing FORTE with baselines using two input formulae (top row). Only FORTE succeeds in exactly reconstructing the input.

Methods	$\Theta^{A}_{\text{diff}} = \Theta_{\text{state}}$	$K \wedge \{\neg f \mid f \in F\}$
seq2seqRNN	$\Theta_{\text{diff}} = A_{V}^{\text{tharte}}$	$K \wedge \{\neg\ f\ f \in F\}$
tree2treeRNN	invalid formula	$\wedge \rightarrow \times\times \in$
tree2treeTF	$\Theta^{A}_{\text{diff}} = \Theta$	$(K \in \lambda) \wedge fF \cup (f \in R)$
FORTE	$\Theta^{A}_{\text{diff}} = \Theta_{\text{state}}$	$K \wedge \{\neg f \mid f \in F\}$

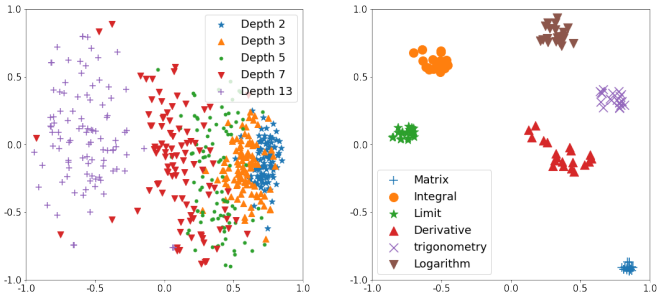


Fig. 5: T-SNE visualizations of FORTE formula embeddings for formulae of different tree structures (left) and different content (right). We see clear separation and clustering of different formulae.

3) Experiment Setup.: We construct training, validation, and test sets by randomly splitting the 170k dataset (recall beginning of Section III-B) 80%-10%-10% for five times. During training, we save the best performing model and parameters using the validation set and then perform formula reconstruction on the test. Whenever beam search is applicable, We use beam size $B = 10$. For the seq2seqRNN baseline and all FORTE variants, we use 500-dimensional hidden states and node embeddings, 2-layer GRUs, 96 batch size, and 50 training epochs. For the two tree2tree baselines [12, 13], GVAE [9], and DVAE [18], we follow the original configurations. All methods are trained on a single NVIDIA RTX 8000 GPU.

4) Quantitative Results.: Table II shows the formula reconstruction results, averaged over all five random data splits, comparing FORTE against various baselines. We observe that both GVAE and DVAE do not work well for this task likely because of a mismatch between their model designs and the data type for our task. For example, both GVAE and DVAE leverage rigid rules during generation, i.e., by specifying which node must connect to which node. Because the scientific formulae that we work with have very diverse structures, it is likely that these rules significantly constrain the generation, leading to unsatisfactory reconstruction. This mismatch between baseline designs and our data can also explain the unsatisfactory performance of the two tree2tree baselines. Specifically, these baselines are designed for computer program translation, where the tree structure is also much less varied than those for scientific formulae. The decoder design in these baselines also incorporates multiple constraints, e.g., by specifying the number of children each node must have. Such constraints likely limit the tree2tree baselines' ability to generate high-quality formula trees. In contrast, FORTE almost perfectly reconstructs complex formulae in our dataset, showing excellent robustness and flexibility.

We also see that FORTE outperforms the seq2seqRNN baseline. This observation is not surprising since FORTE exploits the inherent tree structure of scientific formulae whereas seq2seqRNN does not. Moreover, the results for the four FORTE variants clearly demonstrate the benefits of both binary tree positional embedding and TBS, leading to improvements in all four metrics compared to onehot tree positional embedding and greedy search, respectively. We repeat this experiment on the entire 770k formula dataset comparing only FORTE and seq2seqRNN since the tree-based baselines cannot process non-binary trees. FORTE achieves 85.87% compared to seq2seqRNN's 84.30% on TOP-1 ACC and 90.30% compared to seq2seqRNN's 88.52% on TOP-5 ACC, respectively. These results further validate the advantage of representing scientific formulae as OTs over as sequences.

5) Qualitative Results.: We first visualize in Figure 1 some simple formulae generated by passing randomly sampled embedding vectors, i.e., $\boldsymbol{e} \sim \mathcal{N}(0, \boldsymbol{I})$, through FORTE's decoder. Despite their simplicity, these examples show that our tree decoder can generate valid and diverse formulae.

We also visualize some reconstruction results in Table III on two input formulae, comparing FORTE against seq2seqRNN and the two tree2tree baselines. The ground-truth formulae are at the top of the table and each subsequent row contains the corresponding formulae reconstructed by each method. The two tree-based baselines can correctly generate part of or all symbols in the ground-truth formulae but sometimes in an incorrect order, resulting in formulae that are visually different and even invalid. This observation is likely caused by the absence of a clear termination signal for the generation of children of each parent node during training; since these baselines pre-specified the number of children for each node, they may not be able to properly learn the structural aspects of scientific formulae, which results in generating only partially valid or partially correct formula trees. The seq2seqRNN baseline generates most of the symbols correctly and in the right order but misses or misplaces certain symbols. This observation is likely caused by the loss of tree structural information when we treat a formula as a sequence of symbols, which may lead to incorrect reconstruction. In contrast, thanks to the clear termination signal at each level of the tree and the tree structures being preserved, FORTE perfectly reconstructs both input formulae.

Finally, we visualize FORTE's learned embedding space for formulae in Figure 5. We perform two sets of visualizations to examine whether FORTE has learned both the structure and content of formulae. For the first set, we sample formula trees of varying depth and for the second set, we sample formula trees that belong to 6 distinct subjects. We compute their embeddings and plot their 2-dimensional t-SNE embeddings [34]. From Figure 5, we can see clear separation and

TABLE IV: Examples of top 5 retrieval results comparing FORTE to TangentCFT. Less ideal retrieved formulae are in red.

	$O(mn\log m)$		$\cos\alpha = -\cos\beta\cos\gamma + \sin\beta\sin\gamma\cosh\frac{a}{k}$	
Rank	**FORTE**	**TangentCFT**	**FORTE**	**TangentCFT**
1	$O(mn\log m)$	$O(mn\log m)$	$\cos\alpha = -\cos\beta\cos\gamma + \sin\beta\sin\gamma\cosh\frac{a}{k}$	$\cos\alpha = -\cos\beta\cos\gamma + \sin\beta\sin\gamma\cosh\frac{a}{k}$
2	$O(n\log m)$	$O(n\log m)$	$\cos A = -\cos B\cos C + \sin B\sin C\cosh a$	$\cos(\alpha-\beta) = \cos\alpha\cos\beta + \sin\alpha\sin\beta$
3	$O(nk\log k)$	$O(nm)$	$\cos(A) = -\cos(B)\cos(C) + \sin(B)\sin(C)\cos(a)$	$a = \arccos\left(\frac{\cos\alpha+\cos\beta\cos\gamma}{\sin\beta\sin\gamma}\right)$
4	$O(nk\log((n))$	$O(n*m)$	$\cos A = -\cos B\cos C + \sin B\sin C\cos a$	$\cos A = -\cos B\cos C + \sin B\sin C\cosh a$
5	$O(n\log h)$	$O(mn)$	$\cos a = \cos b\cos c + \sin b\sin c\cos\alpha$	$\cos C = -\cos A\cos B + \sin A\sin B\cosh c$

clustering both for formulae with varying tree depths and for formulae with distinct content. Formulae with deeper trees are less clustered (left plot in Figure 5) likely because they are also more diverse. Overall, these two sets of visualizations further demonstrate that FORTE learns a meaningful embedding space for scientific formulae.

C. Formula Retrieval

In this experiment, we evaluate FORTE's capability in a formula retrieval task. Given a query formula (the query), a retrieval method returns the most related formulae (the retrievals) from a collection of candidate formulae.

1) Query and Retrieval Formulae Processing: We use the first 20 concrete queries (e.g., formula without unknown parts) in the NTCIR-12 Wikipedia math formula retrieval task. We remove one that is too simple (it contains only a single variable and nothing else) and one that incurs an error when being converted to OT. The resulting query set contains 18 queries. Table I shows a few example queries that we consider. The collection of candidate formulae for retrieval is the NTCIR-12 Wikipedia math formula dataset [2] which is a subset of our 770k formula dataset. Because some formulae in this subset are identical, i.e., formulae that appear in multiple Wikipedia articles, we remove duplicates before performing the retrieval experiment. We also remove formulae that incur errors when being converted to OTs. The resulting candidate formulae collection contains roughly 300k formulae. All of the 18 queries are present in this collection. The processing steps for both queries and candidate retrievals are consistent with that in Section III-A.

2) Baselines.: We consider three state-of-the-art baselines designed specifically for the formula retrieval task including **Tangent-CFT** [6], **Tangent-S** [1], and **Approach0** [3]. The first baseline is one of the few data-driven formula retrieval methods to date, while the latter two are based on symbolic sub-tree matching and are data-independent.

a) Evaluation Metrics.: We perform a "pooled" human evaluation for the formula retrieval experiment consistent with prior work [2]. First, for each method and each query, we choose the top 25 retrievals and mix them into a single pool for evaluation. Then, for each query and each retrieval, we ask human evaluators how relevant is the retrieval to the query. Possible judgment ratings are relevant, partially relevant, or irrelevant. To encourage fair and consistent evaluation, we first ask the evaluators to browse through all retrieval formulae for a given query before performing the evaluation. This step calibrates the evaluators' judgments. We also provide evaluators with the following evaluation guideline, quoted from [2]: "*A retrieval is considered* relevant *if both its appearance and the content of the formula match that of the query. If either the retrieval's appearance or content matches that of the query but not both, the retrieval is considered* partially relevant. *Otherwise, the retrieval is* irrelevant *to the query*". In total, three human evaluators are involved, each of whom provides us with his/her independent evaluations for each retrieval in the pool for each query.

TABLE V: Formula retrieval results.

	Metrics (partial)		**Metrics (full)**	
Methods	**map**	**bpref**	**map**	**bpref**
Approach0	0.404	**0.537**	0.486	0.507
Tangent-S	0.403	0.449	0.461	0.472
TangentCFT	0.418	0.471	0.462	0.464
FORTE	0.395	0.455	0.475	0.485
FORTE-App	**0.423**	0.484	**0.509**	**0.513**

We use mean average precision (**MAP**) [35] and **bpref** [36] as the evaluation metrics. They are computed by comparing the human evaluation of the pooled retrievals for each query with each method's top 1000 retrievals. 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 methods. We implement these metrics using a common information retrieval evaluation package [37] for both partially relevant and fully relevant retrievals.

3) Experiment Setup: We use the entire 770k formula dataset to train our FORTE framework and then use the trained encoder to obtain an embedding vector for each formula. For each query, we compute the cosine similarity between its embedding vector and the embedding vector of each formula in the dataset. Finally, we return the formulae with the highest similarity scores as the retrieved ones; See Section II-B2 for detailed computation. Because the retrieval results for each baselines is publicly available,[4,5,6] we do not rerun each baseline and simply use the provided retrieval results for our evaluation.

[4]https://github.com/BehroozMansouri/TangentCFT
[5]https://github.com/approach0/search-engine/tree/ecir2020
[6]https://www.cs.rit.edu/~dprl/files/release_tangent_S.zip

TABLE VI: Zero-shot formula reconstruction results (ACC) on the ARQMath dataset for methods trained on our dataset. FORTE generalizes well to the new dataset.

Methods	ACC-1 ↑	ACC-5 ↑
tree2treeRNN [12]	46.31%	-
tree2treeTF [13]	72.37%	-
seq2seqRNN	43.46%	52.66%
FORTE (binary, greedy)	89.16%	-
FORTE (binary, beam)	**89.51%**	
FORTE (onehot, greedy)	89.00%	-
FORTE (onehot, beam)	89.43%	94.44%

TABLE VII: Zero-shot formula retrieval results (bpref) on the ARQMath dataset. When combining FORTE with Approach0, we achieve the state-of-the-art retrieval performance.

Methods	Partial	Full	Harmonic mean
Approach0	0.477	0.325	0.386
Tangent-S	0.441	0.251	0.320
TangentCFT	0.437	0.305	0.359
FORTE	0.409	0.308	0.351
FORTE-App	**0.502**	**0.328**	**0.398**

4) Quantitative Results.: Table V shows the quantitative evaluation results, averaged over the three evaluators' scores. We observe that FORTE performs well for the fully relevant retrieval evaluation, outperforming both Tangent-S and the data-driven method, TangentCFT. On partially relevant retrieval evaluation, FORTE sometimes falls behind the other baselines. The reason is that, unlike TangentCFT that embeds linearized, sub-components of a formula tree, FORTE focuses on the full tree structure. Therefore, a tree with similar sub-tree components to another tree may differ significantly in their overall structures and get a retrieval score higher from TangentCFT than from FORTE. In addition, unlike Approach0 (and Tangent-S) that directly computes similarity using the symbolic sub-tree components, FORTE (and TangentCFT) computes cosine similarity on the much more abstract formula embeddings, which may cause loss of information compared to explicit symbolic computations used in Approach0. Therefore, following [6], we use a linear combination of the retrieval scores by FORTE and Approach0 as a new retrieval method, which we dub **FORTE-App**, that combine the advantage of both methods. This method achieves state-of-the-art retrieval performance on three of the four metrics.

5) Qualitative Results.: Table IV shows a few qualitative examples comparing formulae retrieved by FORTE to those retrieved by the TangentCFT for the same query. For the first query, all formulae retrieved by FORTE either contain $\log$ or are in the form of $O(\text{variable} \times \text{variable } \log \text{ variable})$, which is the same as that for the query. Similarly, for the second query, all formulae retrieved by FORTE are mostly the same as the query except for a few variables, signs, and functions (e.g., the last $\cos$ function in the 3rd–5th ranked formulae) differences. These examples illustrate FORTE's advantage over baselines in preserving the structure of the query formula and the semantic meaning of symbols in the formula.

D. Zero-Shot Generalization

We also validate FORTE using a recently released formula dataset, ARQMath,[7] where formulae are collected from Math Stack Exchange,[8] a domain where most formulae are math equations that are very different from those in scientific documents. Specifically, we test the zero-shot generalizability of FORTE (after training on our dataset) to the ARQMath dataset without further fine-tuning.

Table VI reports the formula reconstruction performance with respect to the ACC-1 and ACC-5 metrics and compares FORTE with the baselines. We see that that performances for all methods drop compared to Table II. Nevertheless, FORTE still performs well and significantly better than the baselines. In addition, using binary positional encoding with tree beam search still achieves the best performance among different settings, which is consistent with previous results. Table VII reports the formula retrieval performance with respect to the bpref metric comparing FORTE with the baselines. We see that the performance of data-driven methods, including TangentCFT and FORTE, slightly drops without training or fine-tuning on the new dataset. In contrast, the performance of the best data-agnostic method, Approach0, does not drop. These comparisons suggest that improving FORTE's generalizability is an important future research direction. When combining FORTE and Approach0, we achieve the state-of-the-art retrieval performance on the ARQMath dataset, which is consistent with the observation in Table V. This result suggests that combining both data-driven and non-data-driven methods is a promising approach for formula retrieval.

IV. Conclusions and Future Work

In this work, we propose FORTE, a novel, unsupervised scientific formula 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 the formula reconstruction and the formula retrieval tasks and demonstrate our framework's superior performance in both experiments compared to baselines.

Our work opens doors to many future avenues of research. 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 several existing works to potentially improve performance, i.e., in [7] for joint text and math retrieval, in [8] for math headline generation, in [38, 39] for grading students' math homework solutions and providing feedback, and in [27, 28] for neural math reasoning.

[7]https://www.cs.rit.edu/~dprl/ARQMath/

[8]https://math.stackexchange.com/

ACKNOWLEDGEMENTS

This work is supported by NSF grants 1842378, 1937134, 1917713, 2118706, ONR grant N0014-20-1-2534, AFOSR grant FA9550-18-1-0478, and a Vannevar Bush Faculty Fellowship, ONR grant N00014-18-1-2047.

REFERENCES

[1] K. Davila and R. Zanibbi, "Layout and semantics: Combining representations for mathematical formula search," in *Prof. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval*, 2017, p. 1165–1168.

[2] R. Zanibbi, A. Aizawa, M. Kohlhase, I. Ounis, G. Topic, and K. Davila, "Ntcir-12 mathir task overview." in *Proc. NTCIR Conf. Eval. Info. Access*, 2016.

[3] W. Zhong and R. Zanibbi, "Structural similarity search for formulas using leaf-root paths in operator subtrees," in *Proc. Intl. Conf. Neural Info. Process. Syst.*, L. Azzopardi, B. Stein, N. Fuhr, P. Mayr, C. Hauff, and D. Hiemstra, Eds., 2019, pp. 116–129.

[4] W. Zhong, S. Rohatgi, J. Wu, C. Giles, and R. Zanibbi, "Accelerating substructure similarity search for formula retrieval," in *Proc. European Conf. Info. Retrieval*, 2020, pp. 714–727.

[5] L. Gao, Z. Jiang, Y. Yin, K. Yuan, Z. Yan, and Z. Tang, "Preliminary Exploration of Formula Embedding for Mathematical Information Retrieval: can mathematical formulae be embedded like a natural language?" *arXiv e-prints*, Jul. 2017.

[6] B. Mansouri, S. Rohatgi, D. W. Oard, J. Wu, C. L. Giles, and R. Zanibbi, "Tangent-cft: An embedding model for mathematical formulas," in *Proc. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval*, 2019, p. 11–18.

[7] M. Yasunaga and J. Lafferty, "TopicEq: A Joint Topic and Mathematical Equation Model for Scientific Texts," in *Proc. AAAI conf. Artificial Intell.*, 2019.

[8] K. Yuan, D. He, Z. Jiang, L. Gao, Z. Tang, and C. L. Giles, "Automatic generation of headlines for online math questions," in *Proc. AAAI conf. Artificial Intell.*, 2020, pp. 9490–9497.

[9] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, "Grammar variational autoencoder," in *Proc. Int. Conf. Mach. Learn.*, vol. 70, 06–11 Aug 2017, pp. 1945–1954.

[10] R. Zanibbi and D. Blostein, "Recognition and retrieval of mathematical expressions," *Intl. J. Document Anal. Recognit.*, vol. 15, no. 4, pp. 331–357, Dec 2012.

[11] K. Kunen, *Set theory - an introduction to independence proofs*. North-Holland, 1983, vol. 102.

[12] X. Chen, C. Liu, and D. Song, "Tree-to-tree neural networks for program translation," in *Proc. Intl. Conf. Neural Info. Process. Syst.*, 2018, p. 2552–2562.

[13] V. Shiv and C. Quirk, "Novel positional encodings to enable tree-based transformers," in *Proc. Intl. Conf. Neural Info. Process. Syst.*, 2019, pp. 12 081–12 091.

[14] Y. Wang, H.-Y. Lee, and Y.-N. Chen, "Tree transformer: Integrating tree structures into self-attention," in *Proc. Conf. Empirical Methods Natural Lang. Process. and Intl. Joint Conf. Natural Lang. Process.*, 2019, pp. 1061–1070.

[15] X.-P. Nguyen, S. Joty, S. Hoi, and R. Socher, "Tree-structured attention with hierarchical accumulation," in *Proc. Intl. Conf. Learn. Representations*, 2020.

[16] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Oct. 2014, pp. 1724–1734.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. Intl. Conf. Neural Info. Process. Syst.*, 2014, p. 3104–3112.

[18] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen, "D-vae: A variational autoencoder for directed acyclic graphs," in *Proc. Intl. Conf. Neural Info. Process. Syst.*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32, 2019.

[19] K. S. Tai, R. Socher, and C. D. Manning, "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks," *arXiv e-prints*, Feb. 2015.

[20] Y. Shen, S. Tan, A. Sordoni, and A. Courville, "Ordered neurons: Integrating tree structures into recurrent neural networks," in *Proc. Intl. Conf. Learn. Representations*, 2019.

[21] Y. Wang, H.-Y. Lee, and Y.-N. Chen, "Tree transformer: Integrating tree structures into self-attention," in *Proc. Conf. Empirical Methods Natural Lang. Process. and Intl. Joint Conf. Natural Lang. Process.*, Nov. 2019, pp. 1061–1070.

[22] J. Sun, P. Han, Z. Cheng, E. Wu, and W. Wang, "Transformer based multi-grained attention network for aspect-based sentiment analysis," *IEEE Access*, vol. 8, pp. 211 152–211 163, 2020.

[23] Y. Goldberg and G. Hirst, *Neural Network Methods in Natural Language Processing*. Morgan & Claypool Publishers, 2017.

[24] P. Koehn, "Pharaoh: A beam search decoder for phrase-based statistical machine translation models," in *Mach. Transl.: Real Users Res.*, R. E. Frederking and K. B. Taylor, Eds., 2004, pp. 115–124.

[25] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," in *Proc. First Workshop Neural Mach. Transl.*, Aug. 2017, pp. 56–60.

[26] J. Zhuo, Z. Xu, W. Dai, H. Zhu, H. Li, J. Xu, and K. Gai, "Learning optimal tree models under beam search," in *Proc. Intl. Conf. Mach. Learn.*, vol. 119, 13–18 Jul 2020, pp. 11 650–11 659.

[27] G. Lample and F. Charton, "Deep learning for symbolic mathematics," in *Proc. Intl. Conf. Learn. Representations*, 2020.

[28] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli, "Analysing mathematical reasoning abilities of neural models," in *Proc. Intl. Conf. Learn. Representations*, 2019.

[29] K. Yuan, D. He, Z. Jiang, L. Gao, Z. Tang, and C. L. Giles, "Automatic Generation of Headlines for Online Math Questions," *arXiv e-prints*, Nov. 2019.

[30] Y. Deng, A. Kanervisto, J. Ling, and A. M. Rush, "Image-to-markup generation with coarse-to-fine attention," in *Proc. Intl. Conf. Mach. Learn.*, 2017, p. 980–989.

[31] J. F. Pimentel, "Python apted algorithm for the tree edit distance," https://github.com/JoaoFelipe/apted, 2017.

[32] M. Pawlik and N. Augsten, "Tree edit distance: Robust and memory-efficient," *Info. Syst.*, vol. 56, pp. 157 – 173, 2016.

[33] P. Mateusz and A. Nikolaus, "Efficient computation of the tree edit distance," *ACM Trans. Database Syst.*, vol. 40, no. 1, Mar. 2015.

[34] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[35] E. M. Voorhees, D. K. Harman *et al.*, *TREC: Experiment and evaluation in information retrieval*. MIT press Cambridge, 2005, vol. 63.

[36] C. Buckley and E. M. Voorhees, "Retrieval evaluation with incomplete information," in *Prof. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval*, 2004, p. 25–32.

[37] C. Buckley, "trec_eval," https://github.com/usnistgov/trec_eval, 2008.

[38] A. S. Lan, D. Vats, A. E. Waters, and R. G. Baraniuk, "Mathematical language processing: Automatic grading and feedback for open response mathematical questions," in *Proc. ACM Conf. Learn. @ Scale*, 2015, p. 167–176.

[39] M. Zhang, Z. Wang, R. Baraniuk, and A. Lan, "Math operation embeddings for open-ended solution analysis and feedback," *Proc. Intl. Conf. Educ. Data Min.*, pp. 216–227, June 2021.