Mathematical Formula Representation via Tree Embeddings

Zichao Wang\textsuperscript{1}, Andrew Lan\textsuperscript{2}, Richard Baraniuk\textsuperscript{1}

\textsuperscript{1}Rice University
\textsuperscript{2}University of Massachusetts Amherst

Abstract
We propose a new framework for learning mathematical formula representations using tree embeddings. 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. With its unique set of symbols and language structure, mathematical language complements natural language in concisely and precisely communicating essential scientific knowledge. It is now becoming an indispensable part of an ever-growing body of scientific content.

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., [Davila and Zanibbi, 2017]). This task arises in a wide range of scenarios, such as when researchers search for formulae in a large collection of papers or 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 [Yuan \textit{et al.}, 2020]. 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.

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., [Tai \textit{et al.}, 2015; Shen \textit{et al.}, 2019; Wang \textit{et al.}, 2019; Nguyen \textit{et al.}, 2020; Sun \textit{et al.}, 2020]), a number of recent works in formula retrieval exploit formula’s unique structural properties, leading to improved results [Davila and Zanibbi, 2017; Zhong and Zanibbi, 2019; Mansouri \textit{et al.}, 2019; Zhong \textit{et al.}, 2020] compared to other formula representations, e.g., [Gao \textit{et al.}, 2017]. 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, [Yasunaga and Lafferty, 2019] trains a topic model on scientific documents and learns the keywords (topics) of a formula. [Yuan \textit{et al.}, 2020] 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 [Lample and Charton, 2020; Saxton \textit{et al.}, 2019]. 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 de-

\begin{table}[h]
\centering
\begin{tabular}{ll}
\hline
Frequency of this item & Frequency of most common item \\
\hline
\text{\footnotesize{\textsuperscript{(physics)}}} & \text{\footnotesize{\textsuperscript{(physics)}}} \\
\text{\footnotesize{\textsuperscript{(chemistry)}}} & \text{\footnotesize{\textsuperscript{(chemistry)}}} \\
\text{\footnotesize{\textsuperscript{(algebra)}}} & \text{\footnotesize{\textsuperscript{(algebra)}}} \\
\hline
\end{tabular}
\caption{A few examples of mathematical language in our context.}
\end{table}
sign 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 FORmula Representation learning via Tree Embeddings. 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. 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., [Sutskever et al., 2014; Bahdanau et al., 2014]) 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 1–3 together provide a high-level overview of our framework.

### 2.1 Formulae As Operator Trees

Every formula $X$ is inherently tree structured [Zanibbi and Blostein, 2012; Davila and Zanibbi, 2017] and can be represented as a symbolic operator tree (OT):

$$X = (U, <), \quad u \in U, \quad U \subseteq V$$

where $u$ is a math symbol (a node in OT), $U$ 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 $\forall u \in U$ [Kunen, 1983]. 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 [Davila and Zanibbi, 2017] 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 $V_{\text{num}}$ (numbers, decimals), functional $V_{\text{fun}}$ (multiplication, subtraction etc.), variable $V_{\text{var}}$, textual $V_{\text{txt}}$ 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 [Goodfellow et al., 2016]), 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

$$L(\theta, \phi) = \frac{1}{N} \sum_{i=1}^{N} P_x(x^{(i)}_{\text{out}}) \log f_d(f_e(x^{(i)}_{\text{in}}; \theta); \phi),$$

where $N$ is the number of formulae, $f_e$ is the encoder function with parameter $\theta$, $f_d$ is the decoder function with parameter $\phi$ and $P_x$ is the empirical data distribution. $X^{(i)}_{\text{in}}$ and $X^{(i)}_{\text{out}}$ 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
The decoding process.

(a) Input and output formula tree.

encoder input tree  decoder target tree

(b) The decoding process.

Figure 2: (2a) Illustration of FORTE’s input and output operator tree of the same formula. The “E” nodes represent 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.

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 traversing 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 \( \tilde{x}_t \in \mathbb{R}^M \) 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 \( \omega_t \in \mathbb{R}^t \) where \( \ell_t = 0, \ldots, T \) is the depth of node \( u \) in the tree. \( \omega_t \) is composed of the node’s parent’s position appended with its relative positions to its siblings. \( \omega_0 = [0] \) 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 \( \omega_t \) of each node to a fixed-dimensional vector \( p_t \in \mathbb{R}^D \) where

\[
p_t \left[ \lfloor \log_2(C) \rfloor j \ldots \lfloor \log_2(C) \rfloor j + \lfloor \log_2(q_t[j]) \rfloor \right] = \text{bin}(q_t[j]) \quad \forall j = 0, \ldots, \ell_t.
\]

\( C \) is the maximum degree (i.e., number of children) of all trees in the dataset, \( \lfloor \cdot \rfloor \) is the ceiling function, \( \text{bin}(\cdot) \) is the binarization operator (e.g., \( \text{bin}(5) = 101 \)) and \( p_t[j] \) selects the \( j \)-th index of the vector \( p_t \). The resulting dimension of the tree positional embedding \( p_t \) 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 \( f_e : \mathbb{R}^{(M+D) \times T} \rightarrow \mathbb{R}^K \) 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

\[
h = f_e(\{x_t\}_{t=1}^T; \theta), \quad x_t = [\tilde{x}_t^T, p_t^T]^T. \tag{4}
\]

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) [Cho et al., 2014] 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 [Tai et al., 2015; Chen et al., 2018], 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 [Shiv and Quirk, 2019], which uses a onehot-style tree positional embedding, our encoder employs a different binary tree positional embedding which reduces the space complexity from \( \mathcal{O}(L^2C) \) 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 method.
Concretely, the next node is generated as
\[ \tilde{u}_t+1 = \arg\max_{u \in V} \text{softmax}(f_u(\{x'_s\}_{s=1}^t; \phi)), \]
where we also concatenate the formula tree embedding \( h \) to the decoder’s input at each generation step. The next node’s position \( p_{t+1} \) 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 suboptimal. 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. Figure 3 illustrates the TBS generation process. The full algorithm is available in the supplementary material.

TBS extends beam search in NLP (e.g., in [Sutskever et al., 2014; Bahdanau et al., 2014]) which only concerns sequential data and is incapable of generating tree-structured data. Similar to beam search in NLP, by expanding 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 [Zhuo et al., 2020] 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 [Shiv and Quirk, 2019] 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. Further details on the dataset are available in the supplementary material. We plan to open-source the dataset upon publication.
3.1 Formula Reconstruction

In this experiment, we test FORTE’s ability to reconstruct a formula. Because some baselines only work on binary trees [Chen et al., 2018; Shiv and Quirk, 2019], 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 [Chen et al., 2018] which is an RNN-based method capable of encoding and decoding only binary trees; treeTransformer [Shiv and Quirk, 2019] 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 [Pawlik and Augusten, 2016; Pawlik and Augusten, 2015] 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. The supplementary material includes additional details on the experiment setup.

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.

We visualize some reconstruction results in Table 3 on two input formulae. The two tree-based baselines are able to correctly generate part of or all symbols in the ground-truth but sometimes in incorrect order, resulting in visually very different rendered formulae and even invalid ones. The seq2seqRNN baseline generates most of the symbols correctly and in the right order but misses or misplaces certain symbols. In contrast, FORTE perfectly reconstructs both input formulae. We also visualize in Fig. 4 some simple formula trees randomly sampled from our tree decoder given random input, i.e., $h \sim \mathcal{N}(0, \mathbf{I})$. Despite their simplicity, these examples show that our tree decoder can generate valid
and varied formulae. More examples analysis, and comparisons are available in the supplementary material.

### 3.2 Formula Retrieval

In this 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, 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 [Zanibbi et al., 2016]. 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 [Mansouri et al., 2019], which is one of the few data-driven formula retrieval systems to date, and Tangent-S [Davila and Zanibbi, 2017] and Approach0 [Zhong and Zanibbi, 2019], 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, two 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 [Zanibbi et al., 2016], including the number of evaluators involved. We then use the mean average precision (MAP) [Voorhees et al., 2005] and bpref [Buckley and Voorhees, 2004] 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. The supplementary material includes further details on the experiment setup and evaluation procedures.

**Results.** Table 5 presents the quantitative evaluation results, averaged over the two evaluators’ scores.² Both metrics indicate that our framework clearly achieves better retrieval performance than all baselines. 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.

Table 4 provides a few qualitative examples of the retrieval results comparing FORTE to TangentCFT. For the first query, all retrievals from FORTE either contain log or are in the form of $O(\text{variable} \times \text{variable log variable})$, which is the same as the query. Similarly, for the second query, all retrievals from FORTE are the same as the query except for a few variable, sign, and function (e.g., the last $\cos$ function in the 3rd–5th ranked retrieval) changes. These examples illustrate FORTE’s capability in better preserving the structure of the query formula and the semantics of the symbols than 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 the formula reconstruction and the formula retrieval tasks and demonstrate our framework’s superior performance in both experiments compared to baselines.

Our work provides promising preliminary results and 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 a number of existing works to potentially improve performance, i.e., in [Yasunaga and Lafferty, 2019] for joint text and math retrieval, in [Yuan et al., 2020] for math headline generation, in [Lan et al., 2015] for grading students’ math homework solutions, and in [Saxton et al., 2019; Lample and Charton, 2020] for neural math reasoning.

²The Cohen’s Kappa score is 69%, demonstrating substantial agreement between two evaluators’ judgements [McHugh, 2012].
References


A Math Symbol Vocabulary

The conversion from infrequent nodes in each sub-vocabulary to the “unknown” node is as follows. For infrequent $u \in V_{num}$, we convert them to either a “decimal number” node (e.g., 0.123) or “integer number” node (e.g., 123). For infrequent $u \in V_{var}$, we convert them to either a “one character variable” node (i.e., $x$) or “multi-character variable” node (i.e., the symbol $Fe$ for iron in chemistry). For infrequent nodes $u \in V_{txt}$ and $u \in V_{fun}$, we convert them to an “unknown text” and an “unknown other” node, respectively. We retain all $u \in V_{fun}$ because this set is small and often $K > |V_{fun}|$ for practical values of $K$.

B Encoder Design

Because we employ a GRU in our work and because our encoder needs to output a fixed-dimensional embedding (see Eq. 4), we use a simple self-attention layer to achieve this. Concretely, let $h_t$ be the hidden state of the last GRU layer for each node $u_t$, we have:

$$h = \sum_{t=1}^{T} a_t h_t, \quad a_t = \sigma(w^T h_t + b),$$

where $a_t$ is the attention weight, $\sigma(\cdot)$ is the sigmoid function, and $w \in \mathbb{R}^K$ and $b \in \mathbb{R}$ are parameters of the self-attention layer. In general, many other attention mechanisms can also be used, as long as the encoder output is a single embedding.

C Tree Generation Algorithm

Algorithms 2 and 4 provide a high-level pseudo-code for both the basic, greedy search (GS) tree generation algorithm and the tree beam search (TBS) generation algorithm. The main idea of TBS is mainly the same as GS but involves a few technicalities. Note that we only provide an overview of these algorithms. Implementation details and corner cases such as additional termination conditions are not covered. In our experiments, we set beam size $B = 10$ to trade off computational efficiency and generation quality. While TBS takes about 5 more minutes on the entire test set than GS (20min vs. 15min), TBS achieves substantially better generation quality than GS (see Table 2). We use a batch size of 1 during generation and therefore a larger batch size is likely to further reduce the computation time.

D Dataset Preprocessing

The formulae in the raw source are in either MathML or LATEX format. The raw source consists of the Wikipedia math article dataset and ArXiv article dataset from the NTCIR-12 math formula retrieval task [Zanibbi et al., 2016] (in MathML format) and the preprocessed ArXiv dataset from [Yasunaga and Lafferty, 2019] (in LATEX format).

We employ a parser [Davila and Zanibbi, 2017] to convert them to the same operator tree format. We keep formulae that do not incur any errors during the parsing process and whose maximum depth is below 20, the maximum degree is below 10 and the maximum number of nodes is below 250. These thresholds are primarily for removing formulae with extremely complicated operator tree structure that sig-
Algorithm 3: Beam Search Step

Require: Decoder function \( f_d \)
Input : Already generated nodes \( \{ u_{t,b} \}_{b=1}^B \) and positions \( \{ q_{t+1,b} \}_{b=1}^B \)
Output : Next nodes \( \{ u_{t+1,b} \}_{b=1}^B \) and selected stacks \( \{ S_{t,b} \}_{b=1}^B \)

for \( b = 1, \ldots, B \) do
  Generate a set of \( B \) next nodes \( U_b = \{ u_{t+1,1}, \ldots, u_{t+1,B} \} \);
  From \( U_b, \ldots, U_B \) select the \( B \) most probable nodes \( u_{t+1,1}, \ldots, u_{t+1,B} \) and the corresponding stack \( S_t, \ldots, S_B \) for each node;
  Return \( u_{t+1,1}, \ldots, u_{t+1,B} \) and \( S_t, \ldots, S_B \);

Algorithm 4: Tree Beam Generation

Require: Decoder \( f_d \), maximum generation step \( T \), beam size \( B \)
Input : (node, position) tuples \( F = \{ (U_b, Q_b) \}_{b=1}^B \) where \( U_b = \{ t_1, \ldots, t_T \}_{t=1}^T \) and \( Q_b = \{ q_b \}_{t=1}^T \)
Output: \( (node, position) \) tuples \( F = \{ (U_b, Q_b) \}_{b=1}^B \)

Initialize stacks \( S_1, \ldots, S_B \);
Initialize \( U_1, \ldots, U_B, Q_1, \ldots, Q_B \);
u_0 = \text{"start"}, q_0 = [0];
Generate next \( B \) nodes \( u_1, \ldots, u_{1,B} \);
Add \( u_{1,B} \) to \( U_B \) and \( q_{1,B} \) to \( Q_B \) for \( b = [1, B] \);
for \( b = 1, \ldots, B \) do
  if \( u_{1,b} \) is not \"end\" then
    Initialize struct \( A_{1b} \);
    \( A_{1b}.u = u_{1,b}, A_{1b}.n = 0, A_{1b}.q = q_{1,b}; \)
    Push \( A_{1b} \) onto \( S_1 \);
    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 \( S_b \) via Algo. 3;
    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 \( S_b \) via Algo. 1;
        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 \( F \);

significantly slows down the training and evaluation process. For the seq2seqRNN baseline, we use the BPEXrepresentation of formulae and use the tokenizer in [Deng et al., 2017] to process each formula into a sequence of math symbols.

E Formula Autoencoding Experiment

Setups of different methods and Baselines. Table A includes additional details on the model and baselines setup.
ing and additional experiments but we leave improving baselines for future work. All data-dependent methods are trained on a single NVIDIA RTX 8000 GPU with a batch size of 96 (except for the tree2treeRNN baseline which only trains with a batch size of 1). The largest memory consumption is less than 2 GB. Our method consumes roughly 1.3 GB during training. The training time for all methods takes about 5 minutes for an epoch except for the tree2treeRNN baseline, which takes about an hour.

**Metrics.** We compute the accuracy (ACC) metrics as follows. Let $X_{\text{out}}^{(i)}$ be the $i$-th ground-truth operator tree in the test set and $A = \{X_{\text{out}}^{(i)}\}_{j=1}^{J}$ be the set of $J$ generated operator trees for $X^{(i)}$. Then

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

(7)

where

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

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

We compute the tree edit distance (TED) metrics using the algorithm described in [Pawlik and Augsten, 2016; Pawlik and Augsten, 2015]. For the seq2seqRNN baseline that does not output formulae in tree formats, we first use the formula tree parser [Davila and Zanibbi, 2017] to convert generated formulae to their operator trees and then compute TED. Note that some generated formulae from the seq2seqRNN baseline incur error when converted to operator trees. In these cases, we simply remove this generation from TED computation. Doing so gives an advantage to the seq2seqRNN baseline because undesired generations are ignored. Nevertheless, from Table 2 we see that, even given such advantage, seq2seqRNN still underperforms our proposed FORTE framework.

**Visualization Procedures.** The tree output format makes it difficult to visualize. We thus manually examine each generated formula OT and convert it to $\text{LATEX}$ format whenever possible for easier interpretation. Note that the generated tree is not guaranteed to be a formula and sometimes cannot be converted back to $\text{LATEX}$ format; therefore, we simply write “invalid formula” when such a situation occurs. For visualizing the generated tree, we remove the “end” nodes for cleaner figures.

**Additional Generation Examples.** Figure A presents additional examples of generated operator trees and their corresponding rendered formulae from the decoder in FORTE.
top right is the rendered formula. The seq2seqRNN baseline generates most of the symbols correctly but makes mistakes towards the end of the sequence. The tree2treeRNN baseline can generate all symbols correctly but in the wrong order, which makes the formula OT invalid and makes it impossible to render. The tree2treeTF baseline generates the most symbols correctly and in the right order, but fails to do so towards the end of the generation. This example illustrates the baselines’ incapability to generate the entire formula correctly, even they succeed in doing so for the most part of the generation process. In contrast, only FORTE correctly generates the entire formula OT and in the right order.

Table D: Additional examples of top 5 retrieval results comparing FORTE to TangentCFT. We mark less ideal retrieved formulae in red.

<table>
<thead>
<tr>
<th>Rank</th>
<th>FORTE</th>
<th>TangentCFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( w = \begin{cases} w^* &amp; \text{if } w^* &gt; \frac{1}{2} \ \frac{1}{2} &amp; \text{if } w^* \leq \frac{1}{2} \end{cases} )</td>
<td>( w = \begin{cases} w^* &amp; \text{if } w^* &gt; 1 \ \frac{1}{2} &amp; \text{if } w^* \leq \frac{1}{2} \end{cases} )</td>
</tr>
<tr>
<td>2</td>
<td>( Y = \begin{cases} 0 &amp; \text{if } Y^* &gt; 0 \ 1 &amp; \text{if } Y^* &lt; 0 \end{cases} )</td>
<td>( a_i = \begin{cases} 1 &amp; \text{if } w_i &gt; 0 \ 0 &amp; \text{if } w_i &lt; 0 \end{cases} )</td>
</tr>
<tr>
<td>3</td>
<td>( y_i = \begin{cases} g_i^+ &amp; \text{if } g_i^+ &gt; g_i \ g_i^- &amp; \text{if } g_i^+ \leq g_i \end{cases} )</td>
<td>( f(x) = \begin{cases} \exp(-1/x) &amp; \text{if } x &gt; 0 \ 0 &amp; \text{if } x \leq 0 \end{cases} )</td>
</tr>
<tr>
<td>4</td>
<td>( a_i = \begin{cases} 1 &amp; \text{if } w_i &gt; 0 \ 0 &amp; \text{if } w_i &lt; 0 \end{cases} )</td>
<td>( rect(t) = \Pi(t) = \begin{cases} 0 &amp; \text{if }</td>
</tr>
<tr>
<td>5</td>
<td>( y_i = \begin{cases} g_i^+ &amp; \text{if } g_i^+ &gt; g_i \ 0 &amp; \text{if } g_i^+ \leq g_i \end{cases} )</td>
<td>( f(x) = \begin{cases} \frac{1}{2} &amp; \text{if } x &gt; 0 \ \frac{5}{8} &amp; \text{if } x \leq 0 \end{cases} )</td>
</tr>
</tbody>
</table>

Table E: Detailed formula retrieval results from both evaluators.

<table>
<thead>
<tr>
<th>Method</th>
<th>Metrics (Evaluator #1)</th>
<th>Metrics (Evaluator #2)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>map</td>
<td>bpref</td>
</tr>
<tr>
<td>Approach0</td>
<td>0.1286</td>
<td>0.1445</td>
</tr>
<tr>
<td>Tangent-S</td>
<td>0.4901</td>
<td>0.5258</td>
</tr>
<tr>
<td>TangentCFT</td>
<td>0.4942</td>
<td>0.5184</td>
</tr>
<tr>
<td>FORTE</td>
<td>0.5204</td>
<td>0.5374</td>
</tr>
</tbody>
</table>

Setups of Different Methods and Baselines. Because the Tangent-S and approach0 baselines are data-independent, there is no training involved. For fair comparison and consistency with our proposed FORTE framework, the Tangent-S and Tangent-CFT baselines both use the operator tree representations of formulae. For all baselines, we closely follow the steps in their respective paper and open-sourced implementations.

We develop a simple graphical interface (see Fig. B) for the human evaluators to perform the evaluation. The query and one retrieved formula are shown on the top and middle of the interface, respectively. The bottom of the interface contains a user input box where the evaluator inputs his/her judgment. After a retrieved formula is evaluated, the next retrieved formula shows up automatically, replacing the previous one. When all retrieved formulae are evaluated, the interface is closed, and a JSON file with the evaluation results is saved. The evaluator will then open a new interface to evaluate a different query.

Whether a retrieved formula is relevant to a query may be highly subjective. To encourage fair and consistent evaluation, we first ask the evaluators to go through all retrieved formulae for a given query. This step calibrates the evaluators’ judgments. We also provide evaluators with the following evaluation guideline, quoted from [Zanibbi et al., 2016]: “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 half-relevant. Otherwise, the retrieval is irrelevant to the query.”

To compute MAP and bpref reported in the main text, we
first compute both metrics for each evaluator using his/her evaluation results. Then we average the MAP and bpref scores, respectively, from both evaluators.

**Additional Quantitative Results.** Table E presents the MAP and bpref scores from both evaluators, before averaged. We can see that FORTE outperforms all baselines for both metrics and both evaluators.

**Additional Retrieval Examples.** Table D presents additional retrieval results comparing FORTE with Tangent-CFT, which is the best performing baseline. Similar to the results in the main text, here we observe that FORTE is capable of retrieving formulae that are structurally and semantically more similar to the query than Tangent-CFT.