Tree-Based Representation and Generation of Natural and Mathematical Language

Anonymous ACL submission

Abstract

Mathematical language in scientific communications and educational scenarios is important yet relatively understudied compared to natural languages. Recent works on mathematical language focus either on representing stand-alone mathematical expressions, especially in their natural tree format, or mathematical reasoning in pre-trained natural language models. Existing works on jointly modeling and generating natural and mathematical languages simply treat mathematical expressions as text, without accounting for the rigid structural properties of mathematical expressions. In this paper, we propose a series of modifications to existing language models to jointly represent and generate text and math: representing mathematical expressions as sequences of node tokens in their operator tree format, using math symbol and tree position embeddings to preserve the semantic and structural properties of mathematical expressions, and using a constrained decoding method to generate mathematically valid expressions. We ground our modifications in GPT-2, resulting in a model MathGPT, and demonstrate that it outperforms baselines on mathematical expression generation tasks.

1 Introduction

A part of human communication is performed in rigorous mathematical language rather than more flexible natural language, which often occurs in scenarios such as scientific communication and education. While pre-trained large language models such as BERT (Devlin et al., 2018) and GPT-3 (Brown et al., 2020) have enjoyed many successes in representing and generating natural language, there is a need for models that are effective in representing and generating principled mathematical language as well. While existing work focuses on various aspects of mathematical language representation or generation, combining mathematical language with the aforementioned models for natural language remains a challenging problem.

Mathematical and natural language are fundamentally different in many ways. While natural language consists of large sets of words and phrases that often have their meaning grounded in context, mathematical language consists of different symbols: a small set of mathematical operators with precise meaning, variables, and numbers that exist in a continuous space. Furthermore, mathematical language follows rules that are much more strict and rigorous than natural language. For example, the multiplication operation acts on exactly two operands, while an integral operates on a single operand but with upper and lower limit arguments. Operands are either variables, numbers, or the result of applying other operations. Given its hierarchical property, mathematical language is naturally represented with operator trees (OPTs), which are directed tree graphs where non-leaf nodes are operators and leaf nodes are variables or numbers (Zanibbi et al., 2016a; Mansouri et al., 2019). OPTs are effective at capturing both the semantic and structural properties of mathematical expressions.

Existing work primarily focuses on two separate approaches to modeling mathematical language: representation and mathematical reasoning. A line of work focuses on learning meaningful representations of mathematical expressions (often formulas), such as Wang et al. (2021); Davila and Zanibbi (2017), motivated by the task of retrieving similar expressions, which is especially relevant in information search and retrieval. Although these methods produce dense representations of expressions in their natural tree form, they cannot be directly connected to natural language. Some works employ BERT-like models to jointly represent natural and mathematical language (Liang et al., 2022; Peng et al., 2021; Shen et al., 2021). However, these methods are not well suited for generation tasks.

Another line of work focuses on mathematical reasoning, motivated by the task of mathematical problem solving that is especially relevant in educa-

tional applications. These works treat problem solving as a sequence-to-sequence task (Saxton et al., 2019a) and have found success on solving word problems (Huang et al., 2018; Zou and Lu, 2019). State-of-the-art methods use pre-trained large language models (Cobbe et al., 2021; Lewkowycz et al., 2022) and can even generate meaningful step-by-step solutions (Wei et al., 2022). However, these works do not take the principled structure of math into account and treat mathematical expressions as sequences of math LaTeX tokens in the same way as text tokens (Taylor et al., 2022).

1.1 Contributions

In this paper, we introduce a series of novel modifications to language models for the joint representation and generation of natural and mathematical languages. We apply these modifications to the publicly available GPT-2 model as a proof-of-concept, although we believe that these modifications apply to many autoregressive language models. Our contributions can be summarized as follows:

- We develop a set of embeddings that preserve both the semantic and structural properties of mathematical expressions and connect them to natural language token embeddings used in language models. Our embeddings couple the semantic meaning of math tokens with their textual counterparts and explicitly capture the position of nodes in the OPT of an expression.
- We develop a parallelizeable constrained decoding procedure that generates mathematically valid expressions via a set of rules.
- We apply these modifications to GPT-2 and pre-train it on math Wikipedia articles, resulting in a model we call MathGPT.[1] We demonstrate that it outperforms GPT-2 (and other baselines) on downstream generative tasks, especially on generating math expressions, and analyze how it captures the semantic and structural properties of math expressions using its semantic and position embeddings.

2 Methodology

We now detail our proposed modifications grounded in our model MathGPT, visualized in Figure 1. First, we detail how natural and mathematical language are represented jointly by the model. Second, we detail how we provide the model with

[1]Code will be released publicly after the review process.

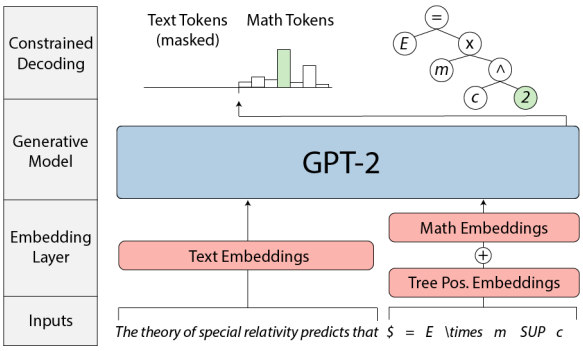

Figure 1: We represent text and math regions differently, with tree position embeddings added to the math token embeddings. The predicted token probability distribution is shown for the next token, *2*; text tokens are masked out by decoding constraints.

token-level tree position information, followed by how we represent math token embeddings via a learnable transformation on text token embeddings. Third, we detail our rules for constrained decoding and tree structure inference at test time.

2.1 Sequence Representation

We consider sequences that contain separable regions of text and math, i.e., $S = (T_1, F^s, M_1, F^e, T_2, F^s, M_2, F^e, \ldots)$, where $T_n = (t_1, \ldots, t_N)$ is a sequence of text tokens, $M_n = (m_1, \ldots, m_N)$ is a sequence of math tokens, F^s indicates the start of a mathematical expression, and F^e indicates the end of an expression. To leverage the structural information of the expressions, we convert each M_n into its corresponding OPT, M_n^{tree}. In M_n^{tree}, each token m_i is assigned a node in the tree, and each $m_i \in \{\mathcal{O}, \mathcal{V}, \mathcal{N}, E\}$, where $\mathcal{O}$ is the set of all operators, $\mathcal{V}$ is the set of all variables, $\mathcal{N}$ is the set of all numbers, and E is a special *end* node. In the tree, operators become parent nodes and their children are either variables, numbers, or other operators. After this initial conversion, we make several modifications to the tree to assist the model with mathematical understanding. First, we add an E node as the last child of every operator node to indicate the end of its list of children. Second, we convert each number into a sub-tree where the head is a special operator O^N and its children are the digits of the number, including the decimal point. Third, since we use a fixed-size vocabulary for math tokens, any out-of-vocabulary token m_i is converted into a sub-tree where the head is a special operator O^U and its children are the text

tokens of m_i. Then, we convert M_n^{tree} back to a linear sequence, M_n^{lin}, by traversing and adding nodes in depth-first order, resulting in the sequence $S' = (T_1, F^s, M_1^{lin}, F^e, T_2, F^s, M_2^{lin}, F^e, \ldots)$. See Supplementary Material E for an illustration of this process along with other data processing steps.

To convert each token $s_i \in S'$ to its embedding $\mathbf{s}_i$ as input to a language model, GPT-2 follows

$$\mathbf{s}_i = \mathrm{emb}_{tok}(s_i) + \mathrm{emb}_{sp}(i),$$

where $\mathrm{emb}_{tok}(s_i)$ is the token embedding for s_i, and $\mathrm{emb}_{sp}(i)$ is the position embedding at index i. Our key innovation in MathGPT is a set of modified position embeddings that explicitly provide the model with OPT structure information:

$$\begin{aligned}\mathbf{s}_i =& \mathrm{emb}_{tok}(s_i) + \mathrm{emb}_{sp}(i) \\ &+ \mathrm{emb}_{tp}(\mathbf{p}_i) + \mathrm{emb}_{type}(s_i),\end{aligned}$$

where $\mathrm{emb}_{type}(s_i)$ is the symbol *type* embedding (text, operator, variable, etc.) of s_i and $\mathrm{emb}_{tp}(\mathbf{p}_i)$ is the *tree position* embedding of s_i. Our approach explicitly captures the tree position and role of each math symbol in the context of the entire mathematical expression to preserve both semantic and structural properties. We also use different *semantic* embeddings $\mathrm{emb}_{tok}(s_i)$ for text and math tokens, which we detail below.

2.2 Tree Position Encoding

For tree position embeddings, we first define $\mathbf{p}_i$, a unique position vector for node m_i, and then use a function emb_{tp} to transform $\mathbf{p}_i$ to a vector with the same dimensionality as the token embeddings. We encode tree positions similar to the approach in (Wang et al., 2021): $\mathbf{p}_i$ is a vector where the entry at each index, p_i^j, represents m_j's index in its list of siblings. By following the indices in $\mathbf{p}_i$ from the tree root, node m_i will eventually be reached, and its index is the last entry in the vector. We then convert $\mathbf{p}_i$ to a vectorized version of the binary representation of each of its entries and finally project the resulting vector using a learnable transformation. The whole process is defined as

$$\begin{aligned}&bin(p_i^j) = \mathrm{concat}(onehot(b_j^1), \ldots, onehot(b_j^K)) \\ &bin(\mathbf{p}_i) = \mathrm{concat}(bin(p_i^1), \ldots, bin(p_i^{|\mathbf{p}_i|})) \\ &\mathrm{emb}_{tp}(\mathbf{p}_i) = \mathbf{W} bin(\mathbf{p}_i),\end{aligned}$$

where b_j^k is the kth digit of p_i^j's binary representation, $onehot$ returns a one-hot 2-vector, and $\mathbf{W}$ is a learnable projection matrix.

2.3 Math Token Embeddings

We construct our semantic embeddings for the math symbols by linking them with the corresponding text tokens in GPT-2's pre-trained vocabulary. Specifically, let the text representation of a math symbol s_i be t_i. We tokenize t_i with the GPT-2 tokenizer to produce a corresponding sequence of text tokens, $(t_i^1, \ldots, t_i^K)$. The embedding of s_i is then given by

$$\begin{aligned}\mathbf{t}_i &= \textstyle\sum_{k=1}^K \mathrm{emb}_{tok}(t_i^k)/K \\ \mathrm{emb}_{tok}(s_i) &= \mathbf{t}_i + \phi_p(\mathbf{t}_i),\end{aligned}$$

where ϕ_p is a fully-connected neural network with a single hidden layer, and we initialize the weights to be small such that $\mathrm{emb}_{tok}(s_i)$ is initially very close to $\mathbf{t}_i$. With this formulation, we leverage the pre-trained information in GPT-2 while updating text token representations during training through MathGPT's tree-structured representations for mathematical expressions. For math symbols that have no corresponding text representations, such as F^s, F^e, O^N, and O^U, we learn their semantic embeddings from scratch.

2.4 Sequence Generation

In addition to modifying GPT-2's input format, we also make several changes to the output process to enable MathGPT to generate mathematically meaningful expressions. We create a new linear predictor head for math tokens, including special control tokens (F^s, F^e, etc.), ϕ_{math}. We concatenate the output of this projection to those of the pre-trained text prediction head, ϕ_{text}, to get a full token probability vector, $\mathbf{a}_i$, at each time step.

To ensure that MathGPT generates mathematically valid expressions, we employ constrained decoding by applying a mask to $\mathbf{a}_i$ that prohibits certain tokens from being generated after s_i. We apply the following constraints: First, text tokens must be followed by text tokens or F^s. Second, F^s must be followed by operator, variable, or number tokens. Third, F^e must be followed by text tokens. Fourth, operator, variable, number, and E tokens must be followed by other operator, variable, number, or E tokens. The exception is when a tree has been fully generated, in which case they must be followed by F^e. Fifth, trees have limited depth and width, so we prevent operator nodes from being generated at the maximum depth level and cap the maximum number of children for each node. Finally, O^U must be followed by text tokens, which

can be followed by other text tokens or E, and O^N must similarly be followed by number tokens.

During training, we minimize the cross-entropy loss using the masked version of token probabilities $\mathbf{a}_i$ to update the GPT-2 parameters along with the MathGPT-specific parameters, including ϕ_{math}, ϕ_p, $\mathbf{W}$, and embeddings of the special tokens.

During testing (generation), we infer the tree position of the next node directly from the position of the previous node from depth-first ordering, according to the following rules: If $s_i \in \mathcal{O}$, then s_{i+1} will be its first child. Thus $\mathbf{p}_{i+1}$ will be a copy of $\mathbf{p}_i$ with a 0 added to the end. If $s_i \in \{\mathcal{V}, \mathcal{N}\}$, then s_{i+1} will be its next sibling. Thus $\mathbf{p}_{i+1}$ will be a copy of $\mathbf{p}_i$ where the last value is incremented by 1. If $s_i = E$, then s_{i+1} will be its parent's sibling. Thus $\mathbf{p}_{i+1}$ will be a copy of $\mathbf{p}_i$ without the last value and the preceding value incremented by 1.

3 Experimental Setup

We now detail a series of experiments to validate the effectiveness of MathGPT. We perform pre-training on a large corpus of math-focused Wikipedia articles and then use the model on various generative downstream tasks involving both natural and mathematical languages.

3.1 Data Pre-Processing

In the pre-training and downstream task datasets, the mathematical expressions are initially represented as plain text or HTML, occasionally wrapped in text-based F^s and F^e tokens, and pre-converted to MathML in the pre-training dataset. To convert them to their OPT representations, we introduce the following data pre-processing pipeline. First, we convert all HTML math-specific symbols, including variables, numbers, and operators, to their LaTeX equivalents, and remove remaining HTML tags. Second, we find all expressions in each text sequence and wrap them with F^s and F^e tokens. Third, we process each sequence with LaTeXML [2], which converts each expression to a tree-like MathML representation. Finally, we process each MathML expression with code from Tangent-CFT (Mansouri et al., 2019) to obtain its standard OPT representation.

We note that LaTeXML introduces several undesirable distortions on mathematical expressions. For example, it is often unable to differentiate between function calls and multiplications with parentheses, multi-character names and multiplications between single character variables, numbers containing commas and comma-delimited lists of numbers, etc. However, we found that in the majority of cases it is accurate enough.

[2] https://math.nist.gov/~BMiller/LaTeXML/

3.2 Pre-Training

We use a pre-trained GPT-2 model to initialize the shared parameters in MathGPT, which enables us to leverage GPT-2's existing representations and capabilities. We then pre-train MathGPT on a large corpus of math-centered Wikipedia articles from the *2016 NTCIR-12 MathIR Task* (Zanibbi et al., 2016b), which enables the model to learn the parameters that are unique from GPT-2. We reserve 5% of the articles for validation and pre-train for 50 epochs, which we found to be sufficient for the model to perform reasonably well on downstream tasks. Additional hyperparameters and model details are listed in Supplementary Material A.

3.3 Downstream Tasks

We evaluate MathGPT on the following downstream generative tasks, which together capture a wide range of mathematical reasoning capabilities. Additional details on the datasets can be found in Supplementary Material D.

Headline Generation We evaluate on the task of math headline generation using the EXEQ-300k dataset (Yuan et al., 2020), which contains pairs of user-authored questions and headlines from Mathematics Stack Exchange. The content in this dataset is generally on college-level math and science topics, containing complex formulas with a large variety of symbols. This task measures the model's ability to extract key information from the question and generate a short summary. Due to reasons we detail below in Section 3.4, we additionally consider two sub-tasks: next mathematical expression prediction and next text region prediction. For next mathematical expression prediction, we consider each expression in the headline to be a generation target, while we use both the question and the portion of the headline up to that expression as input. Similarly, for next text region prediction, we consider each text region that follows a mathematical expression to be a generation target, while we use both the question and the portion of the headline up to that text region as input.

Equation Extraction We evaluate on the task of generating equations in math word problems on a version of the Math23K (Wang et al., 2017) dataset converted to English using Google Translate. While the translations are not perfect, we do see that they largely retain the necessary mathematical information. The dataset contains pairs of middle school-level math word problems and single-variable equations that represent an execution plan to solve the problem. This task measures the model's ability to infer mathematical operations and expression structure from unstructured text.

Student Action Prediction We evaluate on the task of predicting how students act based on feedback while solving math problems in a step-by-step setting. We use a dataset [3] from the Cognitive Tutor system. At each step, the student chooses an action (add, subtract, multiply, etc.) and enters a corresponding input (a mathematical expression) to perform on the problem's equation. If the action is incorrect, the system will provide feedback to the student and let them retry. Our task is to predict exactly what actions students make after receiving feedback after incorrect steps, using the equation being solved, a series of steps the student made, sequential updates to the equation, and a feedback message as input to generate the following student action, input, and outcome as output. This task measures the model's ability to predict which action a student will take based on their previous actions, which involves knowing what the appropriate next step is for solving an equation.

3.4 Evaluation Metrics

Since we evaluate MathGPT on a variety of tasks with different objectives, we similarly measure performance using a wide set of task-specific metrics. For headline generation, where we measure the quality of both generated math and natural language, we use text similarity metrics including **BLEU-4** (Papineni et al., 2002), **ROUGE-L** (Lin, 2004), and **METEOR** (Banerjee and Lavie, 2005). However, since MathGPT outputs mathematical expressions as OPTs while the baselines output them as a sequence of LaTeX tokens, we convert MathGPT's expression output back to LaTeX using a custom tree parser before computing these metrics. We compare the generated output for MathGPT and baselines to a modified version of the ground truth, where the expressions are converted to OPTs via LaTeXML and then converted back to text via the parser. This conversion is necessary since LaTeXML can change the semantics of an expression.

However, these metrics are insufficient since they do not consider the structural integrity of math expressions; for MathGPT, the expressions are generated as trees yet evaluated as text token sequences. To the best of our knowledge, there is no automated metric that can effectively evaluate natural and mathematical languages jointly. Additionally, while human evaluation can be valuable, designing such an experiment is challenging since we need to account for individual text and math properties as well as cohesion between them. We leave both of these aspects for future work. In the current paper, we circumvent this roadblock by including two new evaluation tasks that evaluate text and math *separately*. On math expressions, we use tree edit distance (**TED**) to evaluate their structural integrity.

We use pre-defined train/validation/test splits on the headline generation dataset, and report mean and standard deviation for each metric on the test set over 5 random initializations. For other downstream datasets where pre-defined splits are not available, we perform a 5-fold cross-validation, where the train/test sets are rotated and 10% of the remaining train set is reserved for validation. We similarly report the mean and and standard deviation of each metric on the test set over the 5 folds. In all cases, we perform early stopping on per-token loss on the validation set. In all result tables, we place a * next to a metric value for MathGPT if it outperforms baselines with statistical significance, i.e., $p < 0.05$ from the Student's t-test for cross-validation and Welch's t-test otherwise.

3.5 Baselines

Since the key innovation in MathGPT is a structural modification on top of the original GPT-2 model, our goal is to show that MathGPT outperforms GPT-2 in terms of representing and generating mathematical content. Therefore, we use i) standard GPT-2 and ii) GPT-2 pre-trained on the math-centric Wikipedia articles as our baselines. Moreover, for some of the downstream tasks, we also report state-of-the-art results as an additional baseline. For a fair comparison with MathGPT on text-based metrics, for the headline generation task, we train and evaluate GPT-2 on a version of the dataset where the mathematical expressions are

[3] https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=660

Model	BLEU-4	ROUGE-L	METEOR
MathBERT	49.4	57.7	34.7
MathSum	52.0	54.8	37.5
GPT-2	55.3 ± 1.1	62.1 ± 0.0	43.7 ± 0.3
GPT-2 Wiki	56.1 ± 0.6	62.2 ± 0.1	43.7 ± 0.3
MathGPT	56.5 ± 0.5	62.2 ± 0.1	43.8 ± 0.3

Table 1: Results on headline generation.

converted to OPTs via the processing pipeline and then back to LaTeX using the tree parser. For all other tasks, we train GPT-2 on the original dataset.

4 Experimental Results

We now detail quantitative experimental results to validate the effectiveness of MathGPT in jointly modeling natural and mathematical languages.

4.1 Headline Generation

Table 1 shows results on overall headline generation on the EXEQ-300k dataset; see Supplementary Material B.1 for results on the smaller OFEQ-10k dataset. Table 2 shows results on the next math expression and text region prediction sub-tasks. We emphasize that when evaluated on text and math regions separately, MathGPT significantly outperforms GPT-2, especially on TED, which captures the structural integrity of math expressions, although part of the reason for GPT-2's high TED numbers can be attributed to occasional parsing errors in LaTeXML. Interestingly, MathGPT also significantly outperforms both GPT-2 models in next text region prediction on all metrics, which suggests that trees are highly effective at conveying the underlying meaning of math expressions, which are reflected in the text regions of the headlines.

However, when we evaluate text and math regions jointly on existing text-based metrics, MathGPT's advantage over GPT-2 is minimal and not statistically significant. This result can be attributed to the lack of existing metrics that consider the structural properties of math expressions while combining them with text. Both MathGPT and GPT-2 significantly outperform prior state-of-the-art: MathSum, a sequence-to-sequence method and MathBERT (Peng et al., 2021), a BERT-based method that leverages tree information and adapted to this task. These results show that the GPT family of language models are well-suited to generation tasks on math content even without task-specific architectures such as the copying mechanism.

4.2 Underlying Equation Extraction

Table 3 shows results on the equation extraction task. We also include two task-specific metrics: the percentage of cases where the generated equation and the true equation have the same exact same OPT (**Tree Match**), and the percentage of cases where both evaluate to the same numerical value (**Solve Rate**). We see that MathGPT outperforms both GPT-2 models significantly on all metrics, which implies that MathGPT is effective at both extracting mathematical information from textual problem statements and generating equations as the solution. We observe that MathGPT's advantage over GPT-2 on TED is less than that on the other metrics, due to GPT-2 sometimes generating trees that are similar to the ground truth but containing a few key errors such as using an incorrect operator. We also observe that Solve Rate is higher than Tree Match for all models, since models often generate equations that evaluate to the correct numerical value but have slightly different trees.

4.3 Student Action Prediction

Table 4 shows results on the student action prediction task where we only report the prediction **Accuracy** on each (outcome, action, input) triple. We observe that MathGPT outperforms both GPT-2 models. More specifically, when students are correct, MathGPT predicts the action and input correctly 63.5% of the time, whereas GPT-2 and GPT-2 with math Wikipedia pre-training are correct 61.2% and 61.5% of the time, respectively. However, when students are incorrect, these numbers significantly decrease to 6.6%, 6.4%, and 6.8%. This observation implies that MathGPT outperforms GPT-2 on mathematical reasoning but not on predicting student errors, which is expected since these models do not account for variation in student knowledge.

4.4 Ablation Study

We examine the impact of various components of MathGPT on its downstream performance via an ablation study. Specifically, we create several versions of the model: with no tree position embeddings (**TPE**), with no math symbol type embeddings (**TE**), learning unique math token embeddings instead of linking with text token embeddings (**SE**), and treating most frequent numbers as their own token instead of as subtrees (**NT**). We note that it is difficult to ablate on constrained decoding since it is central to MathGPT; without these con-

	Next Mathematical Expression				Next Text Region		
Model	**BLEU-4**	**ROUGE-L**	**METEOR**	**TED**	**BLEU-4**	**ROUGE-L**	**METEOR**
GPT-2	77.4 ± 0.2	83.1 ± 0.0	56.1 ± 0.1	4.125 ± 0.035	42.5 ± 0.2	58.1 ± 0.3	38.3 ± 0.2
GPT-2 Wiki	77.4 ± 0.2	83.5 ± 0.1	56.2 ± 0.0	4.079 ± 0.045	43.8 ± 0.5	54.5 ± 0.2	40.6 ± 0.0
MathGPT	77.6 ± 0.2	*83.7 ± 0.1	*56.4 ± 0.1	*2.656 ± 0.023	*46.2 ± 0.1	*63.3 ± 0.1	*42.2 ± 0.1

Table 2: Results on next mathematical expression (left) and next text region (right) prediction.

Model	**Tree Match**	**Solve Rate**	**TED**
GPT-2	47.8 ± 1.0	54.6 ± 1.1	2.669 ± 0.099
GPT-2 Wiki	47.2 ± 0.7	54.0 ± 0.8	2.595 ± 0.024
MathGPT	*52.4 ± 1.1	*60.3 ± 1.3	2.449 ± 0.098

Table 3: Results on equation extraction.

Model	**Accuracy**
GPT-2	40.0 ± 0.8
GPT-2 Wiki	40.3 ± 1.2
MathGPT	*41.8 ± 1.0

Table 4: Results on student action prediction.

straints, we may generate unparseable sequences that cannot be interpreted as trees, making some evaluation metrics invalid (e.g., TED). We pre-train all models in the ablation study for 25 epochs and evaluate on the equation extraction task.

Table 5 shows results for the ablation study. We see that all components, except for type embedding, are critical to downstream task performance. Tree position embeddings have a higher impact on accuracy than TED, likely due to these embeddings helping place nodes in correct positions. Removing numeric sub-trees hurts accuracy, likely since it makes it harder for the model to differentiate between multi-token numbers. It also reduces TED, as expected, since number mismatches have a lower overall penalty. Finally, MathGPT pre-trained for 25 epochs outperforms 50 epochs, which implies the model overfits on the Wikipedia data. This observation suggests a more diverse pre-training dataset would help, which we leave for future work.

TPE	**TE**	**SE**	**NT**	**Tree Match**	**TED**	**Solve Rate**
Pre-trained for 50 epochs						
				52.4 ± 1.1	2.449 ± 0.098	60.3 ± 1.3
Pre-trained for 25 epochs						
				53.1 ± 0.9	2.367 ± 0.060	61.2 ± 1.0
×				52.6 ± 0.5	2.371 ± 0.038	60.3 ± 0.8
	×			53.0 ± 0.4	2.362 ± 0.021	61.0 ± 0.7
		×		51.0 ± 0.4	2.464 ± 0.037	58.6 ± 0.6
			×	49.4 ± 0.8	1.734 ± 0.026	57.1 ± 1.0

Table 5: Results of ablation on equation extraction.

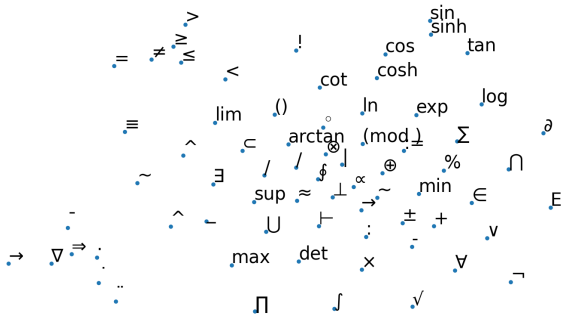

Figure 2: MathGPT operator token embeddings.

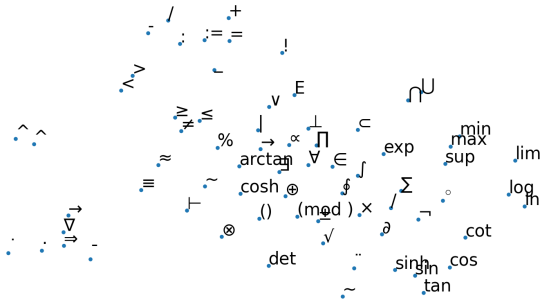

Figure 3: GPT-2 operator token embeddings.

5 Qualitative Analysis

We now qualitatively investigate how MathGPT represents the semantic and structural aspects of mathematical language differently than GPT-2 fine-tuned on the same math Wikipedia articles.

5.1 Math Token Embeddings

Figures 2 and 3 show the semantic embeddings of the top 100 most frequent mathematical *operator* tokens for MathGPT and GPT-2, respectively, visualized in 2D using t-SNE (Van der Maaten and Hinton, 2008). For MathGPT we show $\text{emb}_{tok}(s_i) + \text{emb}_{type}(s_i)$, and for GPT-2 we show the average token embeddings of an operator's LaTeX symbol. We see a few key differences. First, MathGPT seems to group symbols together based on mathematical semantic similarity, whereas GPT-2 seems to group symbols together that may appear in similar contexts. For example, MathGPT groups $=$ and inequalities together and keeps $+$, $-$, and $\pm$ in a separate group. GPT-2 groups $=$ with algebraic

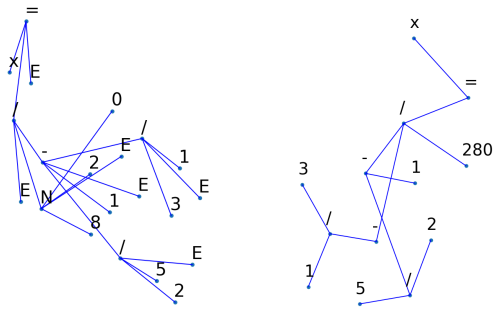

Figure 4: Position embeddings for MathGPT (left) and GPT-2 (right).

operators and other symbols and keeps inequalities in a separate group. Second, MathGPT separates several pairs of symbols that are grouped together by GPT-2 such as (min, max) and ($\bigcup$, $\bigcap$). This observation shows that MathGPT places high importance on an operator's effect on other symbols in addition to its category. We note that different initializations of t-SNE result in different visualizations; see Supplementary Material C.1 for details.

5.2 Tree Position Representations

Figure 4 shows the learned (tree) positional embeddings for MathGPT, i.e., $\text{emb}_{sp}(i) + \text{emb}_{tp}(\mathbf{p}_i)$, and for GPT-2, i.e., $\text{emb}_{sp}(i)$, visualized in 2D using t-SNE, for the mathematical expression $x = 280/(1-(2/5)-(1/3))$. We see that the MathGPT embeddings clearly show a tree structure, where nodes that are deeper in the tree are further from nodes high in the tree and sibling and cousin nodes are close together. For GPT-2, while nodes that appear later in the expression are far from those that appear early, the mathematical structure of the expression is not clearly reflected. While these results are expected, they show that MathGPT's position embeddings explicitly capture the structural properties of mathematical expressions.

6 Related Work

Representations of mathematical language Existing work on studying the representations of mathematical language is mainly motivated by information *retrieval*, i.e., retrieving a semantically and/or structurally relevant mathematical expression (often formula) given a query (Zanibbi et al., 2016a; Davila and Zanibbi, 2017). Both representations based on expert-crafted rules (Zhong and Zanibbi, 2019; Zhong et al., 2020) and those learned from large-scale scientific formula data (Mansouri et al., 2019; Wang et al., 2021) have been shown to be highly effective at this task. However, most of these works do not consider the important textual context around expressions. Several recent works jointly model text and math: TopicEq (Yasunaga and Lafferty, 2019) learns topic keywords associated with expressions, MathSum (Yuan et al., 2020) generates headlines for mathematical discussion forum posts, and one of the MathBERT models (Shen et al., 2021) learns how to grade students' open-ended math responses. However, none of these works leverage the tree structure of math expressions. Another MathBERT model (Peng et al., 2021) and a recent work (Mansouri et al., 2022) jointly represent textual tokens and expressions in their tree format. However, neither is naturally suited for *generation* tasks.

Mathematical reasoning in language models Existing work on studying the mathematical reasoning ability of language models (Lample and Charton, 2020; Saxton et al., 2019b) is mainly motivated by the math *problem solving* task. Despite evidence that pre-trained neural language models have limited mathematical reasoning ability (Saxton et al., 2019a; Jin et al., 2021), they are able to solve simple math word problems accurately using techniques such as verifiers (Cobbe et al., 2021) and chain-of-thought prompting (Wei et al., 2022). However, these models do not take the tree structure of mathematical expressions into account and simply represent them as LaTeX token sequences.

7 Conclusions and Future Work

In this paper, we proposed a series of modifications to common language models to represent and generate text and math jointly. We applied these modifications to GPT-2, resulting in the MathGPT model, which excels at capturing the semantic and structural properties of mathematical expressions in generative tasks. There are many avenues of future work, including i) expand the pre-training dataset for MathGPT to cover a wider range of mathematical topics and complexity levels, ii) develop representations of expressions that are invariant under structural transformations that do not change their semantic meaning, iii) conduct human evaluation to further validate the quality of the generated mathematical expressions in multiple aspects and iv) develop similarity metrics for mathematical language (such as CodeBLEU (Ren et al., 2020) for code) and validate these metrics with evaluations from human experts.

Limitations

There are several limitations to MathGPT, both practical and fundamental. First, the model depends on an external method for converting mathematical expressions to OPTs, currently being LaTeXML. The conversion method is imperfect, which limits MathGPT's capabilities as it will be presented with many distorted expressions during training and at test time. Furthermore, the conversion process is slow and requires dataset-specific engineering to accommodate, making it difficult to deploy the model across many datasets. Second, because MathGPT outputs trees rather than sequences, it is fundamentally difficult to evaluate and utilize in text-based settings without a highly accurate tree-to-text converter. The tree-to-text converter is yet another imperfect process in the pipeline, although it could be improved to a reasonable degree with significant engineering effort. Third, because MathGPT has additional components and requires more information per token than GPT-2, it has higher space and time requirements that make training more expensive. Finally, because MathGPT is pre-trained on highly formal and structured mathematical content, it may struggle to generalize to student-generated mathematical language, which is often error-prone and may exhibit very different patterns.

Ethics Statement

All large language models are prone to reflecting biases seen in their training data. Since MathGPT would likely find its greatest use in an educational setting, extensive care would have to be taken to identify and mitigate bias against students across demographics and backgrounds if deployed in these settings. It is also possible that different patterns exist in mathematical language written by students across demographic groups, such as the choice of variable names or structural choices that reflect different educational backgrounds. Before being deployed in an educational setting, studies should be performed to ensure that the model would not "prefer" any patterns that tend to be exhibited by certain groups of students. It would also be necessary to examine the impact of bias mitigation strategies on removing such preferences and on the effectiveness of the model overall. We did not perform any such studies as part of this work since we did not use any student-generated datasets that contained demographic information, though we welcome such studies and consider it an important part of the future of this line of work.

References

Satanjeev Banerjee and Alon Lavie. 2005. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, pages 65–72.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Kenny Davila and Richard Zanibbi. 2017. Layout and semantics: Combining representations for mathematical formula search. In *Prof. Intl. ACM SIGIR Conf. Res. Develop. Info. Retrieval*, page 1165–1168.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

John A Erickson, Anthony F Botelho, Steven McAteer, Ashvini Varatharaj, and Neil T Heffernan. 2020. The automated grading of student open responses in mathematics. In *Proceedings of the Tenth International Conference on Learning Analytics & Knowledge*, pages 615–624.

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

J. D. Hunter. 2007. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95.

Zhihua Jin, Xin Jiang, Xingbo Wang, Qun Liu, Yong Wang, Xiaozhe Ren, and Huamin Qu. 2021. Numgpt: Improving numeracy ability of generative pre-trained models. *arXiv preprint arXiv:2109.03137*.

Guillaume Lample and François Charton. 2020. Deep learning for symbolic mathematics. In *Proc. Intl. Conf. Learn. Representations*.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh,

Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. 2022. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*.

Zhenwen Liang, Jipeng Zhang, Lei Wang, Wei Qin, Yunshi Lan, Jie Shao, and Xiangliang Zhang. 2022. MWP-BERT: Numeracy-augmented pre-training for math word problem solving. In *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 997–1009, Seattle, United States. Association for Computational Linguistics.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Behrooz Mansouri, Douglas W. Oard, and Richard Zanibbi. 2022. Contextualized formula search using math abstract meaning representation. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, page 4329–4333, New York, NY, USA. Association for Computing Machinery.

Behrooz Mansouri, Shaurya Rohatgi, Douglas W. Oard, Jian Wu, C. Lee Giles, and Richard Zanibbi. 2019. Tangent-cft: An embedding model for mathematical formulas. In *Proceedings of the 2019 ACM SIGIR International Conference on Theory of Information Retrieval*, ICTIR '19, page 11–18, New York, NY, USA. Association for Computing Machinery.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Shuai Peng, Ke Yuan, Liangcai Gao, and Zhi Tang. 2021. Mathbert: A pre-trained model for mathematical formula understanding. *arXiv preprint arXiv:2105.00377*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. CodeBLEU: a Method for Automatic Evaluation of Code Synthesis. *arXiv e-prints*, page arXiv:2009.10297.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019a. Analysing mathematical reasoning abilities of neural models. *arXiv preprint arXiv:1904.01557*.

David Saxton, Edward Grefenstette, Felix Hill, and Pushmeet Kohli. 2019b. Analysing mathematical reasoning abilities of neural models. In *Proc. Intl. Conf. Learn. Representations*.

Shikhar Sharma, Layla El Asri, Hannes Schulz, and Jeremie Zumer. 2017. Relevance of unsupervised metrics in task-oriented dialogue for evaluating natural language generation. *CoRR*, abs/1706.09799.

Jia Tracy Shen, Michiharu Yamashita, Ethan Prihar, Neil Heffernan, Xintao Wu, Ben Graff, and Dongwon Lee. 2021. Mathbert: A pre-trained language model for general nlp tasks in mathematics education. *arXiv preprint arXiv:2106.07340*.

Ross Taylor, Marcin Kardas, Guillem Cucurull, Thomas Scialom, Anthony Hartshorn, Elvis Saravia, Andrew Poulton, Viktor Kerkez, and Robert Stojnic. 2022. Galactica: A large language model for science. *arXiv preprint arXiv:2211.09085*.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11):2579–2605.

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

Zichao Wang, Mengxue Zhang, Richard G. Baraniuk, and Andrew S. Lan. 2021. Scientific formula retrieval via tree embeddings. In *2021 IEEE International Conference on Big Data (Big Data)*, pages 1493–1503.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain of thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Michihiro Yasunaga and John Lafferty. 2019. TopicEq: A Joint Topic and Mathematical Equation Model for Scientific Texts. In *Proc. AAAI conf. Artificial Intell.*

Ke Yuan, Dafang He, Zhuoren Jiang, Liangcai Gao, Zhi Tang, and C Lee Giles. 2020. Automatic generation of headlines for online math questions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9490–9497.

Richard Zanibbi, Akiko Aizawa, Michael Kohlhase, Iadh Ounis, Goran Topic, and Kenny Davila. 2016a. Ntcir-12 mathir task overview. In *Proc. NTCIR Conf. Eval. Info. Access*.

Richard Zanibbi, Akiko Aizawa, Michael Kohlhase, Iadh Ounis, Goran Topic, and Kenny Davila. 2016b. Ntcir-12 mathir task overview. In *NTCIR*.

Mengxue Zhang, Sami Baral, Neil Heffernan, and Andrew Lan. 2022. Automatic short math answer grading via in-context meta-learning. *arXiv preprint arXiv:2205.15219*.

Wei Zhong, Shaurya Rohatgi, Jian Wu, C. Lee Giles, and Richard Zanibbi. 2020. Accelerating substructure similarity search for formula retrieval. In *Proc. European Conf. Info. Retrieval*, pages 714–727.

Wei Zhong and Richard Zanibbi. 2019. Structural similarity search for formulas using leaf-root paths in operator subtrees. In *Proc. Intl. Conf. Neural Info. Process. Syst.*, pages 116–129.

Yanyan Zou and Wei Lu. 2019. Text2Math: End-to-end parsing text into math expressions. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5327–5337, Hong Kong, China. Association for Computational Linguistics.

A Hyperparameters and Implementation Details

MathGPT is implemented in PyTorch, using the pre-trained small OpenAI GPT-2 model (Radford et al., 2019) from the *HuggingFace Transformers* library (Wolf et al., 2020) as a base. For both pre-training and fine-tuning, we use sequence lengths of 1,024, and we limit OPTs to a depth of 32 and a per-node child count of 64. While multi-digit numbers are converted to sub-trees, we use individual nodes to represent single-digit numbers. We use the AdamW optimizer with a learning rate of 1e-5, a weight decay of 1e-2, and a batch size of 4, accumulating gradients every 4 batches. At test time, we generate sequences using beam search with a width of 3. These hyperparameters were chosen based on exploratory evaluations and are mostly the defaults, and no substantive hyperparameter search was performed. We use the same hyperparameters for training MathGPT and GPT-2. All models were trained on *NVIDIA Quadro RTX 8000* or *NVIDIA Tesla V100* GPUs.

For t-SNE, we use the *scikit-learn* (Pedregosa et al., 2011) implementation with a perplexity of 10 and the remaining hyperparameters at their default values. We manually chose random seeds to produce the most visually clear images. We plot data using *matplotlib* (Hunter, 2007). We compute text similarity metrics using the *nlg-eval* (Sharma et al., 2017) library, and compute tree edit distance using the *zss* [4] library.

We note that all software used in the development of this work is either in the public domain, open source, or does not specify a license.

B Additional Downstream Tasks

B.1 Headline Generation - OFEQ-10k

We examine the performance of MathGPT and GPT-2 on the headline generation task using the OFEQ-10k dataset (Yuan et al., 2020). We show the overall task results in Table 6, the expression-only task results in Table 7, and the text-only task results in Table 8. We see surprisingly different results than on the EXEQ-300k dataset. We observe that MathGPT performs worse on the task overall than GPT-2 and GPT-2 Wiki, although it still outperforms them on the expression-only and text-only tasks. We also observe that, counterintuitively, GPT-2 Wiki performs slightly worse on the overall task than GPT-2, although it performs higher on the expression-only and text-only tasks. The negative impact of the Wikipedia pre-training, along with the fact that the trends are reversed when compared to the much larger EXEQ-300k dataset, lead us to believe that pre-training on a larger and more diverse dataset would improve performance on OFEQ-10k. We leave this investigation for future work.

B.2 Student Answer Scoring

We evaluate on the task of scoring student solutions to open-ended math problems from the ASSISTments system. This task helps assess the model's ability to apply mathematical reasoning to student data, as well as generalize to a classification setting. We use the same cleaned dataset and in-context meta-learning method as (Zhang et al., 2022). We also compare to the results from this work, which used a BERT model, and is the current state of the art on this dataset. Since this is a multi-label classification task we use a different set of metrics, specifically **Accuracy**, **F1**, macro-averaged area under the receiver operating characteristic curve (**AUC**), root mean squared error (**RMSE**)

[4] https://github.com/timtadh/zhang-shasha

Model	BLEU-4	ROUGE-L	METEOR
MathSum	29.4	39.0	26.8
GPT-2	34.8 ± 0.6	46.9 ± 0.5	33.2 ± 0.3
GPT-2 Wiki	34.5 ± 0.6	46.4 ± 0.2	32.8 ± 0.3
MathGPT	34.2 ± 0.9	47.2 ± 0.4	32.6 ± 0.5

Table 6: Results on headline generation for OFEQ-10k dataset.

Model	BLEU-4	ROUGE-L	METEOR	TED
GPT-2	58.1 ± 1.4	74.0 ± 0.3	47.0 ± 0.6	3.806 ± 0.066
GPT-2 Wiki	60.0 ± 1.4	76.2 ± 0.3	48.1 ± 0.6	3.444 ± 0.092
MathGPT	$*62.2 \pm 0.8$	$*76.8 \pm 0.1$	$*49.1 \pm 0.3$	$*2.862 \pm 0.031$

Table 7: Results on next expression prediction for OFEQ-10k dataset.

and Cohen's **Kappa**. We show the results of cross-validation on the task in Table 9. We observe that there is no significant difference between MathGPT and GPT-2 on this task. This is possibly due to the fact that many of the samples in the dataset either do not contain math expressions or contain only small ones, minimizing the effect of MathGPT's representations. The results may also imply that MathGPT sees most of its benefits in a generative rather than classification setting, although more experiments would need to be run to confirm this. We did not evaluate this task on GPT-2 Wiki. We note that the improvement over BERT is likely due to additional data processing we performed and small differences in our training setup.

C Additional Qualitative Analysis

C.1 Additional Math Token Representations

We examine the effects of different random seeds for t-SNE initialization on operator token representations. We show two such visualizations for MathGPT in Figure 5 and two such visualizations for GPT-2 (fine-tuned on math Wikipedia articles) in Figure 6. We observe that while most clusters stay the same across initializations, a few tokens tend to float around, in particular $\log$, $\ln$, and $\exp$.

We also show the representations of the 50 most common variable tokens, for both MathGPT and GPT-2, in Figure 7. For both models, we observe that lower- and upper-case versions of the same letter are close together, and that Greek letters are distant from the English letters. Interestingly, in contrast to operator tokens, there is very little change in variable token relationships across the models. This may be because the semantic meaning of variables is highly context-sensitive, preventing MathGPT from making generalizations at the token-level.

D Additional Dataset Details

For completeness and transparency, we list the statistics and other details of all datasets used in this work. We list licenses when they are available, and privacy details when they are relevant.

The math Wikipedia articles used for pre-training are provided under a Creative Commons BY-SA license. We exclusively use the *MathTagArticles* portion of the dataset, which contains 31,839 articles.

The EXEQ-300k and OFEQ-10k datasets consist of (train, validation, test) splits of sizes (261,341, 14,564, 14,574) and (10,301, 1,124, 1,123), respectively. Due to processing errors in a small portion of samples, the EXEQ-300k test set was reduced to a size of 14,474. However, we believe that this reduction is small enough ($\sim 0.7\%$) to not have a significant impact on reported results.

The Math23k dataset consists of 23,162 math word problems originally in Chinese and translated to English for use in this work. We chose to not use the publicly available test split for this dataset because it is very small compared to the size of the dataset (1000 samples), so cross-validation would provide a better measure of performance.

The Cognitive Tutor dataset consists of 8,298 unique problems and 95 students. All student identities are anonymized. Since student responses are constrained by the software, it is unlikely that they contain personally identifying information.

The version of the student answer scoring dataset we use consists of 1,333 unique problems and 130,940 responses, with each assigned a score from 1 to 5. The original dataset was introduced by (Erickson et al., 2020). While student identities are anonymized, it is possible that personally identifying information is present in the open-ended

Model	BLEU-4	ROUGE-L	METEOR
GPT-2	15.1 ± 0.5	40.4 ± 0.5	20.2 ± 0.5
GPT-2 Wiki	20.5 ± 0.2	38.3 ± 0.4	25.0 ± 0.3
MathGPT	20.6 ± 0.5	*43.7 ± 0.2	*26.6 ± 0.3

Table 8: Results on next text region prediction for OFEQ-10k dataset.

Model	Accuracy	F1	AUC	RMSE	Kappa
BERT	–	–	73.3 ± 0.6	1.077 ± 0.002	58.9 ± 0.4
GPT-2	82.4 ± 0.2	61.8 ± 0.5	94.3 ± 0.2	0.933 ± 0.007	63.6 ± 0.4
MathGPT	82.2 ± 0.3	61.8 ± 0.7	94.2 ± 0.1	0.935 ± 0.014	63.7 ± 0.5

Table 9: Results on student answer scoring task.

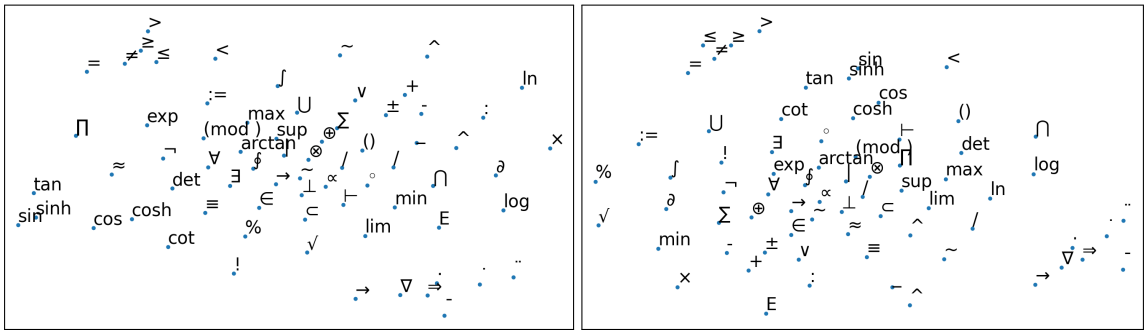

Figure 5: MathGPT operator token embeddings with different t-SNE random seeds.

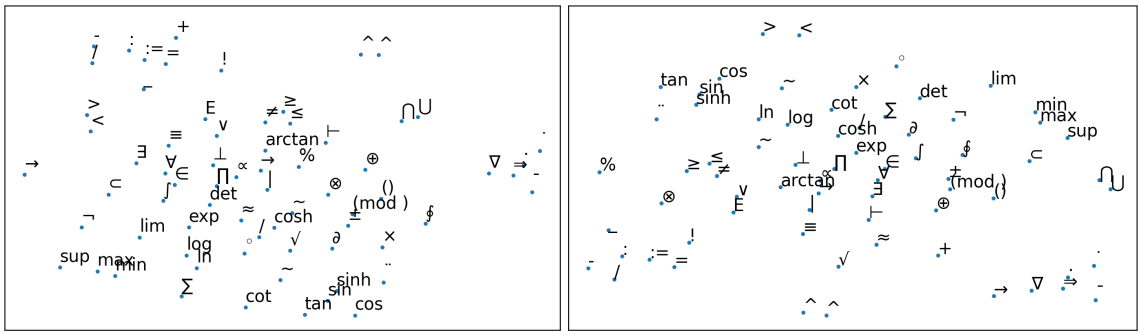

Figure 6: GPT-2 operator token embeddings with different t-SNE random seeds.

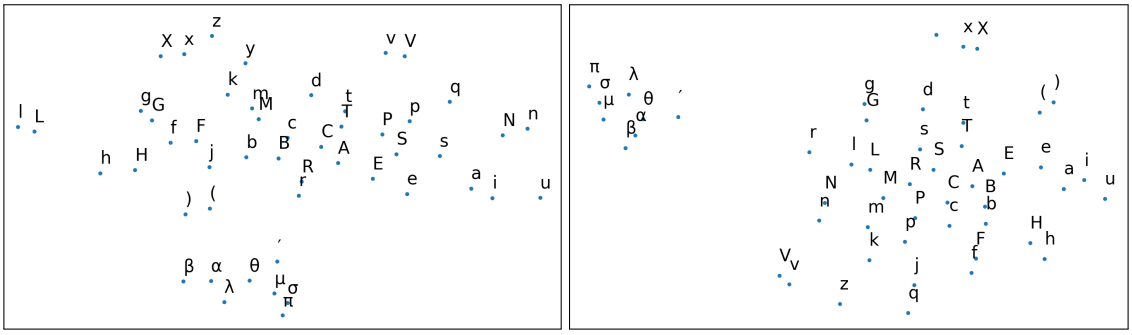

Figure 7: Variable token embeddings for MathGPT (left) and GPT-2 (right).

student responses, and as such the dataset is not publicly available.

E Data Pipeline Illustration

In Figure 8, we show the full data processing pipeline for a single expression.

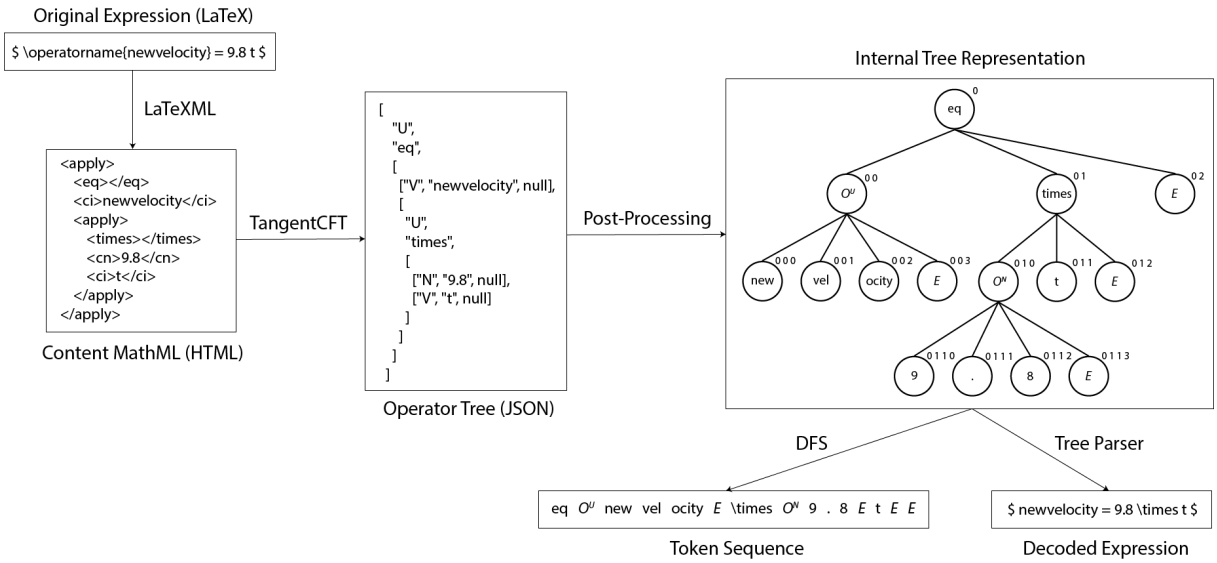

Figure 8: Data processing pipeline for the expression $\mathrm{newvelocity} = 9.8t$. The expression is initially converted to Content MathML format by LaTeXML, and is stored as HTML. It is then converted to a recursive operator tree format by TangentCFT, and is stored as JSON. Each node is represented by a 3-tuple, storing the TangentCFT type, followed by the node's name, followed by the list of children or *null* if there are none. The expression is then sent through the post-processing pipeline, which tokenizes nodes, converts nodes out of the vocabulary to GPT-2-tokenized sub-trees, converts numbers to sub-trees, adds *end* nodes, and computes tree position vectors (shown to the upper right of each node). This representation can be converted to a depth-first traversal of the tokens in order to be processed by MathGPT. It may also be converted back to human-readable text as LaTeX.