Project Proposal: Transformer-based Model for Mathematical Problem Solving

Chau Nguyen
chauminh.nguyen@mail.utoronto.ca

Gabriel You
gabriel.you@mail.utoronto.ca

Takia Talha
takia.talha@mail.utoronto.ca

Taha Siddiqi
taha.siddiqi@mail.utoronto.ca

Department of Mathematical & Computational Sciences
University of Toronto Mississauga

Abstract

Mathematical problem solving often requires both symbolic manipulation and natural language understanding. In this paper, we propose a hybrid, lightweight, and open-source model that combines reason-based mathematics with deep learning to solve mathematical problems expressed in natural language or LaTeX format. Our approach integrates several powerful components:`MathBERT` for tokenizing and encoding both natural language and symbolic math content, and a T5 architecture featuring an encoder-decoder structure to generate a LATEX formatted answer to math word problems. This architecture allows the model to handle complex mathematical reasoning tasks, from simple algebraic equations to advanced calculus, while providing accurate and interpretable solutions. We demonstrate that our model leverages the strengths of neural approaches to deliver a more robust solution for automated math problem-solving. Our code and final report is available for review at our GitHub repository

1 Introduction

A common challenge faced by students in computational sciences is developing the ability to solve mathematically intensive and logically demanding problems. The transition from computation-oriented mathematics to reasoning-based approaches often presents a significant learning curve, primarily due to the difficulty in cultivating mathematical intuition and the limited availability of detailed, step-by-step solutions for comparison. This work aims to address these gaps by designing an accessible model capable of taking a LaTeX-formatted mathematical problem as input and producing a descriptive, rigorous LaTeX-formatted solution.

The development of models that excel in mathematical problem solving is recognized as a highly challenging task, largely due to the dual demands of consistency and mathematical rigor in generated outputs [1]. Current state-of-the-art models, such as DeepMind's FunSearch [2] and OpenAI's O1 [3], leverage large-scale transformer-based language models (LLMs) pretrained on extensive datasets. These models often incorporate symbolic reasoning tools, e.g., the FunSearch algorithm in DeepMind's framework, to enhance their problem-solving capabilities. However, these approaches face notable limitations, including inconsistent outputs caused by varied representations of the same problem, computationally expensive to train, and closed-source architecture information and training data.

To address these shortcomings, we propose an open source transformer-based architecture that uses pre-trained mathBERT embeddings with the T5 architecture [4]. This approach seeks to create a high performance model using transfer learning through the use of neural network architecture.

Our work also outlines key evaluation metrics, including precision in output solutions, logical coherence, and consistency across varying problem representations. We plan to benchmark our model against existing systems, such as FunSearch, using datasets like the MATH benchmark and additional datasets tailored for symbolic reasoning tasks. Through this effort, we aim to provide a reliable and user-friendly tool capable of delivering step-by-step feedback and reasoning for diverse mathematical queries.

2 Background and Related Work

The application of large language models (LLMs) such as BERT [5] and GPT [6] has revolutionized the field of natural language processing (NLP). These transformer-based models excel at a wide range of tasks, including text classification, question answering, and translation. However, when it comes to mathematical reasoning and algebraic problem solving, LLMs often face challenges due to their primary design focusing on natural language processing rather than formal mathematical computations.

Recent work has explored enhancing the mathematical capabilities of LLMs by fine-tuning them on large, diverse mathematical corpora. For instance, BERT, originally designed for NLP tasks, has been shown to benefit from such fine-tuning to improve its ability to understand and solve algebraic problems [5]. Given that BERT and its variants are open source, lightweight, and accessible, they offer a promising avenue for advancing research in mathematical reasoning, as they can be easily customized for a range of tasks.

2.1 Challenges in Mathematical Reasoning for LLMs

Although deep learning models, particularly transformers, are effective at processing unstructured data like natural language, they often struggle with tasks requiring structured reasoning, such as solving algebraic equations or performing symbolic manipulation. The primary limitation stems from the fact that transformers, by design, do not explicitly encode mathematical operations or algebraic logic. Instead, they learn patterns from data, which can sometimes lead to difficulties in reasoning through complex step-by-step processes required for tasks like equation solving.

To address these challenges, recent approaches have focused on fine-tuning transformer models on mathematical datasets to help them develop an understanding of mathematical operations and structures. While these models have made progress in handling algebraic expressions, more complex mathematical reasoning tasks, such as multi-step problem solving and manipulation of algebraic terms, still present challenges.

2.2 Enhancements in Transformer-based Architectures for Mathematical Reasoning

In this context, transformer-based architectures, such as BERT, have been increasingly used for algebraic reasoning tasks. Fine-tuning models on mathematical datasets allows them to develop an understanding of algebraic relationships, helping them solve equations and simplify expressions. These models benefit from the generalization capabilities of large-scale transformers, which can adapt to a variety of mathematical problems through training on diverse datasets.

In particular, BERT and its variants, including mathBERT [5], have been shown to excel in mathematical reasoning tasks when fine-tuned on a corpus of algebraic problems. Unlike earlier models, these transformers learn the contextual dependencies between mathematical terms and operators, which helps them understand complex expressions and perform the necessary operations to solve equations or simplify expressions. The flexibility and accessibility of these models make them an attractive solution for educational and research purposes, allowing for greater experimentation and customization.

2.3 Contributions of the Current Work

In this work, we propose an enhanced transformer-based approach to algebraic problem solving by leveraging the BERT encoder and fine-tuning it on a large mathematical corpus. Our approach focuses on developing a model that can handle complex algebraic reasoning tasks by training the model to understand and manipulate algebraic expressions effectively. We hypothesize that by extending BERT's architecture to better model mathematical reasoning, our model will achieve improved performance on a range of algebraic problem-solving tasks, generating both accurate and interpretable solutions. This work contributes to the growing body of research aimed at advancing LLMs' ability to handle mathematical problem solving, pushing the boundaries of what is achievable in algebraic reasoning within LLMs.

3 Data

3.1 Datasets

For this project, we decided to use the following datasets:

Math Dataset

The MATH dataset [7] consists of 12,500 (7,500 training and 5,000 test) problems from mathematics competitions including the AMC 10, AMC 12, AIME, and more. Many of these problems can be collected from AoPS Contests. [8]

NuminaMath-CoT

The Numina Math CoT dataset has approximately 860k math problems, where each solution is formatted in a Chain of Thought (CoT) manner. The sources of the dataset range from Chinese high school math exercises to US and international mathematics olympiad competition problems. [9]

Here is a breakdown of the math sources in the NuminaMath-CoT dataset:

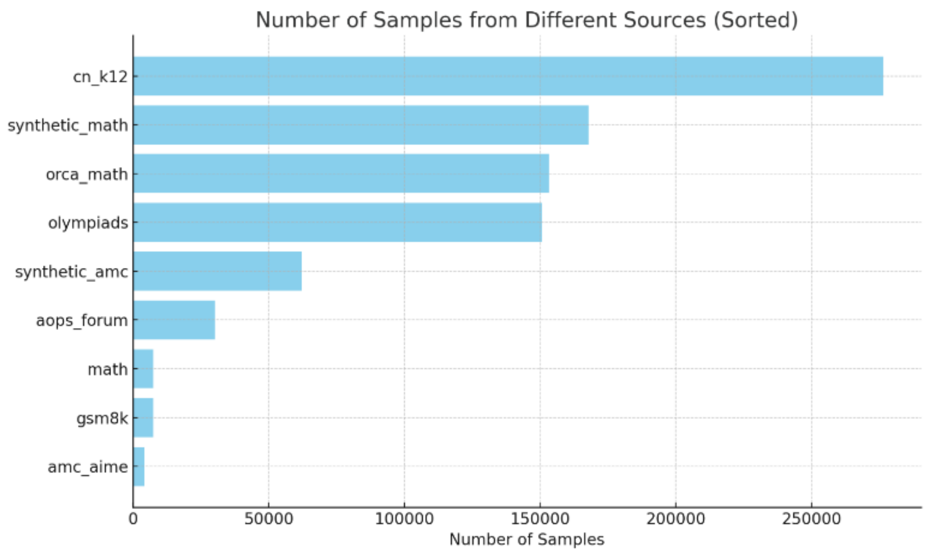


Figure 1: Distribution of math sources in the NuminaMath-CoT dataset.

It is important to note that there is an imbalance of problems in the dataset. The most frequent math topics, such as algebra and geometry, are disproportionately more represented than the remaining categories.

3.2 Data Formatting

Problems and solutions are formatted using LATEX. The usage of LATEX ensures that the data is easily readable and is easy to parse and process for training the model.

The data for the MATH Dataset is formatted as a JSON file, with each problem containing the following fields:

- problem: The text of the problem
- level: The difficulty level of the problem (Level 1 up to Level 5)
- type: The type of math problem (e.g. Number Theory, Geometry, Algebra, Prealgebra, Counting & Probability, Precalculus, Intermediate Algebra)
- solution: The solution to the problem

The data for the Numina Math CoT dataset are formatted as a JSON file, with each problem containing the following fields:

- An array of objects, where the first object contains
  - content: The text of the problem
  - role: the role assigned to the person who can access this data (user)
- The second object contains
  - content: The solution to the problem
  - role: the role assigned to the person who can access this data (assistant)

3.3 Data Preprocessing

We converted the data into the following format:

- problem: The text of the problem
- solution: The solution to the problem

In this way, the features and labels are clearly defined, making it easier to train the model.

Then, we filtered all foreign characters and symbols (i.e. Chinese characters) from the data to ensure that the model is trained on clean and consistent data. This preprocessing step helps to remove any noise or irrelevant information that could affect the model's performance.

Finally, we tokenized the text data using the MathBERT tokenizer, which is specifically designed for mathematical tasks. This tokenizer splits the input into subword tokens that represent both natural language components and mathematical symbols, enabling the model to process math-related queries effectively.

3.4 Data Splitting

We split the data into training, validation, and test sets. The training set is used to train the model, the validation set is used to tune hyperparameters and prevent overfitting, and the test set is used to evaluate the model's performance on unseen data.

The split is as follows:

- Training set: 80%
- Validation set: 10%
- Test set: 10%

We distributed the Numina CoT dataset across the training, validation, and test sets. However, since the MATH dataset only had 12.5k problems, it was merged with the test set to evaluate the model's performance on a different set of problems.

4 Model Architecture

The model architecture is based on the T5 Transformer framework utilizing mathBERTs tokenizer and embeddings, designed to process and generate sequences efficiently. This model leverages a modular design consisting of an encoder-decoder structure tailored for natural language understanding and generation tasks. The architecture is parameterized to balance computational efficiency and performance, making it adaptable to various downstream applications.

4.1 Encoder

The encoder is responsible for processing the input sequence and encoding it into a high-dimensional representation that captures its semantic and syntactic structure. It consists of a stack of N identical layers, where each layer has the following components:

- **Multi-Head Self-Attention:** Captures dependencies across all positions in the input sequence, allowing the model to focus on relevant tokens regardless of their distance. The scaled dot-product attention mechanism is computed as:

  $$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V,$$

  where Q, K, and V represent the query, key, and value matrices, respectively.

- **Feed-Forward Network (FFN):** Applies two linear transformations separated by a ReLU activation, enabling non-linear feature learning. The FFN is parameterized as:

  $$\text{FFN}(x) = \text{ReLU}(xW_1 + b_1)W_2 + b_2.$$

- **Layer Normalization and Residual Connections:** Layer normalization ensures numerical stability, while residual connections facilitate gradient flow during training.

Each layer in the encoder outputs a contextualized representation for every token in the input sequence.

4.2 Decoder

The decoder generates the target sequence one token at a time, conditioned on both the encoder outputs and the previously generated tokens. Similar to the encoder, it consists of M identical layers with additional mechanisms to incorporate encoder information:

- **Masked Multi-Head Self-Attention:** Ensures that predictions at position t depend only on tokens up to $t - 1$ by masking future positions in the attention computation.
- **Cross-Attention:** Attends to the encoder's outputs to integrate contextual information from the input sequence. The attention mechanism operates on the encoder's key-value pairs and the decoder's queries.
- **Feed-Forward Network:** Identical to the encoder's FFN, applied to each decoder layer.

The decoder outputs logits for the target vocabulary, which are transformed into probabilities using a softmax layer.

4.3 Positional Encoding

Since transformers lack an inherent notion of sequence order, positional encodings are added to the token embeddings to incorporate positional information. The positional encoding vector is defined as:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right), \quad PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right),$$

where pos is the position and i is the dimension index.

4.4 Training Objective

The model is trained using a cross-entropy loss function, defined as:

$$\mathcal{L} = -\sum_{t=1}^{T} y_t \log p(y_t | y_{<t}, x),$$

where y_t is the target token at time t, and $p(y_t|y_{<t}, x)$ is the predicted probability of y_t given the input sequence x and previously generated tokens.

4.5 Parameterization

The model's configuration includes:

- **Hidden Dimension ($d_{\textbf{model}}$):** 768.
- **Number of Attention Heads:** 8.
- **Feed-Forward Dimension ($d_{\textbf{ff}}$):** 2048.
- **Number of Encoder Layers:** 6.
- **Number of Decoder Layers:** 6.
- **Dropout Rate:** 0.1.

This configuration results in a model with approximately 109.5 million parameters.

4.6 MathBERT Tokenizer and Encoder

The parsed mathematical expression is fed into a `MathBERT` tokenizer and encoder, which is a specialized version of BERT fine-tuned for mathematical tasks [5]. MathBERT has been trained to handle both natural language and symbolic mathematical notation, which allows it to effectively process math-related queries and understand the underlying structure of the problem. The steps involved are as follows:

- **Tokenization:** The `MathBERT` tokenizer splits the input into subword tokens that represent both natural language components and mathematical symbols. This step is crucial for handling mixed inputs such as text-based questions and mathematical formulas.
- **Contextual Embedding:** The tokenized input is passed through the `MathBERT` encoder, which generates contextual embeddings for each token. These embeddings capture semantic relationships between mathematical operations, variables, and their corresponding natural language descriptions. The encoder's attention mechanism enables the model to focus on relevant parts of the input when generating solutions.

5 Model architecture figure

Below is a simplified overview of the model architecture, which consists of a tokenization step, an embedding layer, an encoder, a decoder, and an output layer.

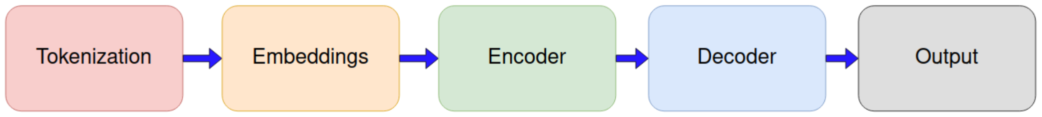


Figure 2: The Transformer - model architecture.

We will describe each component in detail in the following sections.

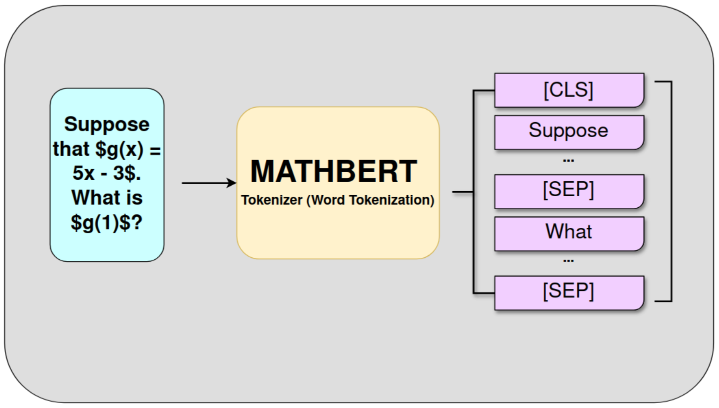

Suppose that $g(x) = 5x - 3$. What is $g(1)$?
MATHBERT
Tokenizer (Word Tokenization)
[CLS]
Suppose
...
[SEP]
What
...
[SEP]

Figure 3: Tokenization of the input sequence.

Tokenization is the process of converting raw text into smaller units, or tokens, that the model can process. In our architecture, we employ subword tokenization to handle diverse vocabulary efficiently, breaking words into meaningful subunits while retaining their contextual integrity.

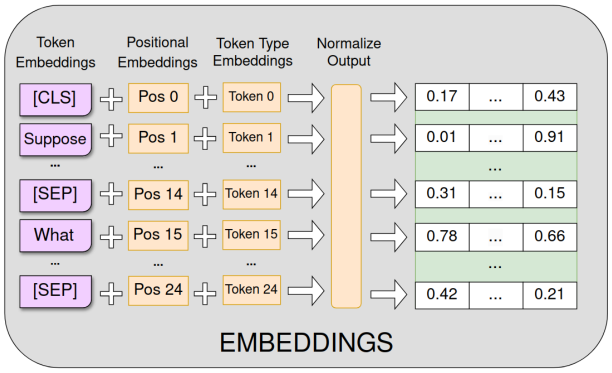


Figure 4: Embedding layer mapping tokens to dense vectors.

Embeddings are dense vector representations of tokens that capture their semantic and contextual meaning. In our architecture, each token is mapped to a fixed-dimensional embedding space, enabling the model to understand relationships between words and symbols. These embeddings serve as the foundation for processing input sequences, preserving both linguistic and mathematical nuances for effective problem-solving.

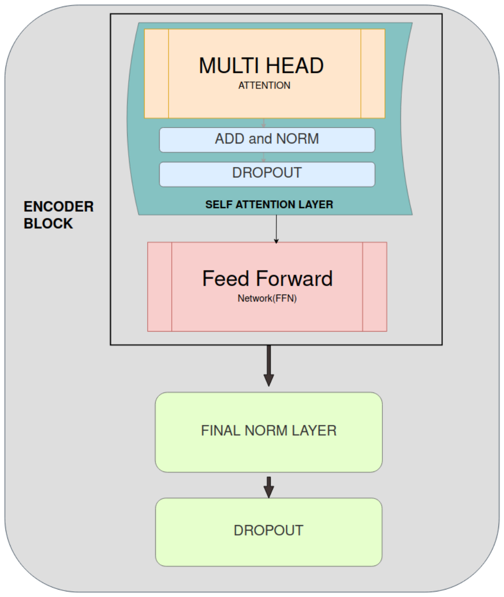


Figure 5: Encoder processing the embedded input sequence.

The encoder is the core component of our architecture that processes input embeddings to extract contextual representations. It uses self-attention mechanisms to capture relationships between tokens across the entire sequence, ensuring a deep understanding of both natural language and mathematical structures. This allows the model to generate rich, context-aware representations essential for solving complex problems.

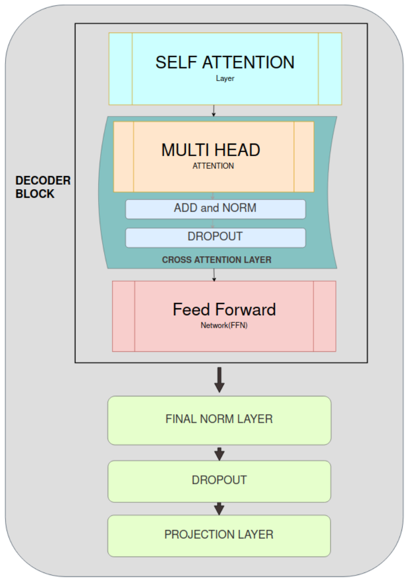
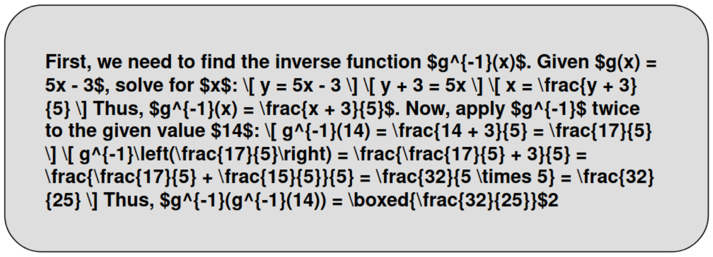


Figure 6: Decoder generating the output sequence.

The decoder transforms contextual representations from the encoder into meaningful outputs, It consists of 6 stacked decoder blocks, each processing the output from the previous block. Each block includes a self-attention layer to capture relationships within the target sequence, a cross-attention layer to focus on relevant encoder outputs, and a feed-forward layer (FFN) for further refinement. After each layer, add & norm and dropout are applied for stability and regularization.



```
First, we need to find the inverse function $g^{-1}(x)$. Given $g(x) =
5x - 3$, solve for $x$: \[ y = 5x - 3 \] \[ y + 3 = 5x \] \[ x = \frac{y + 3}
{5} \] Thus, $g^{-1}(x) = \frac{x + 3}{5}$. Now, apply $g^{-1}$ twice
to the given value $14$: \[ g^{-1}(14) = \frac{14 + 3}{5} = \frac{17}{5}
\] \[ g^{-1}\left(\frac{17}{5}\right) = \frac{\frac{17}{5} + 3}{5} =
\frac{\frac{17}{5} + \frac{15}{5}}{5} = \frac{32}{5 \times 5} = \frac{32}
{25} \] Thus, $g^{-1}(g^{-1}(14)) = \boxed{\frac{32}{25}}$2
```

Figure 7: Final output generated by the model.

The final output generated by the model represents the solution to the input problem. The output is a sequence of tokens that can be converted back into human-readable text or LaTeX format, providing a clear and interpretable solution for the given mathematical problem.

6 Results

We implemented the model and the training process using PyTorch, as well as conducting training using Google Colab T4 GPU. Due to the limited time and computing resources available, we decided to test our model's performance on a subset of the whole training dataset (10,000 out of approximately 670,000 samples) over 10 training epochs. Each epochs takes approximately 30 to 40 minutes to complete one pass through the entire training and validation samples. We present the result and a detailed analysis of the training progress and chosen evaluation metrics.

6.1 Results Table

The following table summarizes the key results of the training process, including the per-epoch values of the training loss, validation loss, and perplexity for both the train and validation sets:

Epoch	Time	Train Loss	Train PPL	Val Loss
01	$34m17s$	3.437	31.091	2.765
02	$58m39s$	2.796	16.381	2.300
03	$83m1s$	2.434	11.406	1.970
04	$117m23s$	2.155	8.627	1.695
05	$141m44s$	1.929	6.883	1.458
06	$176m6s$	1.736	5.674	1.263
07	$220m28s$	1.559	4.755	1.076
08	$254m49s$	1.409	4.092	0.940
09	$289m11s$	1.268	3.555	0.815
10	$313m33s$	1.149	3.156	0.717

7 Discussion

7.1 Training and Validation Losses

The training and validation losses decreased steadily over the course of 10 epochs. The slow reduction in loss suggests that the model is learning effectively, while he training loss was consistently higher than the validation loss. This behavior is indicative of overfitting being controlled, as the model does not drastically overperform on the training data relative to the validation set.

7.2 Perplexity Scores

In order to evaluate how well the model text generation capability is improving when presented with unseen data, we decided to monitor the perplexity score during training. Perplexity is a measure of how well a probability model predicts a sample, with lower perplexity indicating better predictive performance. Formally, it is defined as:

$$\text{Perplexity}(P) = 2^{H(P)}$$

where $H(P)$ is the entropy of the probability distribution P, which quantifies the uncertainty of the model's predictions.

In the context of our fine-tuning task, the perplexity score provides insight into how well the hybrid model can generate mathematically accurate solutions for NLP-based problem-solving tasks without simply repeating solutions from the training set. As the perplexity decreases, it reflects the model's growing ability to predict mathematical outcomes with increasing certainty.

Over the course of training, we observed a consistent decrease in the perplexity score, suggesting that the hybrid model was becoming increasingly better at generating mathematically accurate outputs as the epochs progressed. However, towards the last 3 epochs, the perplexity scores appeared to have slowed down decreasing, which might be indicative of overfitting due to the small selection of dataset.

8 Limitations

One limitation is related to the embeddings used in our model. We utilized MathBert embeddings, which were pretrained on natural language processing (NLP) tasks for mathematical understanding, such as summarization, detecting mathematical formulas, and auto-grading of student performance. However, these embeddings are not specifically designed for mathematical reasoning, which is crucial for our model's purpose. As a result, our model significantly relies on the fine-tuning of the T5 Encoder-Decoder. Given the limited resources, the fine-tuning process may not have been sufficient to fully adapt the embeddings for strong mathematical reasoning, and additional training could potentially improve the model's performance. Future work could also explore integrating symbolic semantic analysis to enhance the model's understanding and handling of complex mathematical problems. [5]

Furthermore, the model consists of mainly algebraic problems, which may limit its performance on other types of mathematical problems, such as geometry, calculus, or number theory. The lack of diversity in the training data could cause the model to overfit to the training set, leading to biases in the model's predictions and hinder its generalizability to a broader range of mathematical tasks. To address this limitation, future work could focus on expanding the dataset to include a more diverse set of mathematical problems, ensuring that the model is robust and effective across various domains.

Lastly, the final limitation of our model is the restricted computing resources, as it was trained on a Google Colab Tesla T4 GPU. In comparison, popular large language models such as ChatGPT, Claude, and Bard were trained for significantly longer periods on much larger datasets and with more extensive computational resources. This disparity in training time and resources means that our model could potentially perform better if it were trained with more computational power. Additionally, the lack of parallelization in our training process further limits the efficiency and scalability of our model. Implementing parallelization techniques could significantly reduce training time and improve model performance by leveraging multiple GPUs or distributed computing resources.

9 Ethical considerations

One ethical consideration is the use of the model in competitive or professional settings, where reliance on the model could lead to unfair advantages or undermine the credibility of the work produced. In competitive exams or professional certifications, using the model to generate solutions could be considered unethical and devalue the merit of the qualifications, while in professional settings, using the model without proper understanding could result in subpar work quality and potential professional misconduct. To mitigate the risk of misuse, clear guidelines and policies should be established regarding the appropriate use of the model, such as prohibiting its use in exams or certifications and implementing measures to detect and prevent cheating. In professional environments, the model should serve as a supplementary tool to assist with tasks rather than replace grading or evaluation, ensuring that the model's use is aligned with ethical standards and minimizing the risk of misuse.

Another ethical issue is the potential for dataset-related biases, as the training data predominantly consists of problems from US contests and Chinese school exercises, which may lead to the model performing better on those types of problems while underperforming on problems from other regions or educational systems. Additionally, the dataset mainly consists of algebra-related questions, which could result in unequal performance and biased grading, as the model may not be equally effective across all types of problems. To mitigate this risk, the dataset should be expanded to ensure diversity and representation from a wide range of problems across different regions, educational systems, and difficulty levels. This can be achieved by curating a balanced dataset from various sources, while regularly evaluating the model's performance to identify and address any biases.

Finally, the last ethical issue is intellectual property rights, as the model is trained on copyrighted data, which could inadvertently promote plagiarism if it provides solutions that students submit as their own, undermining the learning process and leading to academic dishonesty. In professional settings, using the model to generate solutions without proper attribution could violate intellectual property rights and ethical standards. To mitigate this risk, it is essential to ensure that the data used to train the model is properly licensed and that the model does not directly reproduce copyrighted material. This can be achieved by using open-source datasets or those with appropriate licenses for educational use, and by adding a disclaimer to the model's output to indicate that the solution is generated by the model and should be verified and properly attributed before use. Promoting ethical use and proper attribution can help minimize the risk of intellectual property rights violations.

10 Conclusion

References

[1] Mert Ünsal, Timon Gehr, and Martin Vechev. Alphaintegrator: Transformer action search for symbolic integration proofs, Oct 2024.

[2] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, et al. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[3] OpenAI. Introducing openai o1 preview, 2024. Accessed: 2024-12-12.

[4] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer, 2023.

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://aclanthology.org/N19-1423/. [Online; accessed 2024-11-11].

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, and et al. Language models are few-shot learners: Proceedings of the 34th international conference on neural information processing systems, Dec 2020.

[7] Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.

[8] Art of Problem Solving. Aops online community. `https://artofproblemsolving.com/community/c3158_usa_contests`, 2024.

[9] Jia LI, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Costa Huang, Kashif Rasul, Longhui Yu, Albert Jiang, Ziju Shen, Zihan Qin, Bin Dong, Li Zhou, Yann Fleureau, Guillaume Lample, and Stanislas Polu. Numinamath. `[https://huggingface.co/AI-MO/NuminaMath-CoT](https://github.com/project-numina/aimo-progress-prize/blob/main/report/numina_dataset.pdf)`, 2024.