Aligning Context and Formulas: Multi-Stage Fine-Tuning for Large Language Models with A Novel Dataset ContextFormulas50K

Anonymous ACL submission

Abstract

Mathematics plays a crucial role in scientific research. However, it is never easy to formulate a problem using mathematical formulas, even for senior researchers. Fine-tuning pre-trained models on mathematical datasets to facilitate research has become a widely adopted approach. Although there are numerous mathematical datasets, only high-quality datasets could enhance the pre-trained model for a deep insight into formulas because mathematics is a precise discipline. To address this, we propose the **ContextFormulas50K** dataset, which consists of mathematical formulas paired with their contextual text. Based on this dataset, we fine-tune two pre-trained models (i.e. LLaMA-8B-Instruct and Mistral-7B) to generate precise formulas for research work. But in this task, there emerges the **information bottleneck problem**, which means the parameter scale expands much faster than performance. To overcome this problem, we introduce a **Multi-Stage Fine-Tuning**(**MSFT**) approach to equip large language models (LLMs) with better mathematical comprehension. Specifically, there are three stages in our model, and by progressively inserting plug-and-play modules, model performance could be enhanced at each stage, which means our method effectively alleviates information bottlenecks. Experimental results show that our model effectively improves mathematical comprehension and achieves state-of-the-art performance in the formula generation task, outperforming multiple commercial baselines (i.e., GPT-4, GPT-4o, and Claude-3.5-sonnet).

1 Introduction

The rise and rapid development of interdisciplinary fields have created challenges, particularly for researchers unfamiliar with complex mathematics. An appropriate formulation could effectively convey the methodology precisely both in the research process and in scientific writing. However, mathematical formulas in scientific papers can be particularly perplexing for individuals, such as economists and psychologists, especially for those with limited experience in mathematical modeling.

To overcome this obstacle, we intend to apply large language models (LLMs)(Zhao et al., 2024) for formula generation. Therefore, we fine-tune pre-trained models(Devlin et al., 2019; Liu et al., 2019; Raffel et al., 2020; Ding et al., 2023) with our contributed high-quality dataset **ContextFormulas50K**(**CF50K**), which consists of a formula paired with its surrounding contexts. However, we encounter **information bottleneck problem**, which means the parameter scale expands much faster than performance. To solve this issue, with the parameter-efficient fine-tuning(PEFT) methodology(Houlsby et al., 2019; Li and Liang, 2021a; Hu et al., 2021; Ben Zaken et al., 2022), we propose a novel framework that **Multi-Stage Fine-Tuning** (**MSFT**). Specifically, we apply multiple stages to train our inserted modules, and by incrementally inserting modules at each stage, LLM gradually demonstrates superior comprehension of complex mathematical formulas.

In addition, there are three stages, and our approach involves training plug-and-play modules at each stage. On the one hand, when there is no formula in our input, our trained modules would not be inserted, which means that our method produces no side effect for models just to process text. On the other hand, when formula comes across the input, we then insert our trained modules to comprehend mathematics, which demonstrates a novel methodology ***to enhance large language models for a particular data type basically without knowledge loss*** (French, 1999; Kirkpatrick et al., 2017). In this way, the performance for textual queries would not be affected by our fine-tuning process, while LLM can comprehend mathematics via our methodology without a side effect.

Experimental results show that our model im-

proves mathematical comprehension effectively and achieves state-of-the-art performance against multiple commercial models (i.e., GPT-4, GPT-4o(Brown et al., 2020; OpenAI et al., 2024) and Claude-3.5-sonnet) in the task of formula generation.

Our contributions are threefold:

- We introduce a dataset for mathematic comprehension, **ContextFormulas50K** (**CF50K**), consisting of mathematical formulas and their surrounding texts. This dataset facilitates the formula generation task. To our best knowledge, this is the first dataset of its own type.

- We propose our novel **Multi-Stage Fine-Tuning(MSFT)** method, where each stage trains a plug-and-play module for a specific stage-wise task. This approach significantly addresses the **information bottleneck problem**, and achieves state-of-the-art performance even against commercial products (i.e., GPT-4, GPT-4o, and Claude-3.5-sonnet) in complex mathematical comprehension and formula generation.

- We propose a plug-and-play methodology that only brings our fine-tuning modules to work when the formula is concerned. This means that the performance for textual queries would not be affected by our fine-tuning process, which is novel in parameter-efficient fine-tuning approaches.

2 Related Work

2.1 Datasets

In recent years, the availability of specialized datasets in various mathematical fields has played an irreplaceable role in advancing cutting-edge research in areas such as economics and sociology. One notable example is im2latex-100k, which contains $100,000$ images of mathematical formulas along with their corresponding LaTeX code. This dataset aims to facilitate the image-to-text generation task for handwritten mathematical formulas and is widely used to train and evaluate formula recognition. A similar effort is FormulaNet(Schmitt-Koopmann et al., 2022), a high-quality dataset for mathematical formula detection (MFD)(Zanibbi and Blostein, 2012), which makes significant progress in annotating simple formulas. The dataset MATH(Hendrycks et al., 2021) contains nearly $10,000$ mathematical competition problems with detailed solutions, aimed at mathematical reasoning and explanation. The Math23K(Ling et al., 2017) dataset is used for training and evaluating machine learning models to automatically solve mathematical problems, particularly in expression reasoning and formula generation. Ape210K(Zhao et al., 2020), a large-scale collection of Chinese elementary school math problems, far exceeds Math23K in size and offers a higher diversity of problem templates. Compared with Math23K, Ape210K requires the model to have common knowledge. Similarly, AQuA(Ling et al., 2017), which generates question-answer reasoning tasks, helps models progressively derive the final answer to mathematical problems, aiding in the learning of arithmetic programs. MathQA(Amini et al., 2019) extends AQuA by introducing a new representational language to accurately model the operational procedures of each problem, improving both performance and interpretability. These datasets are still unable to share a focus on deep mathematical understanding and are not capable of supporting formula generation tasks. In contrast, we propose **ContextFormulas50K**(**CF50K**), specifically designed for formula generation through context understanding. Our dataset aims to prepare LLMs for mathematical comprehension based on textual descriptions. To our best knowledge, our dataset is the first of its own type, providing a benchmark for future studies.

2.2 Parameter-Efficient Fine-Tuning

As the scale of pre-trained language models expands rapidly, parameter-efficient fine-tuning (PEFT) has become a key technique to reduce the high computational and storage costs of training large models. Traditional fine-tuning requires adjusting all model parameters, which incurs significant overhead. To address this, various PEFT methods have been proposed to minimize resource usage while preserving model performance. Adapter(Houlsby et al., 2019) inserts lightweight, learnable modules in each layer, adjusting only the parameters of the inserted modules to avoid full parameter fine-tuning costs. AdapterFusion(Pfeiffer et al., 2021) combines several Adapters, enhancing adaptability for multi-task learning. Prefix-tuning(Li and Liang, 2021b) manipulates pre-designed prefixes to control model behavior. Other approaches, such as Prompt Tuning(Lester et al., 2021), optimize input prompts

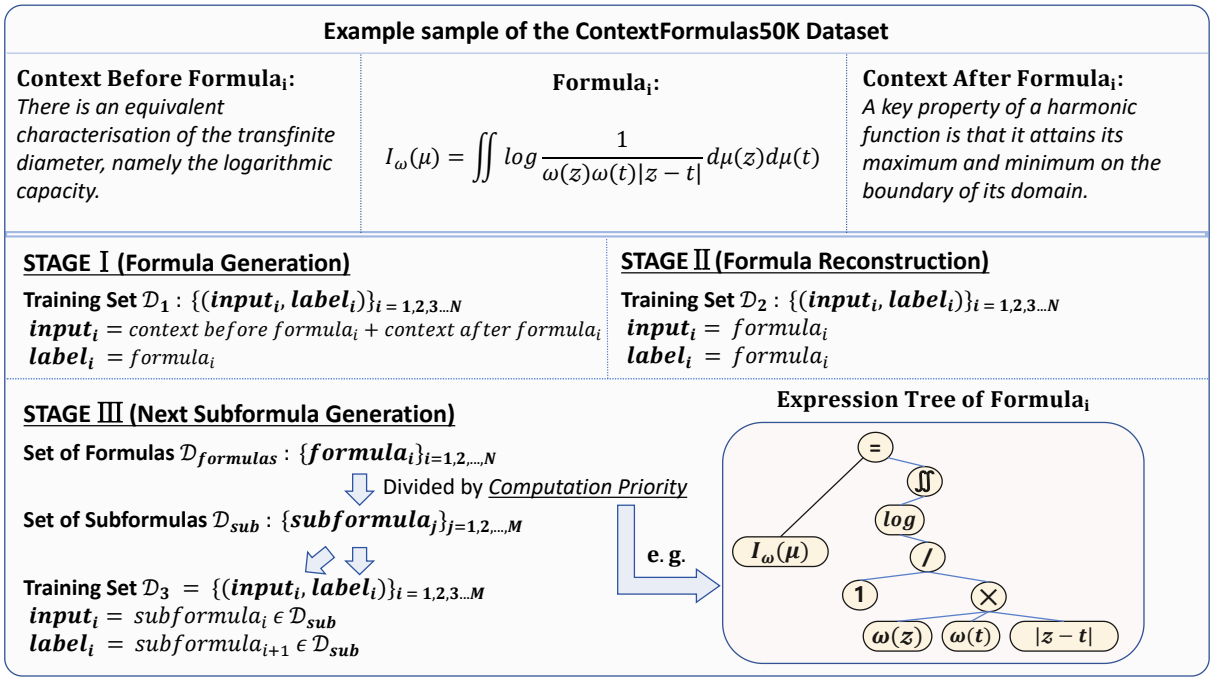

Figure 1: **ContextFormula50K(CF50K) Example**. As shown in the part above the double horizontal line, each sample in **CF50K** is a triplet *(context before formula, formula, context after formula)*, where the i-th sample is displayed. For the part below the double horizontal line, the diagram shows how the **CF50K** samples are transformed into different training sets in each stage of **MSFT**. Each training set consists of $(input_i, label_i)$ pairs based on the objectives of that stage.

instead of model parameters, and BitFit(Ben Zaken et al., 2022) adjusts only the bias parameters for lower overhead. LoRA(Hu et al., 2021) uses low-rank decomposition of weight matrices, adjusting only a small portion of parameters, reducing fine-tuning costs while improving efficiency and performance. The variants like QLoRA(Dettmers et al., 2024) further optimize memory and computation through quantization, while HyperLoRA(Lv et al., 2024) refines the low-rank strategy for better results. MixLoRA(Shen et al., 2024) achieves a zero-shot generalization improvement by dynamically adjusting low-rank adaptation matrices to address task interference in multimodal tasks. LoRA-Flow(Wang et al., 2024) improves the adaptability and performance of LoRA in generative tasks by using dynamic fusion weights. LoRAMoE(Dou et al., 2024) uses low-rank adapters and a router network to enhance the performance of downstream tasks while preventing the forgetting of world knowledge. These innovations further demonstrate the high potential and efficiency of PEFT, particularly in large-scale pre-trained models, offering cost-effective solutions for a wide range of tasks. In this paper, we propose a novel fine-tuning paradigm, **Multi-Stage Fine-Tuning** (**MSFT**), applied to the **CF50K** dataset. This approach trains plug-and-play modules to alleviate information bottlenecks during training, for better mathematical comprehension.

3 ContextFormulas50K Dataset

To obtain a large-scale and high-quality set of mathematical formulas, we choose to use the bulk download method through the official AWS(Amazon Web Services) servers to retrieve LaTeX source papers from the open-access platform arXiv. Then, we carefully select $5,000$ mathematical papers that are rich in formulas. Therefore, the samples in our dataset are more authentic and accurate.

After acquiring the LaTeX source code, we apply regex matching to extract mathematical formulas. For example, these patterns include the beginning and ending tags such as `\begin{equation}` and `\end{equation}`, `\begin{align}` and `\end{align}`. To obtain the context of each formula, we annotate the context of each formula as the text around this formula. If there is no other formula in this paragraph, we define the entire paragraph as the context, or we

define the text from the formula to the previous or next formula as the context, which may cause the context of the formula to overlap. To create a high-quality dataset, we exclude samples with context lengths shorter than 32 tokens, which cannot provide sufficient information to understand one formula. Thus, each sample in our dataset consists of a triplet: *(context before formula, formula, context after formula)*, as shown in Figure 1. In addition, we analyze the co-occurrence of operators as shown in Appendix B by using a heat map of the conditional probabilities.

4 Multi-Stage Fine-Tuning

Initially, we designed our model **Multi-Stage Fine-Tuning**(**MSFT**) based on LoRA, see Figure 2(a). However, LoRA presents **information bottleneck problem**. To solve this problem, we introduce two additional sequential fine-tuning stages, namely **Dual-plugin** and **Quad-plugin**.

4.1 Dual-Plugin

Our goal is to alleviate the information bottleneck encountered in traditional parameter-efficient fine-tuning. To achieve this, we propose the **Dual-plugin** (**D-plugin**, see Figure 2(b)). The introduction of this module does not alter the parameters of the first-stage trained modules but inserts a pair of new weight matrices between two original projection matrices of LoRA, which constitute our **D-plugin**. A training set is composed of pairing the formulas, $\mathcal{D}_2 : \{(x_i, y_i)\}_{i=1,..,N}$, where x_i and y_i represent the formula of the same sample in our dataset(see Figure 1), and N denotes the total number of samples.

We demonstrate LoRA in Figure 2(a) and label the parameter matrices as $L_{up} \in \mathbb{R}^{d\times r_1}$ and $L_{down} \in \mathbb{R}^{r_1\times d}$, where r_1 is the rank of the LoRA module and d is the dimension of h_{att}. For each hidden state $h_{att} \in \mathbb{R}^d$ after attention and concatenation in Transformer (Vaswani, 2017), it undergoes a matrix transformation during forward propagation(see Figure 2(b)), as shown in the following equation:

$$h_{out} = W_{pre}h_{att} + L_{up}D_{up}D_{down}L_{down}h_{att} \quad (1)$$

where h_{out} represents the new hidden state after passing through the concatenated modules, and $W_{pre} \in \mathbb{R}^{d\times d}$ denotes the pre-trained weight matrix that is concatenated, $D_{up} \in \mathbb{R}^{r_1\times r_2}$ and $D_{down} \in \mathbb{R}^{r_2\times r_1}$ represent the pair of projection matrices inserted into the L_{up} and L_{down} weight matrices during this stage, with r_2 is the intermediate dimension of **D-plugin**.

The second stage aims at formula structure. During the training of **D-plugin**, we freeze the parameters of the base model and the LoRA modules in our first stage. We only train the **D-plugin** projection matrices. This allows us to achieve a plug-and-play module by training only the new parameters θ_D, without affecting the performance of any previous work. Therefore, we set the auto-regressive formula generation task as the training objective. The objective is to optimize the following formula:

$$\max_{\theta_D} \sum_{(x,y)\sim\mathcal{D}_2} \sum_{t=1}^{|y|} \log\left(P_{\Phi_1+\Delta\Phi(\theta_D)}\left(y_t \mid x, y_{<t}\right)\right) \quad (2)$$

where $\mathcal{D}_2$ is our constructed pair dataset(see the **STAGE II** section of figure 1) and Φ_1 means all model parameters, including LoRA modules, after first-stage fine-tuning. For all other symbols and process, we follow (Hu et al., 2021), where we freeze Φ_1 and only fine-tuning θ_D.

4.2 Quad-Plugin

Despite the performance improvements from our two-stage fine-tuning, we intend to analyze the substructure of formulas based on computation priority. Thus, our model can generate more mathematically coherent formulas. By the word of computation priority, we mean that $\times$ shall be computed before $+$ normally in a mathematical formula. Therefore, we introduce the **Quad-Plugin** (**Q-plugin**) module, which consists of another pair of weight matrices, $Q_{down} \in \mathbb{R}^{r_3\times r_2}$ and $Q_{up} \in \mathbb{R}^{r_2\times r_3}$ between the **D-plugin** with a residual connection. **Q-plugin** encodes the priority information of the formula into the final stage of fine-tuning. We transform the formulas in our sample into sequences based on computation priority, and then we formulate our training objective as the Seq2Seq learning framework based on the sequences. Thus, we prepare the data $\mathcal{D}_3 : \{(z_i, z_{i-1})\}_{i=1,\cdots,M}$, where it means the next part z_i of a formula and its preceding part z_{i-1}(see Figure 1).

The forward propagation path(see Figure 2(c)) for each hidden state h_{att} after attention and concatenation in Transformer (Vaswani, 2017) is as follows:

$$h_D = D_{down}L_{down}h_{att} \quad (3)$$

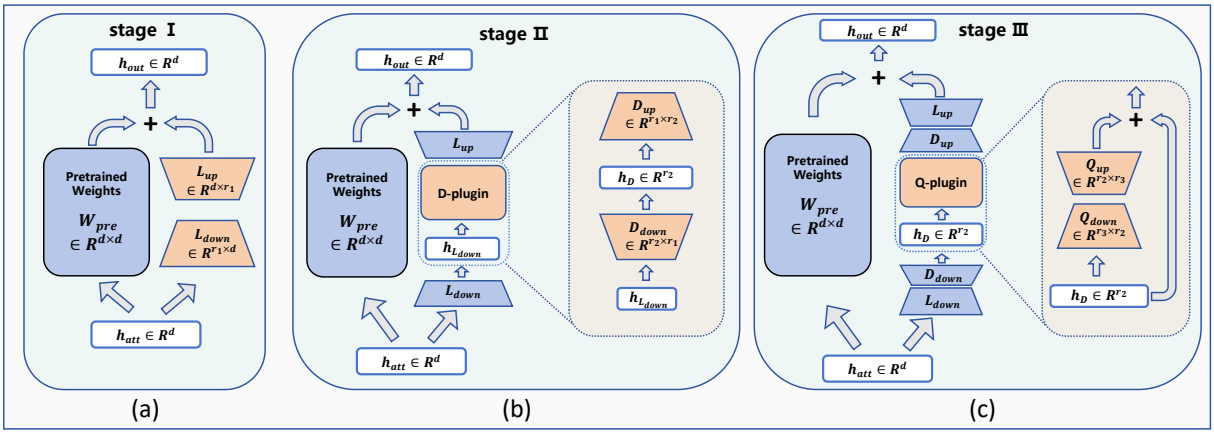

Figure 2: Schematic diagram of **Multi-Stage Fine-Tuning** (**MSFT**). The deep blue modules in the figure are frozen during fine-tuning, while the orange modules are the ones to be trained. W_{pre} represents the pre-trained module. (a) In the first stage, **LoRA** fine-tuning introduces two projection matrices, L_{up} and L_{down}, which are parallel to the pre-trained module. (b) In the second stage, our designed **D-plugin** is inserted between the original **LoRA** layers, consisting of a pair of new weight matrices, D_{up} and D_{down}. (c) In the third stage, the **Q-plugin** is added via a residual connection between the **D-plugin**, composed of another pair of new matrices, Q_{down} and Q_{up}. Additionally, h_{att}, $h_{L_{down}}$, h_D, and h_{out} represent the hidden states of each token after the matrix transformations.

$$h_{out} = W_{pre}h_{att} + L_{up}D_{up}(h_D + Q_{up}Q_{down}h_D) \tag{4}$$

where $h_D \in \mathbb{R}^{r_2}$ is the hidden state after passing through the L_{down} module of **LoRA** and the upward projection matrix D_{down} of **D-plugin**. In Equation 4, the term inside the parentheses represents the residual connection(He et al., 2016) applied to h_D. The meanings of the remaining symbols are the same as those in the previous subsection, and the training objective is also the same:

$$\max_{\theta_Q} \sum_{(x,y)\sim\mathcal{D}_3} \sum_{t=1}^{|y|} \log\left(P_{\Phi_2+\Delta\Phi(\theta_Q)}\left(y_t \mid x, y_{<t}\right)\right) \tag{5}$$

where $\mathcal{D}_3$ refers to the dataset we constructed (as shown in the **STAGE III** part of Figure 1) and θ_Q represents the parameters introduced in this stage. Φ_2 represents all model parameters, including LoRA modules and **D-plugin**, after two-stage fine-tuning. For all other symbols and process, we follow (Hu et al., 2021), where we freeze Φ_2 and only fine-tuning θ_Q.

5 Experiments

In this section, we outline a series of experiments performed on the **CF50K** dataset, aimed at generating mathematical formulas. Our experiments based on two pre-trained models with our method **MSFT** reveal the effectiveness of our model. Additionally, we perform ablation experiments to further validate the effectiveness of our approach.

5.1 Setup

The task involves providing the model with context to generate a mathematical formula. The complex structure of mathematical formulas often leads to longer training times in resource-constrained environments. To address this, we construct a smaller dataset, **CF5K**, which is also used for performance validation and ablation experiments. We conduct three stages of fine-tuning on the full **CF50K** dataset, based on two pre-trained models, namely LLaMA3-8B-Instruct and Mistral-7B. Our fine-tuning hyperparameters vary across different stages, due to different training objectives. Please refer to our Appendix C for details. In the first stage, we apply the **PEFT** method LoRA, illustrated in Figure 2(a). To strike a balance between fine-tuning performance and the parameter scale, we explore the rank r_1 illustrated in Figure 2(a) from the set {8, 16, 32, 64} and we find the setting of $r_1 = 64$ achieves the best performance. We choose the scale factor, a special hyper-parameter of LoRA, from the set {8, 16, 32, 64, 128}, with 128 chosen as the best setting. The second stage introduces **D-plugin**. The internal dimension r_2 of **D-plugin** illustrated in Figure 2(b) is set to 512 chosen from {64,128,256,512}. In the third stage, we use the identical experimental setup but with

Model	Trainable Param.	Param. New/Total	Evaluation Metrics				
			BLEU(↑)	sBLEU(↑)	Rouge-1(↑)	WER(↓)	CER(↓)
LLaMA3-8B-Insruct							
Baseline	-	-	$7.95_{\pm 0.01}$	$7.01_{\pm 0.04}$	$0.15_{\pm 0.01}$	$5.67_{\pm 0.01}$	$2.45_{\pm 0.01}$
w/ LoRA	55M	0.67%	$10.20_{\pm 0.10}$	$10.01_{\pm 0.06}$	$0.20_{\pm 0.01}$	$4.32_{\pm 0.07}$	$2.30_{\pm 0.05}$
w/ LoRA+D-plugin	63M	0.78%	$12.78_{\pm 0.04}$	$11.85_{\pm 0.07}$	$0.23_{\pm 0.01}$	$3.24_{\pm 0.03}$	$1.67_{\pm 0.03}$
w/ LoRA+D-plugin+Q-plugin	67M	0.83%	$13.05_{\pm 0.04}$	$12.04_{\pm 0.07}$	$0.24_{\pm 0.00}$	$3.15_{\pm 0.04}$	$1.61_{\pm 0.03}$
Mistral-7B							
Baseline	-	-	$5.80_{\pm 0.03}$	$6.73_{\pm 0.03}$	$0.15_{\pm 0.00}$	$9.69_{\pm 0.05}$	$5.20_{\pm 0.01}$
w/ LoRA	55M	0.75%	$13.32_{\pm 0.11}$	$10.21_{\pm 0.04}$	$0.22_{\pm 0.00}$	$2.60_{\pm 0.02}$	$1.63_{\pm 0.09}$
w/ LoRA+D-plugin	63M	0.86%	$18.81_{\pm 0.02}$	$15.77_{\pm 0.16}$	$0.32_{\pm 0.00}$	$1.24_{\pm 0.01}$	$0.83_{\pm 0.01}$
w/ LoRA+D-plugin+Q-plugin	67M	0.92%	$\mathbf{20.04}_{\pm \mathbf{0.04}}$	$\mathbf{16.60}_{\pm \mathbf{0.10}}$	$\mathbf{0.33}_{\pm \mathbf{0.01}}$	$\mathbf{1.26}_{\pm \mathbf{0.01}}$	$\mathbf{0.85}_{\pm \mathbf{0.01}}$
Mainstream Commercial Models							
GPT-4	-	-	12.50	7.16	0.18	2.48	1.28
GPT-4o	-	-	13.90	8.05	0.21	2.23	1.18
Claude-3.5-sonnet	-	-	14.09	9.05	0.22	1.58	1.00

Table 1: Model Performance Comparison. The table presents the performance of LLaMA3-8B-Insruct and Mistral-7B after sequentially inserting the LoRA, **D-plugin**, and **Q-plugin** modules. The symbol "**w/**" indicates the insertion of the module. BLEU, sBLEU, Rouge-1, WER, and CER are chosen as evaluation metrics, with each experiment repeated five times to report the mean and standard deviation. Additionally, commercial models GPT-4, GPT-4o, and Claude-3.5-sonnet are included for comparison, and the results of a single experiment are also shown.

an internal dimension r_3 of the **Q-plugin** set to 32, see Figure 2(c).

5.2 Results

A set of assessment metrics is required to evaluate the performance of formula generation. To our best knowledge, there is currently no perfect method for accurately measuring performance. Therefore, we adopt the approach from MathBridge (Jung et al., 2024), utilizing five classic evaluation metrics for natural language generation: BLEU, sBLEU(sacreBLEU), Rouge-1, WER (Word Error Rate), and CER (Character Error Rate). As shown in Table 1, we present the initial performance evaluations of two pre-trained models: LLaMA3-8B-Instruct and Mistral-7B, along with the evaluation results after each stage of the **MSFT**. We list the number of parameters required for training at each fine-tuning stage and their respective proportions relative to the total parameter scale.

Our MSFT method demonstrates significant performance improvements in each fine-tuning stage. Taking LLaMA3-8B-Instruct as an example, after the initial LoRA fine-tuning, the performance shows an preliminary improvement, with its average BLEU scoring 10.20 and sBLEU reaching 10.01. Following the insertion of the trained **D-plugin**, the BLEU and sBLEU scores increased by 25% and 43%, respectively, and the WER and CER decreased by 23% and 26%, respectively. After the third stage of fine-tuning, with the insertion of **Q-plugin**, all metrics have improved further and the evaluation metrics are now close to those of commercial models(e.g., GPT-4). Mistral-7B provides the same insights as our first base model and, after applying **MSFT**, achieves BLEU and sBLEU scores 2.45× and 1.46× higher than its baseline, outperforming even GPT-4o and Claude-3.5-sonnet, both of which are known for their strong mathematical reasoning capabilities.

The D-plugin and Q-plugin effectively address the information bottleneck problem encountered during LoRA fine-tuning. In the LoRA fine-tuning stage, we observe that when the intermediate dimension of the LoRA module reaches 64, further increasing this hyperparameter causes the model size to grow much faster than the improvement in performance, thus resulting in the **information bottleneck problem**. However, after introducing the **D-plugin**, which only accounts for 17% of the total parameters added in the LoRA stage, a noticeable performance improvement is achieved. Despite this, **D-plugin** still encounters the **information bottleneck problem**. Adding the **Q-plugin**, which contributes only 7% of the total parameters introduced by the LoRA module, further improves the performance. Ultimately, the final BLEU and sBLEU scores increase by 28% and 20%, respectively, compared to the LoRA stage, successfully mitigating the **information bottleneck problem**.

D-plugin (r=512)	D-plugin (r=640)	Q-plugin (r=32)	TFS	Trainable Param.	Evaluation Metrics				
					BLEU(↑)	sBLEU(↑)	Rouge-1(↑)	WER(↓)	CER(↓)
✗	✗	✗	✗	55M	$10.93_{\pm 0.10}$	$10.78_{\pm 0.12}$	$0.22_{\pm 0.00}$	$3.92_{\pm 0.03}$	$2.13_{\pm 0.02}$
✓	✗	✗	✓	63M	$10.31_{\pm 0.22}$	$10.47_{\pm 0.28}$	$0.21_{\pm 0.00}$	$4.05_{\pm 0.08}$	$2.34_{\pm 0.04}$
✓	✗	✓	✓	67M	$11.67_{\pm 0.26}$	$11.03_{\pm 0.18}$	$0.22_{\pm 0.01}$	$3.07_{\pm 0.03}$	$1.73_{\pm 0.02}$
✗	✓	✗	✗	67M	$13.64_{\pm 0.19}$	$12.53_{\pm 0.11}$	$0.26_{\pm 0.10}$	$2.47_{\pm 0.14}$	$1.34_{\pm 0.02}$
✓	✗	✗	✗	63M	$14.64_{\pm 0.17}$	$12.86_{\pm 0.26}$	$0.27_{\pm 0.01}$	$2.25_{\pm 0.03}$	$1.26_{\pm 0.01}$
✓	✗	✓	✗	67M	$\mathbf{17.67}_{\pm \mathbf{0.23}}$	$\mathbf{13.86}_{\pm \mathbf{0.27}}$	$\mathbf{0.30}_{\pm \mathbf{0.01}}$	$\mathbf{1.62}_{\pm \mathbf{0.01}}$	$\mathbf{0.97}_{\pm \mathbf{0.03}}$

Table 2: Comparison of single-stage and multi-stage fine-tuning results. In this table, a "✓" under the **D-plugin**, and **Q-plugin** columns indicates whether the corresponding module was inserted, with the internal dimension of each module shown in parentheses below. All comparative experiments conducted here have undergone the first stage of **MSFT** with LoRA fine-tuning at a rank of 64. "**TFS**" stands for "training from scratch," indicating whether these modules were trained together from scratch, corresponding to single-stage fine-tuning. We evaluate model performance using BLEU, sBLEU, Rouge-1, WER, and CER metrics, and also compare the total number of parameters added in different configurations.

STAGE	Joint Learning	Evaluation Metrics		
		BLEU(↑)	sBLEU(↑)	Rouge-1(↑)
II	✓	$12.55_{\pm 0.23}$	$11.86_{\pm 0.25}$	$0.24_{\pm 0.00}$
II	✗	$14.64_{\pm 0.17}$	$12.86_{\pm 0.26}$	$0.27_{\pm 0.01}$
III	✓	$14.14_{\pm 0.48}$	$12.74_{\pm 0.14}$	$0.26_{\pm 0.01}$
III	✗	$\mathbf{17.67}_{\pm \mathbf{0.23}}$	$\mathbf{13.86}_{\pm \mathbf{0.27}}$	$\mathbf{0.30}_{\pm \mathbf{0.01}}$

Table 3: **Impact of Joint Learning (JL)**. The table presents the results of **JL** in the second and third fine-tuning stages, with the intermediate dimension of **D-plugin** set to 512 and **Q-plugin** to 32. Evaluation metrics include BLEU, sBLEU, and Rouge-1. Additional metrics and **JL** comparisons under different intermediate dimensions are provided in Appendix E.

5.3 Ablation Study

In this subsection, we choose LLaMA3-8B-Instruct as the base model on the small-scale **CF5K** dataset to verify that the performance improvement in formula generation is indeed attributed to the **MSFT**.

To demonstrate that stage-by-stage fine-tuning can address the **information bottleneck problem** and that our **D-plugin** and **Q-plugin** are plug-and-play, we conduct two types of comparative experiments. First, we designed a joint learning experiment that adds new modules during fine-tuning and updates the parameters along with those trained in previous stages. Specifically, in the second stage of fine-tuning, both the LoRA modules and the **D-plugin** are trained together. Similarly, in the third stage, LoRA modules, **D-plugin**, and the **Q-plugin** are trained simultaneously. The results of these experiments are shown in Table 3. Directly inserting the two modules into the model for evaluation results in significantly better performance across all metrics than joint learning. This demonstrates the plug-and-play capability of the **D-plugin** and **Q-plugin**.

We also conduct single-stage fine-tuning by inserting multiple modules into the base model at the beginning and training from scratch. In contrast to the previous experiment, in this experiment, we mainly compare single-stage fine-tuning and **multi-stage fine-tuning**, which means the performance of where both modules are trained together was evaluated against where **D-plugin** and **Q-plugin** were added sequentially. The results are shown in Table 2. They indicate that the independently trained **D-plugin** and **Q-plugin** modules in **MSFT** outperform the single-stage fine-tuning approach where both modules are trained together. Furthermore, we examine the scenario where the intermediate dimension of **D-plugin** is set to 640. Although not incorporating **Q-plugin** in, the total number of additional parameters in this setting is comparable to the configuration where **D-plugin** has an intermediate dimension of 512 with **Q-plugin** added. These findings demonstrate that the performance improvement introduced by the plug-and-play modules is not solely attributed to an increase in the parameter scale. In contrast, simply increasing the intermediate dimension of **D-plugin** leads to a decrease in performance, indicating the appearance of the **information bottleneck problem**. However, incorporating **Q-plugin** effectively mitigates this issue, resulting in a substantial performance boost.

5.4 Case Study

We select two samples from **CF50K** as examples and choose three outputs from the LLaMA3-8B-Instruct model to demonstrate its formula generation ability after **MSFT**. For each example, we also select the output from GPT-4o for comparison, as

Example 1 **Objective:** **Formula Generation** **Input:** *There is an equivalent characterisation of the transfinite diameter, namely the logarithmic capacity.A key property of a harmonic function is that it attains its maximum and minimum on the boundary of its domain.* **Label:** $I_\omega(\mu) = \iint log \frac{1}{\omega(z)\omega(t)\lvert z-t\rvert} d\mu(z)d\mu(t)$	**Model:** ***Llama3-8B-Instruct(after MSFT)*** **Output1:** $\alpha_\Omega = \frac{1}{2} inf \left\{ \iint \frac{d\mu_1(x)d\mu_2(y)}{x-y^d} : \mu_1, \mu_2 \in \mathcal{M}_+^1(\Omega) \right\}$ **Output2:** $u(x) = \frac{1}{2\pi} \iint f(y)\, log \frac{1}{\lvert x-y\rvert} dy$ **Output3:** $f(0) = \int_0^r \int_0^s \frac{h(0)}{s} dsdr$ **Model:** ***GPT-4o*** **Output:** $\lim_{\varepsilon \to 0} \int log\lvert z-\zeta\rvert d\sigma(\zeta) = 0$
Example 2 **Objective:** **Formula Generation** **Input:** *For k = 3 this gives* $r_{GHZ\to W} \approx 6.17$*.Then* ψ *can be asymptotically transformed into* φ *with rate 1 and strong converse exponent at most* $r =- log\ q$*.* **Label:** $A_j = \sum_{i \in I_j} z^{u_j(i)} i$	**Model:** ***Llama3-8B-Instruct(after MSFT)*** **Output1:** $\psi = \frac{1}{\sqrt{q}} \sum_{i=0}^{q-1} i$ **Output2:** $\psi = \sum_{i=0}^{q-1} \alpha_i i$ **Output3:** $\mathcal{N}(\rho) = \sum_i p_i(\sqrt{1-r}\rho + \sqrt{r}\lvert\psi\rangle\langle\psi\rvert)$ **Model:** ***GPT-4o*** **Output:** $r_{GHZ\to W} \approx -log\ q$

Figure 3: **Outputs of two CF50K samples.** The figure presents the results from LLaMA3-8B-Instruct after applying **MSFT** on two different samples. Each sample shows three outputs, with the output of GPT-4o used for comparison. More formula generation results and output formulas from the original pre-trained model can be found in Appendix F.

shown in Figure 3.

The MSFT method helps the pre-trained model generate more accurate mathematical operators. In sample 1, the operators $\iint$, log and the reciprocal operation, as well as the $\sum$ and the multiplication operator in sample 2, all appear in the three outputs of the LLaMA3-8B-Instruct model after **MSFT**. Although there are differences in these output formulas, they all include the operators mentioned above. This is because the **MSFT** method helps the model learn more detailed internal structures of formulas, including operators, enabling it to better capture the operator information embedded in the context when generating formulas.

When the input text contains mathematical formulas, GPT-4o might overlook implicit mathematical symbols in context. In sample 1, where no explicit mathematical symbols appear, GPT-4o generates a formula close to the label. However, in sample 2, where the input context contains mathematical formulas like $r_{GHZ\to W} \approx 6.17$ and $r = -\log q$, GPT-4o tends to produce an output like $r_{GHZ\to w} \approx -\log q$. In this case, the mathematical symbols in the output are mostly derived from the explicit formulas in the context. The presence of these explicit mathematical formulas makes GPT-4o pay more attention to the mathematical symbols appearing in the context, while potentially neglecting the hidden mathematical symbols embedded within the text. More formula generation results and output formulas from the original pre-trained model can be found in Appendix F.

6 Conclusion

We introduce (**ContextFormulas50K**)**CF50K**, a mathematical formula dataset with contexts, and propose **Multi-Stage Fine-Tuning** (**MSFT**) to generate formulas based on context. **MSFT** progressively trains plug-and-play **D-plugin** and **Q-plugin** modules to mitigate the **information bottleneck problem**, improving formula generation performance. Extensive evaluations were conducted on two pre-trained models, including performance assessments and ablation studies. The results demonstrate that these plug-and-play modules significantly enhance model performance, with LLaMA3-8B-Instruct and Mistral-7B even achieving results comparable to GPT-4, GPT-4o, and Claude-3.5-sonnet, validating the effectiveness of **MSFT**. We expect our contributed **CF50K** dataset to serve as a new benchmark to evaluate formula generation capabilities with our novel method **MSFT**.

Limitations

Our experiments confirm that pre-trained models significantly improve their ability to generate well-formed mathematical formulas after training in the **CF50K** dataset using the **MSFT** approach. However, due to time and resource constraints, we have not yet explored the applicability of **MSFT** to other downstream NLP tasks, which remains a direction for future research. Furthermore, since **CF50K** is derived from cutting-edge mathematical literature, models trained on this dataset can achieve greater improvements in formula generation within advanced mathematical domains compared to basic mathematical problems. In the future, we plan to expand the scope and scale of **CF50K** to address this limitation.

References

Aida Amini, Saadia Gabriel, Shanchuan Lin, Rik Koncel-Kedziorski, Yejin Choi, and Hannaneh Hajishirzi. 2019. MathQA: Towards interpretable math word problem solving with operation-based formalisms. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 2357–2367, Minneapolis, Minnesota. Association for Computational Linguistics.

Elad Ben Zaken, Yoav Goldberg, and Shauli Ravfogel. 2022. BitFit: Simple parameter-efficient fine-tuning for transformer-based masked language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1–9, Dublin, Ireland. Association for Computational Linguistics.

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.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. Qlora: Efficient finetuning of quantized llms. *Advances in Neural Information Processing Systems*, 36.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Ning Ding, Yujia Qin, Guang Yang, Fuchao Wei, Zonghan Yang, Yusheng Su, Shengding Hu, Yulin Chen, Chi-Min Chan, Weize Chen, et al. 2023. Parameter-efficient fine-tuning of large-scale pre-trained language models. *Nature Machine Intelligence*, 5(3):220–235.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Wei Shen, Limao Xiong, Yuhao Zhou, Xiao Wang, Zhiheng Xi, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. LoRAMoE: Alleviating world knowledge forgetting in large language models via MoE-style plugin. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1932–1945, Bangkok, Thailand. Association for Computational Linguistics.

Robert M French. 1999. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

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

Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. 2019. Parameter-efficient transfer learning for nlp. In *International conference on machine learning*, pages 2790–2799. PMLR.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Kyudan Jung, Sieun Hyeon, Jeong Youn Kwon, Nam-Joon Kim, Hyun Gon Ryu, Hyuk-Jae Lee, and Jaeyoung Do. 2024. Mathbridge: A large corpus dataset for translating spoken mathematical expressions into *latex* formulas for improved readability. *arXiv preprint arXiv:2408.07081*.

James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526.

Brian Lester, Rami Al-Rfou, and Noah Constant. 2021. The power of scale for parameter-efficient prompt tuning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 3045–3059, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021a. Prefix-tuning: Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online. Association for Computational Linguistics.

Xiang Lisa Li and Percy Liang. 2021b. Prefix-tuning: Optimizing continuous prompts for generation. *arXiv preprint arXiv:2101.00190*.

Wang Ling, Dani Yogatama, Chris Dyer, and Phil Blunsom. 2017. Program induction by rationale generation: Learning to solve and explain algebraic word problems. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 158–167, Vancouver, Canada. Association for Computational Linguistics.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *Preprint*, arXiv:1907.11692.

Chuancheng Lv, Lei Li, Shitou Zhang, Gang Chen, Fanchao Qi, Ningyu Zhang, and Hai-Tao Zheng. 2024. HyperLoRA: Efficient cross-task generalization via constrained low-rank adapters generation. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 16376–16393, Miami, Florida, USA. Association for Computational Linguistics.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Jonas Pfeiffer, Aishwarya Kamath, Andreas Rücklé, Kyunghyun Cho, and Iryna Gurevych. 2021. AdapterFusion: Non-destructive task composition for transfer learning. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 487–503, Online. Association for Computational Linguistics.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Felix M Schmitt-Koopmann, Elaine M Huang, Hans-Peter Hutter, Thilo Stadelmann, and Alireza Darvishy. 2022. Formulanet: A benchmark dataset for mathematical formula detection. *IEEE Access*, 10:91588–91596.

Ying Shen, Zhiyang Xu, Qifan Wang, Yu Cheng, Wenpeng Yin, and Lifu Huang. 2024. Multimodal instruction tuning with conditional mixture of LoRA. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 637–648, Bangkok, Thailand. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

Hanqing Wang, Bowen Ping, Shuo Wang, Xu Han, Yun Chen, Zhiyuan Liu, and Maosong Sun. 2024. LoRA-flow: Dynamic LoRA fusion for large language models in generative tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 12871–12882, Bangkok, Thailand. Association for Computational Linguistics.

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.

Richard Zanibbi and Dorothea Blostein. 2012. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 15:331–357.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2024. A survey of large language models. *Preprint*, arXiv:2303.18223.

Wei Zhao, Mingyue Shang, Yang Liu, Liang Wang, and Jingming Liu. 2020. Ape210k: A large-scale and template-rich dataset of math word problems. *arXiv preprint arXiv:2009.11506*.

A Ethics Statement

Our dataset samples are sourced from the latest scientific literature on arXiv. We obtained these documents through bulk download via the Amazon Web Services(AWS) platform, a paid service officially recommended by arXiv as a legitimate method for accessing full-text papers. Furthermore, since our dataset is derived from arXiv, a reputable pre-print platform, and we have implemented sensitive word detection, we ensure that our dataset contains no sensitive content, and the proposed **MSFT** approach does not introduce any social risks or negative impacts.

B Mathematical Symbols of CF50K

To obtain a clear overall understanding of **CF50K**, we conducted a statistical analysis of symbols, including operators and special characters, in the mathematical formula of each sample. We analyzed the co-occurrence of operators as shown in Figure 4. Due to the large variety of symbols in the dataset, the figure does not display all mathematical symbols. Instead, we selected the most frequent and representative ones. Using a heat map of conditional probabilities, we observe the relationships between different symbols. The symbols $=$, $+$, and $-$ appear much more frequently with other symbols than other mathematical symbols, which aligns with typical mathematical formulas. This further demonstrates the validity of our dataset.

C Experimental Details

We conduct all experiments on the PyTorch(Paszke et al., 2019) framework, with all pre-trained models sourced from the Hugging Face(Wolf et al., 2020) platform. In each training phase of **MSFT**, we use the AdamW optimizer and a learning rate scheduler with cosine decay and a warm-up period of one epoch, starting with an initial learning rate of 10^{-4}, and a warm-up period of one epoch. The first two stages are trained for 10 epochs, while the third stage is trained for 20 epochs. Furthermore, the maximum input length of the model is set to 512, while the output length for mathematical formulas is set to 384. For LLaMA3-8B-Instruct and Mistral-7B, we integrated the **MSFT** method by adding the modules introduced to the original W_k, W_q, W_v, and W_o components of the Transformer(Vaswani, 2017) architecture. Apart from using **CF50K**, we also use a mini-version of this dataset, **CF5K**, in our experiments. And these two datasets do not have overlapping samples. The partition of the dataset for training, validation, and test sets in different stages of **MSFT** is shown in the table 4. Since each stage has distinct training objectives, the partitioning strategy differs even when the data originates from the same dataset. The upward projection matrix of **D-plugin** is initialized using Kaiming initialization (He et al., 2015) with $a = 0.01$, where a is the key hyper-parameter, while the downward projection matrix is initialized with zeros. In the third stage, both weight matrices are initialized using Kaiming initialization with $a = \sqrt{5}$.

Stage	Dataset	Train	Valid	Test
Main Experiment				
I	CF50K	47868	2519	1559
II	CF50K	47868	2519	1559
III	CF5K	56900	1161	495
Ablation Experiment				
I	CF5K	4755	250	495
II	CF5K	4755	250	495
III	CF5K	56900	1161	495

Table 4: **Experimental Dataset Partition.** We present the datasets used in both the main experiment and the ablation experiment, along with the partitioning of the training, validation, and test sets. Due to time and resource constraints, we created a smaller subset, **CF5K**, which has no sample overlap with **CF50K**.

All experiments are conducted on NVIDIA RTX A6000. To reduce the randomness in the model outputs, each experiment is repeated five times and we calculate the mean and standard deviation of the results. A full **MSFT** training on **CF50K** takes approximately one week.

D Pre-trained Models

In the experiments, we utilized two pre-trained models: LLaMA3-8B-Instruct and Mistral-7B. This section provides a brief introduction to these

STAGE	Intermediate Dimension	Joint Learning	Evaluation Metrics				
			BLEU(↑)	sBLEU(↑)	Rouge-1(↑)	WER(↓)	CER(↓)
II	64	✓	$9.10_{\pm 0.27}$	$9.09_{\pm 0.37}$	$0.19_{\pm 0.00}$	$5.14_{\pm 0.11}$	$2.67_{\pm 0.03}$
II	64	✗	$9.87_{\pm 0.24}$	$8.88_{\pm 0.28}$	$0.21_{\pm 0.00}$	$3.94_{\pm 0.20}$	$1.92_{\pm 0.07}$
II	128	✓	$9.46_{\pm 0.24}$	$9.57_{\pm 0.14}$	$0.20_{\pm 0.01}$	$4.67_{\pm 0.16}$	$2.35_{\pm 0.05}$
II	128	✗	$10.60_{\pm 0.29}$	$9.79_{\pm 0.14}$	$0.22_{\pm 0.01}$	$3.68_{\pm 0.11}$	$1.83_{\pm 0.04}$
II	256	✓	$10.63_{\pm 0.20}$	$9.30_{\pm 0.12}$	$0.20_{\pm 0.01}$	$3.64_{\pm 0.18}$	$1.89_{\pm 0.07}$
II	256	✗	$12.02_{\pm 0.36}$	$11.22_{\pm 0.37}$	$0.24_{\pm 0.01}$	$3.00_{\pm 0.17}$	$1.57_{\pm 0.06}$
II	512	✓	$12.55_{\pm 0.23}$	$11.86_{\pm 0.25}$	$0.24_{\pm 0.00}$	$2.89_{\pm 0.06}$	$1.57_{\pm 0.02}$
II	512	✗	$\mathbf{14.64_{\pm 0.17}}$	$\mathbf{12.86_{\pm 0.26}}$	$\mathbf{0.27_{\pm 0.01}}$	$\mathbf{2.25_{\pm 0.03}}$	$\mathbf{1.26_{\pm 0.01}}$
III	16	✓	$13.35_{\pm 0.27}$	$12.34_{\pm 0.17}$	$0.25_{\pm 0.01}$	$2.47_{\pm 0.02}$	$1.38_{\pm 0.03}$
III	16	✗	$16.64_{\pm 0.36}$	$13.49_{\pm 0.17}$	$0.29_{\pm 0.01}$	$1.74_{\pm 0.04}$	$1.02_{\pm 0.03}$
III	32	✓	$14.14_{\pm 0.48}$	$12.74_{\pm 0.14}$	$0.26_{\pm 0.01}$	$2.46_{\pm 0.07}$	$1.35_{\pm 0.04}$
III	32	✗	$\mathbf{17.67_{\pm 0.23}}$	$\mathbf{13.86_{\pm 0.27}}$	$\mathbf{0.30_{\pm 0.01}}$	$\mathbf{1.62_{\pm 0.01}}$	$\mathbf{0.97_{\pm 0.03}}$
III	64	✓	$13.66_{\pm 0.25}$	$12.20_{\pm 0.16}$	$0.26_{\pm 0.00}$	$2.44_{\pm 0.02}$	$1.37_{\pm 0.01}$
III	64	✗	$17.11_{\pm 0.21}$	$13.72_{\pm 0.15}$	$0.29_{\pm 0.01}$	$1.74_{\pm 0.08}$	$1.02_{\pm 0.03}$
III	128	✓	$13.40_{\pm 0.11}$	$12.33_{\pm 0.32}$	$0.26_{\pm 0.00}$	$2.47_{\pm 0.12}$	$1.41_{\pm 0.02}$
III	128	✗	$14.65_{\pm 0.26}$	$13.92_{\pm 0.11}$	$0.27_{\pm 0.01}$	$2.35_{\pm 0.02}$	$1.30_{\pm 0.03}$

Table 5: **Impact of Joint Learning (JL)**. The table presents the results of **JL** in the second and third fine-tuning stages. Evaluation metrics include BLEU, sBLEU, Rouge-1, WER, and CER. We compare the intermediate dimension r_2 of the **D-plugin** across $\{64, 128, 256, 512\}$ and the intermediate dimension r_3 of the **Q-plugin** across $\{16, 32, 64, 128\}$.

models.

- **LLaMA3-8B-Instruct**(Touvron et al., 2023) is a large-scale instruction-tuned language model developed by Meta, featuring 8 billion parameters. As an improved version of its predecessors, it has been trained with a focus on instruction-following tasks, leveraging both supervised fine-tuning and reinforcement learning. The model benefits from enhanced Transformer architecture and extensive pre-training on diverse, high-quality datasets. It has demonstrated strong performance in various natural language processing tasks, particularly in reasoning, code generation, and mathematical problem-solving, making it a powerful tool for research and practical applications.

- **Mistral-7B**(Jiang et al., 2023) is a powerful 7-billion-parameter language model developed by Mistral AI. We use the Mistral-7B-Instruct-v0.2 version. It is designed to achieve high efficiency and strong performance in various natural language processing tasks. Compared to other models of similar scale, Mistral-7B incorporates architectural optimizations that enhance its reasoning ability, fluency, and adaptability. Trained on a diverse and high-quality dataset, it excels in code generation, logical inference, and mathematical problem solving. Its balance between model size and computational efficiency makes it well suited for both research and real-world applications.

E Ablation Study

We conducted **Joint Learning**(**JL**) experiments on LLaMA3-8B-Instruct that progressively integrate new modules during fine-tuning while updating parameters alongside those optimized in earlier stages. Specifically, in the second fine-tuning stage, both the LoRA module and the **D-plugin** are jointly trained. Likewise, in the third stage, training is performed simultaneously on LoRA, **D-plugin**, and **Q-plugin**.

We compare the intermediate dimension r_2 of the **D-plugin** across $\{64, 128, 256, 512\}$ and the intermediate dimension r_3 of the **Q-plugin** across $\{16, 32, 64, 128\}$. The evaluation is carried out using BLEU, sBLEU, Rouge-1, WER, and CER. The results of these experiments are presented in Table 5. All performance metrics are observed to degrade after **JL** is applied, indicating the need to freeze the modules trained in previous stages at each step. This further validates the effectiveness of the **MSFT** approach.

F Case Study

In this section, we provide a detailed analysis of two sample outputs from **CF50K**. Using LLaMA3-8B-Instruct as the baseline, we present three outputs from the original model and five from **MSFT**-enhanced models, with GPT-4o outputs included for comparison, as shown in Figures 5 and Figures 6. Observing these results, we find that models fine-tuned with **MSFT** demonstrate a significantly improved ability to generate well-structured mathematical formulas, often closely matching the ground-truth labels. This validates the effectiveness of our dataset and the **MSFT** approach in supporting scientific research, particularly in the construction of mathematical formulas.

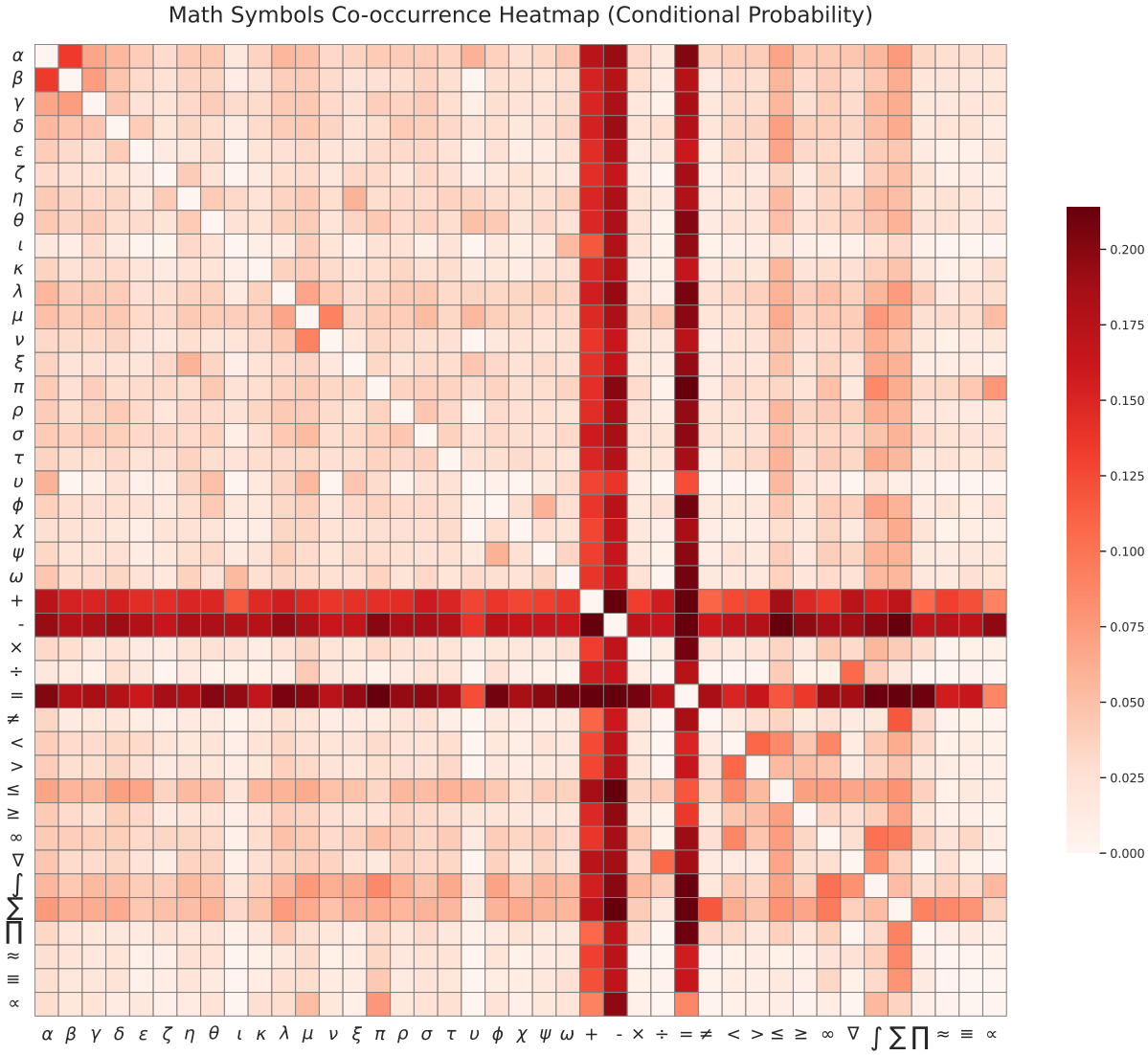

Figure 4: Co-occurrence heatmap of mathematical symbols in **CF50K**, which illustrates the relationship between mathematical symbols (including operators and unknown parameters). The heatmap illustrates the co-occurrence frequency of these symbols using their conditional probabilities. The deep red grid means a stronger correlation. For instance, the darkest red grids of the grids correspond to the symbols $=$, $+$, and $-$, which indicates that these symbols frequently co-occur with other symbols.

Example 1

Text before the formula: *There is an equivalent characterisation of the transfinite diameter, namely the logarithmic capacity.*

Formula: $I_\omega(\mu) = \iint log \frac{1}{\omega(z)\omega(t)|z-t|} d\mu(z)d\mu(t)$

Text after the formula: *A key property of a harmonic function is that it attains its maximum and minimum on the boundary of its domain.*

Objective: **Formula Generation**

Input: *There is an equivalent characterisation of the transfinite diameter, namely the logarithmic capacity. A key property of a harmonic function is that it attains its maximum and minimum on the boundary of its domain.*

Label: $I_\omega(\mu) = \iint log \frac{1}{\omega(z)\omega(t)|z-t|} d\mu(z)d\mu(t)$

Model: ***Llama3-8B-Instruct(before MSFT)***

Output1: $P_I(v) \propto e^{Mv}$

Output2: $d_\infty(D) = exp(-\frac{1}{C(D)})$

Output3: $u(z) = -logC(z,\Omega) + \frac{1}{2}D^2 log\left|\frac{D}{2}(\frac{1}{|z-z_0|})\right|$

Model: ***Llama3-8B-Instruct(after MSFT)***

Output1: $\alpha_\Omega = \frac{1}{2} inf\left\{\iint \frac{d\mu_1(x)d\mu_2(y)}{x-y^d} : \mu_1, \mu_2 \in \mathcal{M}_+^1(\Omega)\right\}$

Output2: $u(x) = \frac{1}{2\pi}\iint f(y)\, log \frac{1}{|x-y|} dy$

Output3: $f(0) = \int_0^r \int_0^s \frac{h(0)}{s} dsdr$

Output4: $u(x) = \frac{1}{2\pi}\iint log|z-x|\mu dz$

Output5: $\boldsymbol{\varkappa} = \frac{1}{4}\iint \log(\frac{1}{d(1,1)})\mu(1)\mu(1)$

Model: ***GPT-4o***

Output: $\lim_{\varepsilon\to 0}\int log|z-\zeta|d\sigma(\zeta) = 0$

Figure 5: **CF50K Sample Output 1**. This figure presents three outputs from the LLaMA3-8B-Instruct pre-trained model, along with five outputs enhanced by **MSFT**. Additionally, one output from GPT-4o is included for comparison.

Example 2

Text before the formula:
For k = 3 this gives $r_{GHZ\to w} \approx 6.17$.

Formula: $A_j = \sum_{i\in I_j} z^{u_j(i)} i$

Text after the formula:
Then ψ *can be asymptotically transformed into* φ *with rate 1 and strong converse exponent at most* $r =- log\ q$.

Objective: **Formula Generation**

Input: *For k = 3 this gives* $r_{GHZ\to w} \approx 6.17$. *Then* ψ *can be asymptotically transformed into* φ *with rate 1 and strong converse exponent at most* $r =- log\ q$.

Label: $A_j = \sum_{i\in I_j} z^{u_j(i)} i$

Model: ***Llama3-8B-Instruct(before MSFT)***

Output1: $\lim_{k\to\infty}\frac{1}{k} log\left|\Psi_{GHZ}^k(\psi^{\otimes k})\right|^2 = r$

Output2: $\lim_{k\to\infty}\frac{1}{k} log|\Psi_k(x) - \Phi_k(x)| \leq - logq$

Output3: $\sum_{i=1}^3 \theta_i \frac{\partial H}{\partial v_i}$

Model: ***Llama3-8B-Instruct(after MSFT)***

Output1: $\psi = \frac{1}{\sqrt{q}}\sum_{i=0}^{q-1} i$

Output2: $\psi = \sum_{i=0}^{q-1} \alpha_i i$

Output3: $\mathcal{N}(\rho) = \sum_i p_i(\sqrt{1-r}\rho + \sqrt{r}|\psi\rangle\langle\psi|)$

Output4: $\psi = \sum_{i=1}^q \lambda_i \psi_i$

Output5: $\psi \approx \left(\sum_{k=1}^q \sqrt{\lambda_k}\right)\varphi$

Model: ***GPT-4o***

Output: $r_{GHZ\to w} \approx - log\ q$

Figure 6: **CF50K Sample Output 2**. This figure presents three outputs from the LLaMA3-8B-Instruct pre-trained model, along with five outputs enhanced by **MSFT**. Additionally, one output from GPT-4o is included for comparison.