Enhancing Large Language Model Performance with Gradient-Based Parameter Selection

Haoling Li[1*], **Xin Zhang**[2†], **Xiao Liu**[2], **Yeyun Gong**[2], **Yifan Wang**[1], **Qi Chen**[2], **Peng Cheng**[2]

[1]Tsinghua University
[2] Microsoft Research
li-hl23@mails.tsinghua.edu.cn, xinzhang3@microsoft.com

Abstract

Large language models (LLMs) have revolutionized numerous fields of research, driving significant advancements in natural language processing, machine translation, and beyond. Although the extensive number of parameters contributes a lot to the great success, existing studies indicate that not all model parameters hold equal importance, which further leads to redundancy during the parameter update process. Recent works for reducing redundant parameter updates for LLMs either lack task-specific data information, may leading to suboptimal model performance, or discard transformer components or insignificant parameters, limiting the model's scalability across different tasks and potentially compromising the LLM structure. To address these issues and further enhance the performance of LLMs, we propose Gradient-Mask Tuning (GMT), a method that selectively updates parameters based on gradient information, which is specific to the target tasks. Specifically, after calculating gradients during back propagation, we measure their absolute values and mask those with small absolute values. Our empirical results in various training paradigms like SFT and DPO for various domains of tasks demonstrate that GMT not only preserves the original network structure but also enhances the potential performance of LLMs. Further analysis indicates that GMT exhibits insensitivity to mask ratio and possesses computational efficiency comparable to vanilla training approach.

Introduction

Large language models (LLMs) have pivoted the centerpiece of various tasks such as textual understanding, natural language planning and instruction following (Achiam et al. 2023; Touvron et al. 2023; Jiang et al. 2023; Wang et al. 2024b; Luo et al. 2024a), which can be attributed to their extensive number of parameters, with performance and unique abilities significantly enhancing as the number of parameters increases. To optimize the performance of LLMs in downstream tasks, full-parameter fine-tuning is commonly employed by researchers and practitioners.

However, a recent trend shows an observable phenomena that not all parameters of LLMs hold the same importance, which leads to redundancy during the parameter update process (Huang et al. 2024; Jiang et al. 2024; Yu et al. 2023b; Luo et al. 2024b). The existence of the redundancy of LLM parameter updates can be proven by several branches of works, like model pruning methods (Ma, Fang, and Wang 2023), fine-tuning methods dropping the delta parameters (Yadav et al. 2024; Yu et al. 2023b), and so on, which shows minimal impact on performance. Furthermore, inspired by recent works (Liu et al. 2024b; Lv et al. 2024), we believe that reducing redundancy in parameter updates has the potential to enhance model performance.

Existing methods for reducing the redundancy of LLM parameter updates can be broadly classified into two categories based on their implementation. The first involves sparsifying the network during the training process using criteria such as randomness (Woo et al. 2024; Hui et al. 2024). However, a significant drawback of these methods is their failure to leverage information from task-specific data, may leading to suboptimal optimization objectives (Chen et al. 2024). The second focuses on eliminating non-essential structural components of the transformer block (Men et al. 2024; Song et al. 2024) or discarding insignificant parameters (Yadav et al. 2024; Yu et al. 2023b). Yet, these one-off operations constrain the model's scalability across different tasks and impede its ability to adjust or reconfigure its computational complexity, which is crucial for adapting to varying task demands.

To address the above issues, a method that effectively identifies and retains the most critical LLM parameters during training is needed to reduce redundant updates and enhance overall model performance. In this work, we introduce the Gradient-Mask Tuning (GMT), a method that selects the set of parameters to be updated based on the gradient information associated with task-specific data. Specifically, after calculating the gradients in the backward process, we measure the absolute values of gradients and mask those with small values. There are several reasons for using gradient magnitude as a criterion. Firstly, gradients inherently capture task-specific information and can be utilized without additional computational overhead during training. Secondly, we assert that the gradient effectively assesses network parameter significance across different training datasets, allowing delicate control of the training process. Thirdly, identifying and removing insignificant subsets of gradients during the training process, which does

[*] Work done during his internship at Microsoft.
[†] Corresponding author.

not have a drastic impact on the convergence of the model, and may even lead to a regularization effect that improves the performance of the model (Hoefler et al. 2021).

To evaluate the effectiveness of GMT, we provide theoretical analysis and conduct experiments with both Supervised Fine-tuning (SFT) and Direct Preference Optimization (DPO) training paradigms on several widely-used benchmarks on math reasoning, code generation, and general domain using various base models. The experimental results demonstrate that GMT exploits the gradient information to identify more important network parameters for sparse updating with acceptable extra time spent. This approach enhances model performance and reduces redundant parameter updates compared to several established baseline methods, including vanilla fine-tuning, Drop, and RMT. Furthermore, it illustrates that GMT is not sensitive to mask ratio selection and is more effective than random parameter updating. We further analyze the computational FLOPs and time efficiency of GMT, demonstrating that it enables a plug-and-play replacement of vanilla SFT without destroying the network architecture.

Our contributions can be summarized as:

- We propose GMT, which masks the gradients with small absolute values and discriminates the importance of parameters during training, naturally utilizing the information of task-specific data. This approach effectively reduces redundant parameter updates to the LLM and enhances performance on downstream tasks.
- We conduct both theoretical analysis and exhaustive experiments with different base models on several benchmarks comparing with representative baselines. The results confirm the effectiveness of GMT.
- We further demonstrate the adaptability, robustness and efficiency of the GMT by analyzing the drop strategy, mask ratio and time efficiency.

Related Works

Reducing redundant parameter updates in LLMs is a significant research problem (Dalvi et al. 2020). Addressing this issue can prevent the waste of computational resources and optmize the model training process. We categorized the reduction of redundant parameter updates into post-processing and in-training based on the stage of implementation. This section describes the properties of the typical methods of both strategies as well as the respective drawbacks.

Post-processing Strategy The post-processing strategy, although requiring tuning with full parameter participation, achieves the same purpose as in-training through a series of tuned parameter drop and reset strategies. Inspired by dropout, mixout (Lee, Cho, and Kang 2019) stochastically mixes source and target parameters as a regularization technique and extends to fine-tune downstream tasks. DARE (Yu et al. 2023b) finds that setting most (90% or even 99%) of the delta parameters to zero does not diminish the ability to fine-tune the LLM, and extends to model fusion accordingly. Ties-merging (Yadav et al. 2024) describes the delta parameter as a type of task vector that trims it, and then averages the parameters that are identical for multiple model aggregation symbols to achieve model merging. ExPO (Zheng et al. 2024) is inspired by model interpolation (Wortsman et al. 2022) and obtains better aligned models by direct extrapolation from the parameters of the aligned model obtained by DPO or RLHF and the initial SFT model. Although these methods can be implemented directly on fine-tuned models that are open access and require little additional computational overhead, they never achieve optimal performance due to the lack of refinement in the training process. Some methods attempt to directly remove redundant layers (Men et al. 2024) or components of transformer blocks (Song et al. 2024) in LLMs based on quantitative criteria. However, these methods compromise the structural integrity of LLMs, hindering their applicability across a wider range of scenarios.

In-training Strategy The in-training strategies aim to dynamically mitigate the impact of model parameter redundancy during the training process. DropBP (Woo et al. 2024) calculates the sensitivity of each layer to allocate an appropriate dropout rate, thereby randomly dropping certain layers during backpropagation to improve efficiency and reduce redundancy. HFT (Hui et al. 2024) randomly selects half of parameters for learning new tasks while freezing the remaining parameters to preserve previously learned knowledge, thus eliminating the redundancy of parameter knowledge across different learning stages. PAFT (Pentyala et al. 2024) separately conducts the SFT and DPO training processes, using L1 regularization to sparsify delta parameters and reduce redundancy, thereby achieving alignment and parallel training. These methods either employ randomness or regularization during the training process and lack guidance from task-specific data. Our GMT is focused on the task-specific data information and fine-grained tuning of parameters, which not only preserves the model's generalizability but also significantly improves its adaptability and performance on specific tasks.

Methodology

In this section, we elucidate the principle underlying the GMT method, providing a comprehensive explanation through mathematical formulation and theoretical analysis. Figure 1 presents the framework of our method and compares it with a one-off drop strategy for delta parameters (i.e., the difference between the fine-tuned parameters and the pre-trained parameters). Specifically, GMT effectively identifies the most critical parameters to be updated during training by the absolute magnitude of the gradient. This dynamic selection strategy prevents redundant parameter updates, thereby improving the performance of LLM.

Gradient-Mask Tuning

To decrease redundant updates by leveraging the information of task-specific data during training, while preserving the integrity of the original LLM architecture, we propose GMT for selecting critical parameters.

For each batch of training data, we compute the gradient of the loss function $\mathcal{L}(\Theta)$ with respect to the parameters.

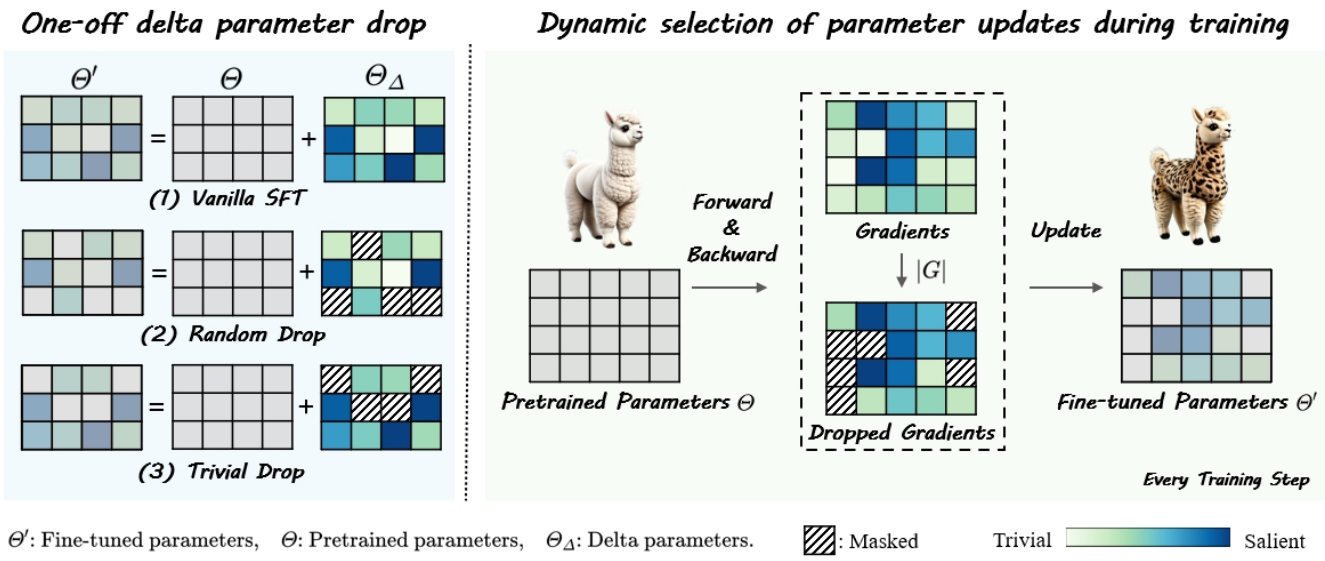

Figure 1: Illustration of our proposed method GMT, compared with the one-off drop delta parameters approach. The figure on the left delineates the distinction between a trivial drop and a random drop, wherein the trivial drop serves to diminish the redundant updates that arises during the fine-tuning process. Building upon this insight, we refine the training procedure by preferentially updating more significant parameters, as determined by the gradient information pertinent to the task-specific data. This selective updating is operationalized through the implementation of a masking strategy that filters out gradients with smaller absolute values.

To facilitate gradient accumulation, we define an accumulation interval N, representing the number of mini-batches over which gradients are accumulated before performing a parameter update. The accumulated gradient Γ_{ij} for each parameter θ_{ij} is defined as:

$$\Gamma_{ij} = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\theta_{ij}} \mathcal{L}(\Theta, \mathcal{B}_n) \tag{1}$$

where $\mathcal{B}_n$ represents the n-th mini-batch of data and $\nabla_{\theta_{ij}} \mathcal{L}(\Theta, \mathcal{B}_n)$ is the gradient of the loss function with respect to θ_{ij} for the mini-batch.

Upon accumulating the gradients over N mini-batches, we employ gradient information as a signal to identify the importance of parameters at the element-wise level in a fine-grained manner. This operation involves sorting the components of each accumulated gradient Γ_{ij} by their absolute values and selecting a pre-defined top percentile k for updating the parameters. The masked gradient $\mathcal{M}(\Gamma_{ij}, k)$ is calculated as:

$$\mathcal{M}(\Gamma_{ij}, k) = \{g_{ij} \mid g_{ij} \in \Gamma_{ij},\ |g_{ij}| \geq T_k\} \tag{2}$$

where g_{ij} represents the ij-th component of the accumulated gradient for parameter θ_{ij}. The threshold T_k is the value such that the absolute values of the components of Γ_{ij} that are greater than or equal to T_k fall within the top k percentile of all components by magnitude.

The subsequent parameter update step utilizes the masked gradient $\mathcal{M}(\Gamma_{ij}, k)$:

$$\theta_{ij}^{(t+1)} = \theta_{ij}^{(t)} - \eta \cdot \mathcal{M}(\Gamma_{ij}, k) \tag{3}$$

Algorithm 1: Gradient-Mask Tuning

Hyperparameters: The accumulation interval N, the training step T and the learning rate η.
Input: The initial model parameters $\Theta^{(0)}$ and the n-th mini-batch of data $\mathcal{B}_n$.
Output: The updated model parameters $\Theta^{(T)}$.

for t in $0 \rightarrow T-1$ **do**
  $\Gamma \leftarrow 0$
  for n in $1 \rightarrow N$ **do**
    $\Gamma \leftarrow \Gamma + \nabla_{\Theta} \mathcal{L}(\Theta^{(t)}, \mathcal{B}_n)$
  end for
  $\Gamma \leftarrow \frac{1}{N}\Gamma$
  $T_k \leftarrow$ Percentile threshold based on Γ
  $\mathcal{M}(\Gamma, k) \leftarrow \{g_{ij} \mid g_{ij} \in \Gamma_{ij},\ |g_{ij}| \geq T_k\}$
  for $\theta_{ij} \in \Theta$ **do**
    if $|\Gamma_{ij}| \geq T_k$ **then**
      $\theta_{ij}^{(t+1)} \leftarrow \theta_{ij}^{(t)} - \eta \cdot \Gamma_{ij}$
    end if
  end for
end for=0

where η is the learning rate and t indexes the current training step. The detailed algorithmic training procedure is given in Algorithm 1.

Theoretical Analysis

We present the theoretical analysis in two parts. First, we explain from the optimization perspective (Fu et al. 2023; Hui et al. 2024) why removing unimportant parameter up-

dates can enhance model performance, by fine-tuning partial parameters to optimize the upper bound of the original loss function in Appendix. Then, we discuss the theoretical analysis of the absolute value of the gradient used in GMT , demonstrating that this absolute value can effectively signify the importance of the parameter.

In order to consider the impact of task-specific data on the importance of each parameter in a model, we identify and update only a sparse subset of parameters based on their impact on the loss function. We define the loss function $L(\Theta; \mathcal{D})$ over the dataset $\mathcal{D}$ as:

$$\mathcal{L}(\Theta; \mathcal{D}) = \frac{1}{n} \sum_{i=1}^{n} \ell(\Theta; (\mathbf{x}_i, y_i)) \tag{4}$$

where ℓ represents the loss for a single data point, with $\mathbf{x}_i$ and y_i denoting the input features and corresponding label, respectively. The full parameter set is denoted by Θ.

To discern the impact of individual parameters θ_{ij} on the loss function $\mathcal{L}(\Theta; \mathcal{D})$, we consider the impact of their removal while keeping all other parameters constant. The differential effect of excluding a parameter θ_{ij} is quantified by the change in loss $\Delta\mathcal{L}_{ij}(\Theta; \mathcal{D})$, which is formulated as:

$$\Delta\mathcal{L}_{ij}(\Theta; \mathcal{D}) = \mathcal{L}(\mathbf{I} \odot \Theta; \mathcal{D}) - \mathcal{L}((\mathbf{I} - \mathcal{E}_{ij}) \odot \Theta; \mathcal{D}) \tag{5}$$

where $\mathbf{I}$ represents the identity matrix and $\mathcal{E}_{ij}$ denotes an indicator matrix that has the same dimensions as Θ. In $\mathcal{E}_{ij}$, all elements are zero except for the (i, j) element, which is one. The Hadamard product, indicated by $\odot$, performs an element-wise multiplication, isolating the effect of the single parameter θ_{ij}.

Given the computational infeasibility of evaluating $\Delta\mathcal{L}_{ij}$ for each parameter, we invoke the first-order Taylor series expansion around the current parameter vector Θ, which provides a linear approximation of the loss function's behavior in the vicinity of Θ. The first-order approximation is represented by the gradient of the loss function with respect to θ_{ij}:

$$\Delta\mathcal{L}_{ij}(\Theta; \mathcal{D}) \approx \nabla_{\theta_{ij}} \mathcal{L}(\Theta; \mathcal{D}) \cdot (-\theta_{ij}) \tag{6}$$

where $\nabla_{\theta_{ij}} \mathcal{L}(\Theta; \mathcal{D})$ denotes the partial derivative of the loss function concerning the parameter θ_{ij}. The negative sign arises from the fact that we are considering the removal of the parameter, which corresponds to a negative perturbation in its value.

Our objective is to discern the important parameters within the network architecture that are relevant to task-specific data. To this end, we employ the magnitude of the gradient $\nabla_{\theta_{ij}}$ as the criterion for saliency. It is imperative to recognize that a high magnitude of the gradient (regardless of its sign) typically denotes that the parameter θ_{ij} exerts a substantial influence on the loss function, whether the effect is positive or negative. Consequently, such parameters must be preserved to facilitate learning on the corresponding weights. The absolute value of the gradient $\nabla_{\theta_{ij}}$ is then used as a saliency measure to determine the importance of the parameter:

$$s_{ij} = \left| \nabla_{\theta_{ij}} \mathcal{L}(\Theta; \mathcal{D}) \right| \tag{7}$$

Using this saliency measure, we construct the binary mask matrix M such that $M_{ij} = 1$ if the parameter θ_{ij} is deemed important based on a predefined sparsity level κ:

$$M_{ij} = I[s_{ij} \geq \tilde{s}_{(\kappa)}] \tag{8}$$

where $\tilde{s}_{(\kappa)}$ is the saliency threshold selected to ensure that only the top κ influential parameters have their corresponding entries in M set to one. The indicator function $I[\cdot]$ yields one if the condition within the brackets is true and zero otherwise.

Experiments

Experimental Setup

To evaluate the effectiveness of the GMT approach, we conducted a comprehensive assessment of various models across a range of tasks, including code generation, mathematical reasoning, and general domains. Our experiments encompassed two training paradigms, SFT and DPO, allowing for a comparative analysis of the baseline performance in diverse settings. Furthermore, we performed further experimental analysis of mask ratio, drop strategy, and the overall efficiency of the approach.

Training and Evaluation. For code generation, we employ the Magicoder-Evol-Instruct-110K (Wei et al. 2023) as the training data, which is a decontaminated version of evol-codealpaca-v1[1]. We utilize MISTRAL-7B (Jiang et al. 2023) and DEEPSEEK-CODER-BASE-6.7B (Guo et al. 2024) as the base models. The trained models are evaluated using the HumanEval (Chen et al. 2021) and MBPP (Austin et al. 2021) benchmarks, which are widely recognized for their effectiveness in measuring the proficiency of Python text-to-code generation. To enable a more comprehensive evaluation, we also introduce HumanEval+ and MBPP+, both of which are provided by EvalPlus [2] (Liu et al. 2024a). For math reasoning, the MetaMathQA (Yu et al. 2023a) dataset is employed to fine-tune on the MISTRAL-7B and LLAMA3-8B models. The evaluation is conducted using the GSM8k (Cobbe et al. 2021) and MATH (Hendrycks et al. 2021) benchmarks, which are specifically constructed to test the model's capacity for mathematical reasoning and problem-solving. For the general domain, the TÜLU V2 (Wang et al. 2024a) dataset is utilized in SFT phase training on the LLAMA2-7B (Touvron et al. 2023) and LLAMA2-13B model, the UltraFeedback (Cui et al. 2023) is utilized in DPO phase training. Following HFT (Hui et al. 2024), we evaluate model on MMLU (Hendrycks et al. 2020), GSM8k (Cobbe et al. 2021), BBH (Suzgun et al. 2023), TyDiQA (Clark et al. 2020), TruthfulQA (Lin, Hilton, and Evans 2022) and HumanEval (Chen et al. 2021).

Implementation Details. We choose different base models for different tasks. All training experiments were done on NVIDIA A100 and NVIDIA H100 machines. In addition, we utilize BFloat16 precision and set the weight decay

[1] https://huggingface.co/datasets/theblackcat102/evol-codealpaca-v1

[2] https://evalplus.github.io/leaderboard.html

Method	HumanEval	HumanEval+	MBPP	MBPP+	Average	GSM8k	MATH	Average
	MISTRAL-7B					MISTRAL-7B		
Pre-trained†	28.7	23.8	51.9	42.1	36.6	-	-	-
SFT	68.3	64.0	56.1	46.9	58.8	75.3	27.0	51.2
Drop	68.3	64.0	54.6	46.1	58.3	74.6	27.6	51.1
HFT	67.1	61.6	57.1	48.1	58.5	77.8	27.3	52.6
RMT	**70.7**	**64.6**	55.1	45.4	59.0	74.6	25.1	49.9
GMT	69.5	62.2	**59.6**	**48.6**	**60.0**	**78.6**	**28.5**	**53.6**
	DEEPSEEK-CODER-BASE-6.7B					LLAMA3-8B		
Pre-trained†	47.6	39.6	72.0	58.7	54.5	-	-	-
SFT	76.8	73.8	74.9	62.4	72.0	78.1	29.2	53.7
Drop	76.2	72.6	74.7	62.4	71.5	78.1	29.4	53.8
HFT	74.4	70.1	75.2	62.9	70.7	74.2	28.5	51.4
RMT	72.6	67.1	**78.8**	**67.5**	71.5	80.3	31.3	55.8
GMT	**78.0**	**74.4**	75.7	63.7	**73.0**	**82.0**	**32.0**	**57.0**

Table 1: Experimental results for a single task in a specific domain. All models are evaluated with zero-shot prompting. † We obtain the results of the code benchmark from EvalPlus. "-" denotes that no zero-shot results were officially reported. Bold text indicates the best results for the fine-tuned model on each benchmark.

to 0. We use the cosine learning rate scheduler after a linear warm-up stage with a ratio of 0.03.

Baselines. In order to thoroughly evaluate the effectiveness of our method, we compare GMT to the following baselines:

- **SFT**: Vanilla supervised fine-tuning.
- **Drop**: As an extension of (Yu et al. 2023b), dropping a preset ratio of trivial delta parameters on a one-off basis after the vanilla supervised fine-tuning.
- **HFT**: Half Fine-Tuning, half of the parameters are selected for learning the new task, and the other half is frozen to retain previous knowledge.
- **RMT**: Random Mask-Tuning, a preset ratio of the parameters are randomly updated at a fine-grained element-wise level during the training process.

To ensure fairness, we apply an identical mask ratio across Drop, RMT, and GMT.

Main Results

Code Generation. The experimental results for the code generation task are presented in Table 1. By integrating task-specific gradient information with avoidance of random parameter selection, GMT achieves an average performance improvement of 1.2% on the MISTRAL-7B model and 1% on the DEEPSEEK-CODER-BASE-6.7B model. In contrast, HFT freezes half of the parameters to maintain the original capabilities of the model, and the learning process is hampered by the lack of sufficient parameter updates for a specific task domain. Furthermore, the RMT method shows some benefits in reducing redundant parameter updates on MISTRAL-7B. However, it fails to maintain this performance on another model, suggesting that it lacks robustness due to its inability to utilize task-specific data information. Compared to SFT, a one-off drop directly on a fine-tuned model does not improve performance due to its sensitivity to the optimal drop ratio. In the specialized field of code generation, this improvement underscores the efficacy of GMT in optimizing model performance through fine-grained parameter selection.

Math Reasoning. Table 1 presents a comparative analysis of the GMT method against various baseline approaches. GMT outperforms other methods by leveraging gradient information to capitalize on sparsity independent of the inconsistencies associated with random methods. Particularly, GMT performs excellently on the GSM8k benchmark, outperforming SFT by 3.3% and 3.9% on both models. In contrast, the HFT method shows an improvement over the baseline on MISTRAL-7B. However, it performs terribly on LLAMA3-8B, showing the instability of the method, which is attributed to the strategy of randomly chosen parameters. Drop is slightly improved on the MATH benchmark but not as effective as the GMT method that performs the drop operation throughout the training process. In summary, this strategic use of gradient signal associated with the task-specific data ensures stable and progressive performance improvements throughout the learning process, as evidenced by its success across different models.

General Domain. Table 2 shows the experimental comparison of the proposed GMT with the baseline. In the experiments with the fine-tuned LLAMA2-7B model, the GMT method outperforms the SFT by 3.0% on average across all tasks, and in particular, it significantly leads by 13.0% on GSM8k. GMT likewise demonstrated superior performance than HFT in multitasking scenarios in the general domain, obtaining an average lead of 1.1%. After the expansion of the LLM size, the GMT maintains its performance, with a 3.3% improvement in fine-tuning performance of LLAMA2-13B compared to the vanilla SFT. Notably, both RMT and

Model	Method	MMLU 0-shot, EM	GSM8k 8-shot CoT, EM	BBH 3-shot CoT, EM	TyDiQA 1-shot, F1	TruthfulQA 0-shot, MC2	HumanEval 0-shot, Pass@10	Average
LLAMA2-7B	Pre-trained	41.6	12.0	39.9	48.4	38.5	26.2	34.4
LLAMA2-13B	Pre-trained	52.2	34.5	50.7	50.3	49.8	32.7	45.0
Supervised Fine-tuning (SFT) on TÜLU V2								
LLAMA2-7B	SFT	48.5	25.0	42.2	51.2	41.7	**36.9**	41.0
	HFT	**50.8**	30.5	43.6	52.3	45.4	34.6	42.9
	RMT	47.4	34.5	**44.4**	52.9	**47.9**	33.7	43.4
	GMT	47.6	**38.0**	43.3	**53.1**	47.5	34.6	**44.0**
LLAMA2-13B	SFT	50.6	45.0	47.8	55.0	42.6	42.4	47.2
	HFT	54.5	46.5	**53.7**	56.7	45.7	**43.5**	50.1
	RMT	54.6	50.5	52.8	56.5	45.2	41.4	50.2
	GMT	**54.6**	**54.0**	51.5	**57.1**	**46.0**	39.5	**50.5**
Direct Preference Optimization (DPO) on UltraFeedback								
LLAMA2-7B	DPO	48.9	28.0	42.9	50.2	45.7	35.6	41.9
	HFT	**50.7**	30.5	42.8	43.9	49.8	35.1	42.1
	RMT	48.5	34.0	43.8	53.1	54.7	37.1	45.2
	GMT	48.5	**36.0**	**44.7**	**53.8**	**54.7**	**39.8**	**46.3**
LLAMA2-13B	DPO	52.0	44.0	47.1	51.5	45.5	**44.3**	47.4
	HFT	**55.0**	45.5	51.4	53.2	49.5	42.9	49.6
	RMT	54.4	55.5	50.9	56.3	51.7	40.5	51.6
	GMT	54.5	**56.0**	**51.9**	**56.7**	**53.3**	41.1	**52.3**

Table 2: Results of LLAMA2-7B and LLAMA2-13B fine-tuned on the TÜLU V2 dataset. The results of Pre-trained, SFT, and HFT are taken from (Hui et al. 2024). Bold text indicates the best results on each benchmark. The DPO phase is initialized with the corresponding methods using the HFT- and GMT-based SFT models fine-tuned on TÜLU V2, respectively.

HFT perform better than SFT, suggesting that the general domain of multi-tasking benefits from appropriate sparsity, even when the parameter selection strategy is completely random. With the same amount of parameter updates, GMT shows further improvement over RMT, suggesting that the strategy of selecting parameters based on gradient information derived from task-specific data in our method is practically effective. After extending the GMT to DPO, the superiority of the GMT approach remains evident. Experimental results across two model sizes indicate that GMT identifies parameters more worthy of updating in multi-task learning, thereby reducing the redundancy of parameter updates and enhancing model performance.

Further Discussions

Analysis of Mask Ratio. To comprehensively ascertain the sensitivity of the hyperparameter mask ratio selection on GMT, we analyze the influence of the amount of parameter updates during training. Comparative experiments were conducted across three domains. We iterated through all experiments, applying the range of parameters to be fine-tuned with 10% granularity. For extreme cases, experiments were performed with mask ratios of 95% and 99%. As shown in Figure 2, the experimental results indicate that despite fluctuations in the number of parameter updates, the proposed method maintains a relatively consistent performance level with no more than 90% mask ratio. Our GMT reveals that the optimal performance in fields like mathematics and coding, which require specialized knowledge, is achieved with mask ratios of 20% to 40%. Even under extreme mask ratio settings, GMT maintains impressive performance in task-specific domains such as math reasoning and code generation. However, its effectiveness significantly diminishes in the general domain within multi-task scenarios. This suggests a stronger adaptability of GMT to single-task scenarios, where the entire learning process can be completed with merely a 1% parameter update. Conversely, in multi-task settings, the potential lack of sufficient parameter updates may lead to an inadequate learning process. Further analysis demonstrates that the performance of the proposed GMT exhibits robustness to variations in the mask ratio, despite the latter being an adjustable hyperparameter within the model's configuration.

Analysis of Drop Strategy. Our analysis reveals that not all delta parameters contribute equally to the model's performance. Specifically, we find that parameters with salient values are critical for maintaining the efficacy of the LLM, and their removal leads to a notable and rapid degradation in model functionality. To systematically explore this phenomenon, we designed a series of experiments using the MISTRAL-7B model, focusing on domain-specific applications in mathematics (GSM8k and MATH). The experimental results are depicted in Appendix.

Our results indicate substantial differences in the impact of each strategy on model performance. The strategy of dropping a portion of the delta parameters with small absolute values after vanilla fine-tuning demonstrates a relatively robust performance, only showing significant degradation

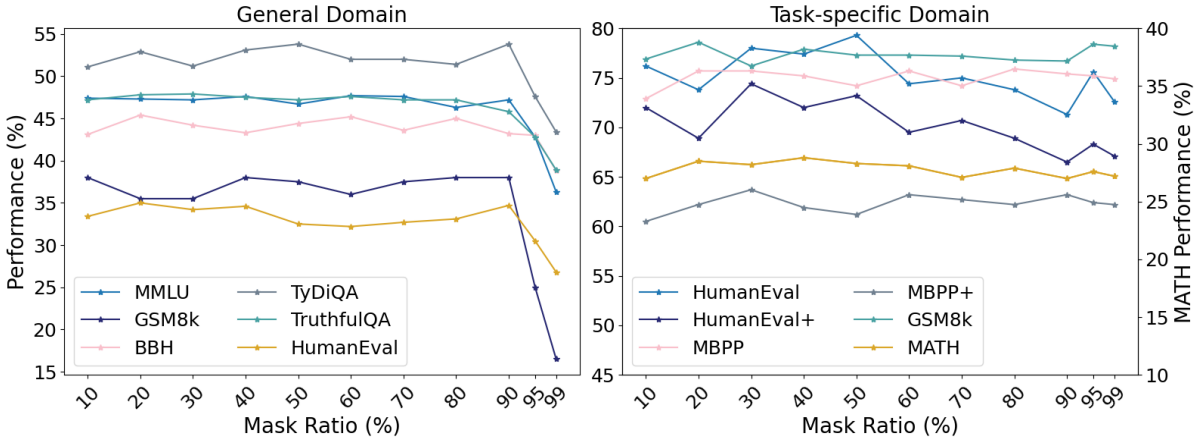

Figure 2: Fine-tuning performance of the proposed GMT with respect to various mask ratios during training in three domains. The training LLMs utilized for code generation task, math reasoning task, and general domain are DEEPSEEK-CODER-BASE-6.7B, MISTRAL-7B, and LLAMA2-7B, respectively. The experimental results for the MATH benchmark are presented on the secondary y-axis located on the right side of the figure.

Model	**Method**	**Traning Speed** (samples/s)	**FLOPs** ($1e^{18}$)
MISTRAL-7B Code Generation	Vanilla SFT	15.32	1.481
	RMT	13.37	1.481
	GMT	13.43	1.480
LLAMA2-13B General Domain	Vanilla SFT	13.53	1.706
	RMT	12.54	1.707
	GMT	12.74	1.707

Table 3: Comparison of training speed and computational FLOPs, the number of train samples per second is an indicator of training speed.

when the dropout rate reaches 80%. In contrast, the random drop strategy leads to an apparent performance decline at a much lower sparsity level of 40%. The strategy of dropping parameters with salient magnitudes results in a precipitous decline in model capabilities. This finding underscores the importance of these significant delta parameters in sustaining the functional integrity of the model. Expectedly, the utilization of the gradient information to trivial drop during the training process can elevate the upper limit of LLM performance and is not constrained by drop rate.

Such experiments demonstrate that the parameter updates during the fine-tuning process of LLM are redundant, and that the use of a reasonable sparse updating strategy enables the model to learn better. Comparison with the experiments of random drop indicates that our strategy of considering task-specific data during training, using gradient information as a signal, and retaining neurons with large update magnitudes for completing the update is effective.

Analysis of Efficiency. To demonstrate the time-efficiency of proposed GMT, we compare the training speed and computed FLOPs of GMT, vanilla SFT, and RMT, with the metric of number of training samples per second responding to the training speed, and the results are shown in Table 3. The FLOPs computed by the three tuning strategies during training are quite comparable due to the fact that the gradient information utilized by GMT is a by-product of model training and does not impose additional derivation operations. In terms of training speed, the RMT is close to GMT, being 14% and 6% slower than vanilla SFT in the code generation and general domain, respectively. Since GMT needs to calculate the threshold of the gradient being masked based on a preset ratio, a process that requires a `TopK` operation after the model backpropagates the gradient, and then drops values with smaller absolute values of the gradient based on the threshold. Nonetheless, this trade-off between a small minor time overhead and performance improvement is completely acceptable.

Conclusion

In this paper, we propose the Gradient-Mask Tuning (GMT), a pragmatic approach to optimize LLMs by selectively updating parameters based on gradient information. GMT identifies more important parameters to eliminate redundant updates introduced in fine-tuning and improves model performance on various tasks. We evaluate the performance of GMT on multiple models in the code generation, math reasoning, and general domain. Experiments demonstrate that GMT not only elevates the upper limits of LLM performance across diverse tasks but also exhibits robustness to various fine-tuning settings. We further analyze the computational efficiency of GMT to confirm that no additional computational FLOPs are needed, as well as acceptable extra time consumption. Moreover, GMT allows for fine-tuning without destroying the network structure, and thus can readily replace SFT and DPO to accomplish the optimization process.

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Austin, J.; Odena, A.; Nye, M.; Bosma, M.; Michalewski, H.; Dohan, D.; Jiang, E.; Cai, C.; Terry, M.; Le, Q.; et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; Pinto, H. P. d. O.; Kaplan, J.; Edwards, H.; Burda, Y.; Joseph, N.; Brockman, G.; et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Chen, Y.; Wang, S.; Lin, Z.; Qin, Z.; Zhang, Y.; Ding, T.; and Sun, R. 2024. MoFO: Momentum-Filtered Optimizer for Mitigating Forgetting in LLM Fine-Tuning. *arXiv preprint arXiv:2407.20999*.

Clark, J. H.; Choi, E.; Collins, M.; Garrette, D.; Kwiatkowski, T.; Nikolaev, V.; and Palomaki, J. 2020. Tydi qa: A benchmark for information-seeking question answering in ty pologically di verse languages. *Transactions of the Association for Computational Linguistics*, 8: 454–470.

Cobbe, K.; Kosaraju, V.; Bavarian, M.; Chen, M.; Jun, H.; Kaiser, L.; Plappert, M.; Tworek, J.; Hilton, J.; Nakano, R.; et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Cui, G.; Yuan, L.; Ding, N.; Yao, G.; Zhu, W.; Ni, Y.; Xie, G.; Liu, Z.; and Sun, M. 2023. Ultrafeedback: Boosting language models with high-quality feedback. *arXiv preprint arXiv:2310.01377*.

Dalvi, F.; Sajjad, H.; Durrani, N.; and Belinkov, Y. 2020. Analyzing redundancy in pretrained transformer models. *arXiv preprint arXiv:2004.04010*.

Fu, Z.; Yang, H.; So, A. M.-C.; Lam, W.; Bing, L.; and Collier, N. 2023. On the effectiveness of parameter-efficient fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, 12799–12807.

Guo, D.; Zhu, Q.; Yang, D.; Xie, Z.; Dong, K.; Zhang, W.; Chen, G.; Bi, X.; Wu, Y.; Li, Y.; et al. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming–The Rise of Code Intelligence. *arXiv preprint arXiv:2401.14196*.

Hendrycks, D.; Burns, C.; Basart, S.; Zou, A.; Mazeika, M.; Song, D.; and Steinhardt, J. 2020. Measuring Massive Multitask Language Understanding. In *International Conference on Learning Representations*.

Hendrycks, D.; Burns, C.; Kadavath, S.; Arora, A.; Basart, S.; Tang, E.; Song, D.; and Steinhardt, J. 2021. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*.

Hoefler, T.; Alistarh, D.; Ben-Nun, T.; Dryden, N.; and Peste, A. 2021. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks. *Journal of Machine Learning Research*, 22(241): 1–124.

Huang, W.; Qin, H.; Liu, Y.; Li, Y.; Liu, X.; Benini, L.; Magno, M.; and Qi, X. 2024. SliM-LLM: Salience-Driven Mixed-Precision Quantization for Large Language Models. *arXiv preprint arXiv:2405.14917*.

Hui, T.; Zhang, Z.; Wang, S.; Xu, W.; Sun, Y.; and Wu, H. 2024. HFT: Half Fine-Tuning for Large Language Models. *arXiv preprint arXiv:2404.18466*.

Jiang, A. Q.; Sablayrolles, A.; Mensch, A.; Bamford, C.; Chaplot, D. S.; Casas, D. d. l.; Bressand, F.; Lengyel, G.; Lample, G.; Saulnier, L.; et al. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Jiang, S.; Liao, Y.; Zhang, Y.; Wang, Y.; and Wang, Y. 2024. TAIA: Large Language Models are Out-of-Distribution Data Learners. *arXiv preprint arXiv:2405.20192*.

Lee, C.; Cho, K.; and Kang, W. 2019. Mixout: Effective regularization to finetune large-scale pretrained language models. *arXiv preprint arXiv:1909.11299*.

Lin, S.; Hilton, J.; and Evans, O. 2022. TruthfulQA: Measuring How Models Mimic Human Falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 3214–3252.

Liu, J.; Xia, C. S.; Wang, Y.; and Zhang, L. 2024a. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in Neural Information Processing Systems*, 36.

Liu, Y.; He, H.; Han, T.; Zhang, X.; Liu, M.; Tian, J.; Zhang, Y.; Wang, J.; Gao, X.; Zhong, T.; et al. 2024b. Understanding llms: A comprehensive overview from training to inference. *arXiv preprint arXiv:2401.02038*.

Luo, R.; Gu, T.; Li, H.; Li, J.; Lin, Z.; Li, J.; and Yang, Y. 2024a. Chain of history: Learning and forecasting with llms for temporal knowledge graph completion. *arXiv preprint arXiv:2401.06072*.

Luo, Z.; Zhang, X.; Liu, X.; Li, H.; Gong, Y.; Qi, C.; and Cheng, P. 2024b. Velocitune: A Velocity-based Dynamic Domain Reweighting Method for Continual Pre-training. *arXiv preprint arXiv:2411.14318*.

Lv, Z.; Zhang, W.; Chen, Z.; Zhang, S.; and Kuang, K. 2024. Intelligent model update strategy for sequential recommendation. In *Proceedings of the ACM on Web Conference 2024*, 3117–3128.

Ma, X.; Fang, G.; and Wang, X. 2023. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36: 21702–21720.

Men, X.; Xu, M.; Zhang, Q.; Wang, B.; Lin, H.; Lu, Y.; Han, X.; and Chen, W. 2024. Shortgpt: Layers in large language models are more redundant than you expect. *arXiv preprint arXiv:2403.03853*.

Pentyala, S. K.; Wang, Z.; Bi, B.; Ramnath, K.; Mao, X.-B.; Radhakrishnan, R.; Asur, S.; et al. 2024. PAFT: A Parallel Training Paradigm for Effective LLM Fine-Tuning. *arXiv preprint arXiv:2406.17923*.

Song, J.; Oh, K.; Kim, T.; Kim, H.; Kim, Y.; and Kim, J.-J. 2024. SLEB: Streamlining LLMs through Redundancy Verification and Elimination of Transformer Blocks. *arXiv preprint arXiv:2402.09025*.

Suzgun, M.; Scales, N.; Schärli, N.; Gehrmann, S.; Tay, Y.; Chung, H. W.; Chowdhery, A.; Le, Q.; Chi, E.; Zhou, D.; et al. 2023. Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them. In *Findings of the Association for Computational Linguistics: ACL 2023*, 13003–13051.

Touvron, H.; Martin, L.; Stone, K.; Albert, P.; Almahairi, A.; Babaei, Y.; Bashlykov, N.; Batra, S.; Bhargava, P.; Bhosale, S.; et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Wang, Y.; Ivison, H.; Dasigi, P.; Hessel, J.; Khot, T.; Chandu, K.; Wadden, D.; MacMillan, K.; Smith, N. A.; Beltagy, I.; et al. 2024a. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36.

Wang, Y.; Liu, Y.; Shi, C.; Li, H.; Chen, C.; Lu, H.; and Yang, Y. 2024b. Inscl: A data-efficient continual learning paradigm for fine-tuning large language models with instructions. *arXiv preprint arXiv:2403.11435*.

Wei, Y.; Wang, Z.; Liu, J.; Ding, Y.; and Zhang, L. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Woo, S.; Park, B.; Kim, B.; Jo, M.; Kwon, S.; Jeon, D.; and Lee, D. 2024. DropBP: Accelerating Fine-Tuning of Large Language Models by Dropping Backward Propagation. *arXiv preprint arXiv:2402.17812*.

Wortsman, M.; Ilharco, G.; Gadre, S. Y.; Roelofs, R.; Gontijo-Lopes, R.; Morcos, A. S.; Namkoong, H.; Farhadi, A.; Carmon, Y.; Kornblith, S.; et al. 2022. Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time. In *International conference on machine learning*, 23965–23998. PMLR.

Yadav, P.; Tam, D.; Choshen, L.; Raffel, C. A.; and Bansal, M. 2024. Ties-merging: Resolving interference when merging models. *Advances in Neural Information Processing Systems*, 36.

Yu, L.; Jiang, W.; Shi, H.; Jincheng, Y.; Liu, Z.; Zhang, Y.; Kwok, J.; Li, Z.; Weller, A.; and Liu, W. 2023a. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. In *The Twelfth International Conference on Learning Representations*.

Yu, L.; Yu, B.; Yu, H.; Huang, F.; and Li, Y. 2023b. Language models are super mario: Absorbing abilities from homologous models as a free lunch. *arXiv preprint arXiv:2311.03099*.

Zheng, C.; Wang, Z.; Ji, H.; Huang, M.; and Peng, N. 2024. Weak-to-strong extrapolation expedites alignment. *arXiv preprint arXiv:2404.16792*.