LoRA-Flow: Dynamic LoRA Fusion for Large Language Models in Generative Tasks

Hanqing Wang[*1] **Bowen Ping**[*2] **Shuo Wang**[†3] **Xu Han**[3,4,5]
Yun Chen[†1] **Zhiyuan Liu**[3,4,5] **Maosong Sun**[3,4,5]
[1]Shanghai University of Finance and Economics [2]Peking University
[3]Dept. of Comp. Sci. & Tech., Tsinghua University, Beijing, China
[4]Institute for AI, Tsinghua University, Beijing, China
[5]Beijing National Research Center for Information Science and Technology

Abstract

LoRA employs lightweight modules to customize large language models (LLMs) for each downstream task or domain, where different learned additional modules represent diverse skills. Combining existing LoRA modules to address new tasks can enhance the reusability of learned LoRA modules, particularly beneficial for tasks with limited annotated data. Most prior works on LoRA combination primarily rely on task-level weights for each involved LoRA, making different examples and tokens share the same LoRA weights. However, in generative tasks, different tokens may necessitate diverse skills to manage. Taking the Chinese math task as an example, understanding the problem description may depend more on the Chinese LoRA, while the calculation part may rely more on the math LoRA. To this end, we propose LoRA-Flow, which utilizes dynamic weights to adjust the impact of different LoRA modules. The weights at each step are determined by a fusion gate with extremely few parameters, which can be learned with only 200 training examples. Experiments across six generative tasks demonstrate that our method consistently outperforms baselines with task-level fusion weights. This underscores the necessity of introducing dynamic fusion weights for LoRA combination.[1]

1 Introduction

Large language models (LLMs) have demonstrated superior performance over previous smaller models across a wide range of tasks (OpenAI, 2023; Anil et al., 2023; Touvron et al., 2023a,b), thereby extending the applicability of AI systems to numerous real-world scenarios. Because of their substantial model size, training all parameters of LLMs can often be prohibitively expensive. Therefore, several researchers have proposed a set of parameter-efficient fine-tuning (PEFT) approaches (Houlsby et al., 2019; Li and Liang, 2021). Among these, LoRA (Hu et al., 2022) stands out as one of the most popular due to its efficiency and simplicity.

The basic idea of LoRA is to learn an additional module for each downstream domain or task. Rather than solely employing a single LoRA to address learned tasks, several recent studies have delved into the potential of combining existing LoRA modules to tackle unseen tasks (Zhang et al., 2023; Huang et al., 2024; Chronopoulou et al., 2023). This direction holds the potential to substantially enhance the reusability of learned parameters, facilitating the integration of diverse model capabilities.

Most existing LoRA fusion approaches employ a task-level weight distribution when combining different LoRA modules. This implies that all test examples and tokens share the same fusion ratio.

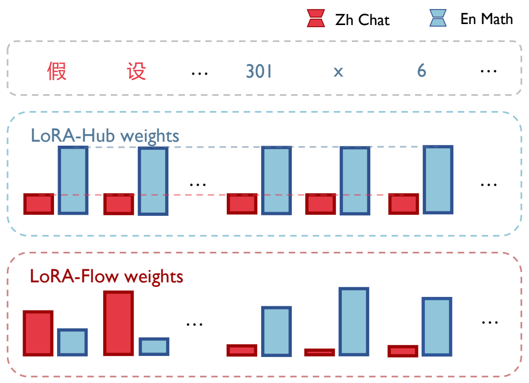


Figure 1: Illustration of the proposed LoRA-Flow method. For the token y_t at the t-th step, we use a gate that conditions on the prefix $\mathbf{y}_{<t}$ to determine the fusion weights. The dynamic fusion weights are intended to control the influence of different LoRA modules, to better cope with various types of tokens in generative tasks. Red and blue rectangles represent the weights assigned to the two involved LoRA modules.

[*] Equal Contribution.
[†] Corresponding authors.
[1] Code and models will be publicly available at `https://github.com/thunlp/LoRAFlow`.

However, for some complex generative tasks (e.g., solving mathematical problems or generating code according to provided instructions), the LLM may need to dynamically employ various types of capabilities to address the entire problem effectively. Figure 1 illustrates an example, where we have trained a Chinese chat LoRA (i.e., Zh Chat) and an English math LoRA (i.e., En Math), and our objective is to address a Chinese math problem. Intuitively, comprehending the Chinese problem description may rely more on the Chinese chat LoRA, whereas performing the calculation might depend more on the English math LoRA.

In this work, we propose LoRA-Flow, which can dynamically determine the token-level weights of different LoRA modules for generative tasks. At each time step, the fusion weights are generated by a gate module that conditions the current prefix. The fusion gate comprises an extremely small number of parameters, accounting for only approximately 0.2% of those in a LoRA. We find through experiments that the fusion gate can be learned through only 200 training examples. Figure 1 gives an example of LoRA-Flow. We also observe significant variations in weights across different model layers, suggesting that the impact of LoRA modules differs across layers. In summary, the contributions of this work can be outlined as follows:

- We propose LoRA-Flow, a method that combines existing LoRA modules with dynamic fusion weights to effectively control the influence of each LoRA module across various generation steps.
- We verify the effectiveness of LoRA-Flow on six different generation tasks, and the results show that LoRA-Flow can consistently outperform the baselines that use task-level fusion weights (e.g., LoRA-Hub).
- By carefully designed analyses from various aspects, we provide deeper insights into the integration of LoRA modules. We consider this journey to be fruitful in constructing a flexible plug-and-play community for LLMs, enabling developers to leverage plugins created by others to build up their own LLM applications.

2 Background

Large Language Models Most recent LLMs utilize a decoder-only architecture, comprising stacked layers of identical structure to form the large-scale model. For a sequence $\mathbf{y}$, an LLM estimates its probability in the following way:

$$P(\mathbf{y}|\boldsymbol{\theta}_{\text{base}}) = \prod_{t=1}^{T} P(y_t|\mathbf{y}_{<t};\boldsymbol{\theta}_{\text{base}}), \quad (1)$$

where y_t denotes the token at the t-th step and $\mathbf{y}_{<t}$ is the prefix before t. $\boldsymbol{\theta}_{\text{base}}$ represents the parameters of the basic LLM.

LoRA LoRA (Hu et al., 2022) is a parameter-efficient fine-tuning method that can achieve comparable performance to full fine-tuning in certain scenarios but at a significantly lower cost. Specifically, for a given matrix $W \in \mathbb{R}^{m\times d}$ within the model, we can learn two low-rank matrices $A \in \mathbb{R}^{r\times d}$ and $B \in \mathbb{R}^{m\times r}$ to approximate the parameter update for W:

$$\Delta W = BA. \quad (2)$$

LoRA Fusion Each trained LoRA possesses unique capabilities, and their combination can integrate various skills of LLMs. Formally, we use $\Delta W_1 = B_1A_1$ and $\Delta W_2 = B_2A_2$ to represent two existing LoRA modules. Zhang et al. (2023) propose to merge two LoRA modules in the following way:

$$\mathbf{h}' = W\mathbf{x} + \lambda\Delta W_1\mathbf{x} + (1-\lambda)\Delta W_2\mathbf{x}, \quad (3)$$

where λ is a hyper-parameter that needs to be manually tuned.

LoRA-Hub (Huang et al., 2024) further improve the fusion method in the following ways:

$$\begin{aligned}\mathbf{h}' &= W\mathbf{x} \\ &\quad + (w_1B_1 + w_2B_2)(w_1A_1 + w_2A_2)\mathbf{x},\end{aligned} \quad (4)$$

where the fusion weights w_1 and w_2 are learned in a few-shot manner. While LoRA-Hub can automatically determine fusion weights for different LoRA modules, the weights across different tokens remain the same for a given task. This shared weight scheme may constrain the expressive capacity of the involved LoRA modules, particularly in complex generative tasks that entail diverse types of context.

3 Approach

To handle generative tasks more flexibly, we propose LoRA-Flow, which employs dynamic fusion

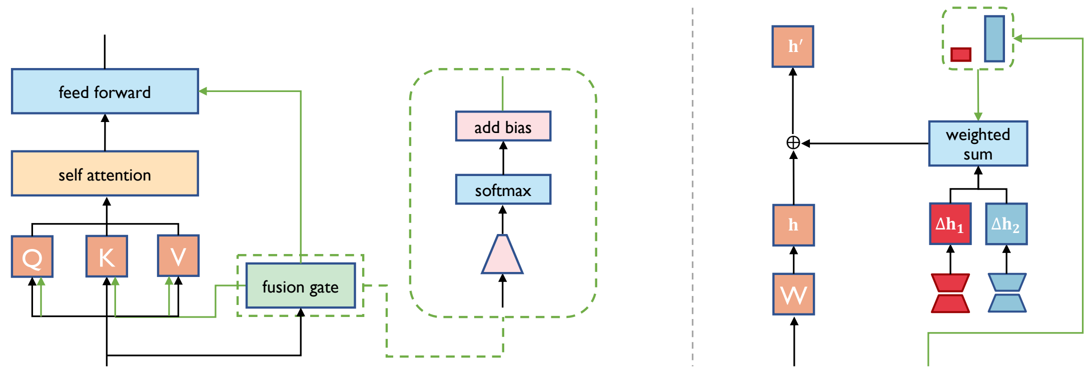


Figure 2: Left: we use layer-wise fusion gates to facilitate dynamic LoRA fusion, which project input hidden states of each layer into fusion weights. Right: for a certain module, the provided fusion weights are used to aggregate the outputs of different LoRA modules. Since our goal is to leverage the abilities acquired by existing LoRA modules to address new tasks, we only train the fusion gate with a few examples, while keeping both the model and the LoRA modules frozen. The number of parameters of the fusion gate is only approximately 0.2% of those in a LoRA.

weights at each generation step. Subsequently, we will first introduce the way we calculate the fusion weights in Section 3.1, and then detail how we integrate the weights into the model in Section 3.2. Finally, we describe the training algorithm in Section 3.3.

3.1 Calculating Fusion Weights

At the t-th step, we aim to determine the fusion weights using the prefix $\mathbf{y}_{<t}$, which captures the context of the current token. Given that the backbone model has already compressed the context information into hidden vectors, we propose directly utilizing the hidden state at the t-th step to avoid redundant computations.

There are three levels of hidden states, each containing different granularities of information:

- Step-level hidden states $\mathbf{x}_t$: the input word embedding at the t-th step.
- Layer-level hidden states $\mathbf{x}_t^l$: the input representation to the l-th layer at the t-th step.
- Module-level hidden states $\mathbf{x}_t^{l,\text{type}}$: the input vector to a specific module (e.g., the query projection in the self-attention network).

As found in some previous studies, the hidden states in various layers may lie in different manifolds (Voita et al., 2019). Therefore, simply using the step-level hidden state $\mathbf{x}_t$ to compute the fusion weights for the entire model may be not sufficiently effective. On the other side, using module-level hidden states would introduce plenty of new parameters for the fusion gates, as each module requires an independent gate to cope with its input states $\mathbf{x}_t^{l,\text{type}}$. We thus use the layer-level hidden state $\mathbf{x}_t^l$. Empirical studies in Section 6.1 indicate that layer-wise fusion weights can outperform the other two types of counterparts.

As shown in Figure 2, for the l-th layer, the fusion gate takes in the input representation $\mathbf{x}_t^l$ and then projects it into fusion weights:

$$\mathbf{w}^l = \text{softmax}\left(W_{\text{gate}}^l \mathbf{x}_t^l\right) + \mathbf{b}^l, \tag{5}$$

where $W_{\text{gate}}^l \in \mathbb{R}^{k\times d}$ and $\mathbf{b}^l \in \mathbb{R}^{k\times 1}$. k is the number of LoRA modules. Both W_{gate}^l and $\mathbf{b}^l$ are learnable parameters. Given that k is typically substantially smaller than d and r, the number of parameters within the fusion gates is negligible compared to that of the LoRA. For instance, in Llama-2-7b (Touvron et al., 2023b), a LoRA for the entire model contains 117.44M parameters with $r = 64$, while the gates for combining two LoRA modules only consist of 0.26M parameters in total.

3.2 Integrating Fusion Weights

As mentioned in Section 3.1, we use layer-wise fusion weights in LoRA-Flow. Once we get the fusion weights at the l-th layer, we feed the weights $\mathbf{w}^l$ to all the modules that contain LoRA modules. Different from LoRA-Hub (Huang et al., 2024) that separately composes LoRA A matrices and LoRA B matrices (as shown in Eq (4)), we integrate the outputs of different LoRA modules, treating each LoRA as a complete module. The reason for this operation is to combine LoRA modules with different middle ranks. Let $\Delta\mathbf{h} = [\Delta\mathbf{h}_1; \cdots; \Delta\mathbf{h}_k] \in \mathbb{R}^{d\times k}$ denote the outputs of all the involved LoRA modules, the fusion process can be expressed by

$$\mathbf{h}' = \mathbf{h} + \Delta\mathbf{h}\mathbf{w}^l, \tag{6}$$

where $\mathbf{h}$ is the module's output in the backbone. Figure 2 shows an example.

3.3 Training

For a backbone model $\boldsymbol{\theta}_{\text{base}}$ and a set of learned LoRA modules $\boldsymbol{\theta}_{\text{LoRA}} = \{\boldsymbol{\theta}^1_{\text{LoRA}}, \cdots, \boldsymbol{\theta}^k_{\text{LoRA}}\}$, we train the fusion gate $\boldsymbol{\theta}_{\text{fusion}}$ on the new task:

$$\hat{\boldsymbol{\theta}}_{\text{fusion}} = \underset{\boldsymbol{\theta}_{\text{fusion}}}{\text{argmax}} \left\{ \mathcal{L}(\boldsymbol{\theta}_{\text{total}} | \mathcal{D}_{\text{new}}) \right\}, \tag{7}$$

where $\boldsymbol{\theta}_{\text{total}} = \boldsymbol{\theta}_{\text{base}} \cup \boldsymbol{\theta}_{\text{LoRA}} \cup \boldsymbol{\theta}_{\text{fusion}}$ denotes the total parameters. The likelihood is defined as

$$\mathcal{L}(\boldsymbol{\theta}_{\text{total}} | \mathcal{D}_{\text{new}}) = \sum_{i=1}^{N} P(\mathbf{y}_i | \boldsymbol{\theta}_{\text{total}}), \tag{8}$$

where N is the number of training examples on the new task. We follow LoRA-Hub (Huang et al., 2024) to learn the fusion modules in a few-shot manner, where N is set to 200 in our experiments. We investigate the effect of N in Section 6.4. Since $\boldsymbol{\theta}_{\text{fusion}}$ consists of only a few parameters, these limited training examples are adequate for learning an effective fusion mechanism.

4 Experiment

4.1 Setup

Base Model We use Llama-2 (Touvron et al., 2023b) as our base LLM to examine the performance of various LoRA fusion approaches, as it is among the most widely used open-source LLMs. Due to computational constraints, we use Llama-2-7b by default.

LoRA Training To conduct LoRA fusion experiments, we first learn several LoRA modules on the tasks with sufficient supervised data:

- Chinese chat (Zh Chat): we use the data released by Lai et al. (2023) to learn the Chinese chat LoRA, which is expected to possess the ability to understand and generate Chinese text. There are 52K training examples in total.

- Russian chat (Ru Chat): the training data of Russian chat LoRA is also from Lai et al. (2023), which consists of 52K training examples in Russian.

- Spanish chat (Es Chat): the training data is also from Lai et al. (2023), containing 52K training examples in Spanish.

- English math (En Math): the training data for English math LoRA is constructed by Yu et al. (2023), which is comprised of 395K mathematical problems in English.

- English code (En Code): we train the English code LoRA with the Magicoder dataset (Wei et al., 2023), which consists of 186K code generation problems in English.

We integrate LoRA modules into the query, key, value, and output projections within attention networks and the three linear projections in feedforward networks. By default, the LoRA rank r is set to 64 and the value of α is set to 16. For the En code LoRA, we set r=256, as we have observed that the code LoRA with r=64 is inadequate for learning the task effectively. Each LoRA module is trained using 8 A100 80G GPUs, where each mini-batch contains 128 training examples. We use the cosine warmup schedule and the peak learning rate is 1e-4. For each task, the LoRA is trained by 3 epochs and the warmup ratio is set to 0.04.

LoRA Fusion We evaluate LoRA fusion methods in a few-shot manner. Specifically, we conduct the evaluation on six tasks, including Zh math, Ru math, Es math, Zh code, Ru code, and Es code. For each task, we combine the chat LoRA in the target language and the task LoRA in English. Briefly, we use *language LoRA* to represent the target-language chat LoRA and *task LoRA* to denote the math or code LoRA trained in English.

For each task, we construct 200 training examples for the few-shot training, which is firstly translated by GPT-3.5 based on the English math or code data and then verified by humans. For fusion experiments on math, we also construct 100 problems as the validation set. For code experiments, we employ humans to create 20 problems as the validation set, since the code generation test examples are more time-consuming to annotate.[2] Both LoRA-Hub and the proposed LoRA-Flow are trained with the few-shot data. We use the validation to search the training hyperparameters. The search space of the peak learning rate is {1e-3, 1e-4}, and the search space of the batch size is {2, 4,

[2]See Appendix B for the translation prompt and Appendix C for the details of human annotation.

8}.[3] Each fusion module is trained with 5 epochs, using the few-shot training data.

Evaluation For fusion experiments on math, we use MGSM (Shi et al., 2023) as the test set, which is a widely used multilingual evaluation benchmark for the math abilities of LLMs. For fusion experiments on code generation, we construct a multilingual version of HumanEval (Chen et al., 2021). The original English problem descriptions in HumanEval are translated by GPT-3.5 into other languages and then verified by humans. We report the accuracy and the pass@1 score on MGSM and HumanEval, respectively.

4.2 Main Results

Table 1 shows the results of the involved methods on different generative tasks. For comparison, we also show the performances of the base model and single LoRA. The task LoRA modules trained in English already demonstrate a notable degree of cross-lingual transfer capabilities, outperforming the language LoRA modules on non-English tasks. For example, in the Zh math task, the task LoRA achieves an accuracy of 26.8, whereas the language LoRA achieves only 5.2. On the code generation tasks, the task LoRA also significantly outperforms the language LoRA modules.

When combining the language and task LoRA modules, we compare our method with two baselines that use task-level fusion weights. The "Average" baseline refers to simply averaging the outputs of the two involved LoRA modules. As introduced in Section 2, LoRA-Hub learns task-level fusion weights using few-shot training data, similar to the proposed LoRA-Flow. However, LoRA-Flow utilizes dynamic fusion weights, which vary across different time steps and model layers. We use the open-source code released by Huang et al. (2024) to reimplement LoRA-Hub in our experiments.

From the experiments, we find that "Average" and LoRA-Hub perform even worse than the single task LoRA, demonstrating that combining different LoRA modules with static weights is not effective enough for complex generative tasks like solving mathematical problems or generating code segments. Specifically, LoRA-Hub only outperforms the single-task LoRA on the Zh code task.

Trained with the same few-shot training data as LoRA-Hub, LoRA-Flow outperforms the baselines across all six examined tasks. For instance, LoRA-Flow achieves a performance improvement of 2.3 compared to LoRA-Hub (22.6 vs. 20.3) on code generation tasks. This confirms the necessity of employing dynamic weights for generative tasks.

4.3 Results on Larger Model

To confirm the effectiveness of LoRA-Flow with larger LLMs, we also conduct experiments on Llama-2-13b. Given that larger models require higher training costs, we only train three LoRA modules for the 13b model, namely Zh Chat, En Math, and En Code. The results of different fusion approaches on the 13b model are shown in Table 2. On the larger model, all the methods involved achieve better results than those on the 7b model. Compared to LoRA-Hub, LoRA-Flow exhibits superior performances on both the math and the code tasks. These results also demonstrate that LoRA-Flow is compatible with models of various sizes.

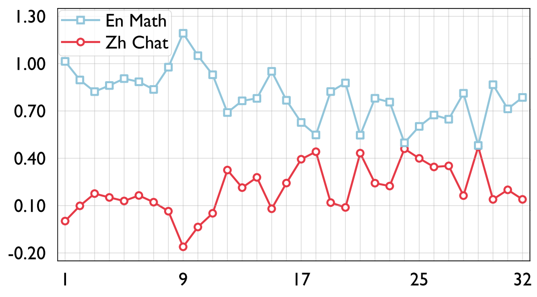


Figure 3: Average fusion weights for the Zh Chat and En Math LoRA modules across different layers.

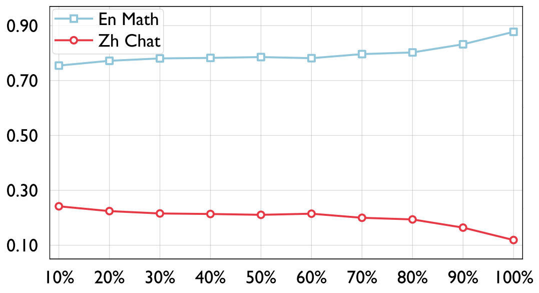


Figure 4: Average fusion weights for the Zh Chat and En Math LoRA modules at different time steps.

5 Analysis

To gain deeper insights into the behavior of LoRA-Flow, we conduct a comprehensive series of analyses from various perspectives. We first estimate the average fusions at different model layers in Section 5.1, and then investigate how the fusion

[3]Refer to Table 5 in Appendix for the best hyperparameters of each generation task.

Method		MGSM (Math)				HumanEval (Code)			
		Zh	Ru	Es	Avg.	Zh	Ru	Es	Avg.
BASE MODEL		4.4	3.2	2.4	3.3	0.0	0.0	2.4	0.8
SINGLE LORA	LANG	5.2	3.6	3.6	4.1	12.2	14.0	10.4	12.2
	TASK	26.8	32.8	41.2	33.6	18.3	23.2	21.9	21.1
LORA FUSION	AVERAGE	12.8	10.4	18.4	13.9	17.1	17.7	18.3	17.7
	LORA-HUB	20.8	28.4	36.8	28.7	19.5	21.3	20.1	20.3
	LORA-FLOW	**33.2**	**37.6**	**42.0**	**37.6**	**20.7**	**23.8**	**23.2**	**22.6**

Table 1: Evaluation results on MGSM and HumanEval. "LANG" denotes the chat LoRA in the target language and "TASK" represents the math or code LoRA trained in English. LoRA fusion methods combine the language LoRA and the task LoRA to accomplish the new task. The best score is highlighted in **bold**.

Method	MGSM	HumanEval
BASE MODEL	6.8	0.0
LANG LORA	7.6	18.9
TASK LORA	36.8	34.7
AVERAGE	16.4	30.4
LORA-HUB	40.0	34.2
LORA-FLOW	**41.2**	**35.4**

Table 2: Performance of different fusion methods on Llama-2-13b. The results are reported on the Zh math and Zh code test sets.

weights change across time steps in Section 5.2. Finally, we provide a specific case in Section 5.3.

5.1 Fusion Weights across Different Layers

Figure 3 shows the average fusion weights at different layers for the two involved LoRA modules (i.e., Zh Chat and En Math) on the Chinese math task. For both the Zh Chat and En Math LoRA, the weights at various layers exhibit notable variations, suggesting distinct fusion schemes across different model layers. A trend is observed where lower layers assign greater weights to the En Math LoRA, while higher layers allocate more weight to the Zh Chat LoRA. This phenomenon can be explained by the following reason: the bottom layers primarily utilize the math LoRA for reasoning, while the top layers rely more on the language LoRA for text generation in the target language. In contrast, approaches such as LoRA-Hub, which combines LoRA modules with fixed weights, regard the capabilities of each transformer layer as equivalent, overlooking the discrepancies in their abilities. This constrains the potential of LoRA fusion approaches in generative tasks.

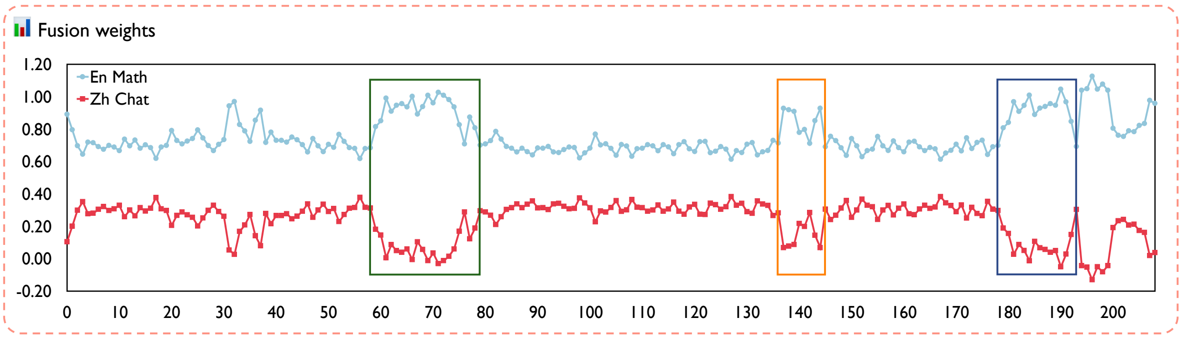
5.2 Fusion Weights at Different Time Steps

We also analyze the average fusion weights at different time steps during the decoding process. We split all the tokens generated by the model into 10 bins according to their relative positions. Figure 4 presents the results, where "10%" denotes the initial 10% tokens on the left, and "20%" represents the subsequent 10%-20% tokens. This figure illustrates the average trend of the fusion dynamics. We observe that the weight of the En Math LoRA is increasing, while the weight of the Zh Chat LoRA is decreasing. The results underscore the necessity for dynamic fusion weights in generative tasks, as they show that different time steps require varying fusion weights, reconfirming our intuition. Since the average trend is estimated based on the entire test set, we will present a specific case in the following section.

5.3 Detailed Analysis on Fusion Weights

To better investigate the fusion procedure of LoRA-Flow, we provide an example in Figure 5, which consists of the specific tokens generated by the model and the corresponding fusion weights for the two involved LoRA modules (i.e., Zh Chat and En Math). As discussed in Section 5.2, the fusion weights exhibit significant variation across different tokens. Notably, due to the incorporation of a bias vector in the fusion gate, as depicted in Eq (5), it is possible for certain tokens to have negative fusion weights. We draw inspiration from the work of Zhang et al. (2023), who observed that negative fusion weights can yield specific effects.

On average, the fusion gate assigns higher weights to the En math LoRA. Additionally, we



Input

贾马尔的手机能容纳的照片数量是布列塔尼手机的 6 倍。布列塔尼的手机能容纳的最大照片数量是贾马尔在动物园拍摄的鸭子照片中鸭子数量的 50 倍。如果贾马尔的手机能容纳 1,800 张照片，贾马尔在动物园拍摄的照片中能看到多少只鸭子？

Output

贾马尔的手机能容纳的照片数量是布列塔尼手机的 6 倍，那么布列塔尼手机能容纳的照片数量是 1,800 / 6 = 300 张。\n布列塔尼的手机能容纳的最大照片数量是贾马尔在动物园拍摄的鸭子照片中鸭子数量的 50 倍，那么贾马尔在动物园拍摄的照片中能看到的鸭子数量是 300 / 50 = 6 只。\n#### 6\n答案是： 6

Figure 5: Detailed analysis for the fusion procedure of LoRA-Flow. The upper subgraph illustrates the fusion weights for each token, while the bottom subgraph details the content. From the fusion weights, we observe three segments where the fusion weights for the Zh Chat LoRA noticeably decrease while those for the En Math LoRA increase. We highlight the tokens corresponding to these segments using green, yellow, and red colors, respectively. Surprisingly, these three segments mainly contain numbers, which are closely related to mathematical reasoning ability. We also offer English translations of the input and output Chinese text in Figure 7 in the Appendix.

observe certain troughs in the fusion weights for the Zh Chat LoRA. To better understand this phenomenon, we identify the tokens corresponding to these troughs. As shown in Figure 5, we use the same color to mark the weight throughs and the corresponding text segments. Surprisingly, we observed that when the fusion weight for the Zh chat LoRA decreases, the text segments mainly contain numbers and mathematical calculations (e.g., "1800 / 6 = 300"), which align more closely with the math capability learned by the En math LoRA.

By examining the case, we observe a strong correlation between the fusion weights with the generated content. When the content involves mathematical reasoning, LoRA-Flow will improve the fusion weight of the En math LoRA. The significant fluctuation in weights suggests that relying on fixed weights throughout the decoding process is impractical, underscoring the necessity of dynamic fusion weights.

6 Discussion

6.1 Ablation Study

To further investigate the granularity of the gates in LoRA-Flow, we conduct an ablation study to compare the performance of different types of gates: step-level, layer-level, and module-level gates. As explained in Section 3.1, the step-level gate estimates the fusion weight for each step, which is shared by the entire model. Layer-level gates represent the default setting in LoRA-Flow, computing the fusion weight at each layer. Module-level gates provide a specific fusion weight for each module (e.g., the query projection module in the last layer), offering greater flexibility than layer-level gates but introducing more trainable parameters.

Method	MGSM			
	Zh	Ru	Es	Avg.
TASK LORA	26.8	32.8	41.2	33.6
LORA-HUB	20.8	28.4	36.8	28.7
STEP-LEVEL	30.0	32.4	**44.0**	35.5
LAYER-LEVEL	**33.2**	**37.6**	42.0	**37.6**
MODULE-LEVEL	30.4	34.0	42.4	35.6

Table 3: Ablation study on various levels of fusion gates.

Table 3 presents the results of varying gate levels. On average, layer-level fusion gates achieve the highest scores, surpassing both step-level and module-level gates. On the Zh math task, the confidence intervals for the step-, layer-, and module-level fusion models are (29.21, 30.78), (32.85, 33.55), and (29.61, 31.18), respectively. Nevertheless, the other two fusion gate types also outperform LoRA-Hub. Specifically, the step-level fusion gate achieves an average score of 35.5, while that of LoRA-Hub is 28.7. These results suggest that while utilizing shared fusion weights across different model layers, employing dynamic fusion methods at various time steps yields superior results compared to using task-level static weights. Since different model layers may have different capabilities, using layer-level fusion gates can further improve the performance.

Method	MGSM	HumanEval
FT New LoRA	18.8	12.2
FT Lang LoRA	16.4	15.9
FT Task LoRA	27.6	18.9
LoRA-Flow	**33.2**	**20.7**

Table 4: Comparison between few-shot fine-tuning and LoRA fusion. The results are reported on the Zh math and Zh code tasks.

6.2 LoRA Fusion vs. Few-shot Fine-Tuning

In scenarios with limited data, combining existing LoRA modules can effectively leverage the knowledge acquired in other tasks. In this section, we compare LoRA-Flow with other few-shot fine-tuning baselines: (1) FT new LoRA: using the same few-shot training examples to learn a new LoRA for the target task; (2) FT lang LoRA: fine-tuning the language LoRA on the target task; and (3) FT task LoRA: fine-tuning the task LoRA on the target task. The results are shown in Table 4. Our method, which dynamically integrates the language and task LoRA modules, surpasses all three few-shot fine-tuning baselines, indicating the effectiveness of utilizing existing LoRA modules in few-shot scenarios.

6.3 Generalization across Different Tasks

Similar to LoRA-Hub (Huang et al., 2024), we learn the fusion module for each task with few-shot training data. In this study, we also explore the generalizability of fusion gates across various tasks. For example, can the gate learned for the Zh code task, which is trained to combine the Zh chat LoRA and the En code LoRA, effectively combine the Ru chat LoRA and the En math LoRA?

Figure 6 shows the results, depicting the performance difference between merging two LoRA modules and employing solely the task LoRA, which is

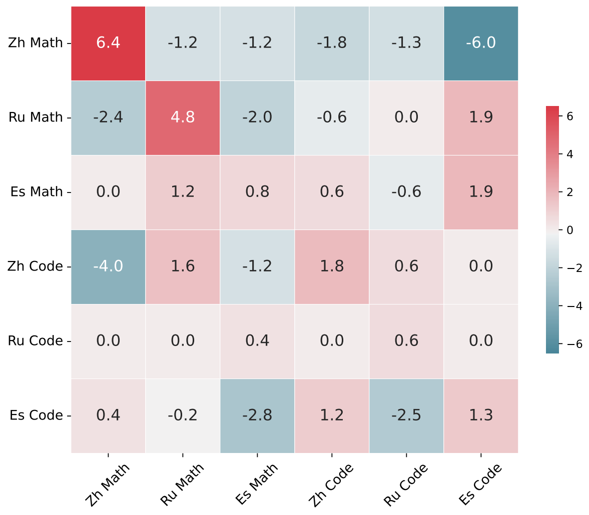


Figure 6: Generalizability of the fusion gates. Each row represents a training task, and each column represents an evaluation task. The value represents the performance gap between combining the language and task LoRA modules and using the task LoRA only. Please refer to Section 6.3 for detailed explanations.

a strong baseline. Specifically, the initial row indicates employing the gate trained with Zh math data for all six generative tasks. Remarkably, numerous values appear to be positive, suggesting that while zero-shot generalization is not a specific consideration during method design, it still demonstrates some degree of generalization capability. Take the third row as an example, training a gate on the Es math task results in a performance enhancement of 0.8 on the training task. Impressively, this Es math gate also leads to improved performance on three other tasks (i.e., Ru math, Zh code, and Es code). Furthermore, the enhancement observed in the Es code task surpasses that achieved in the Es math task. We leave it to future work to further improve the zero-shot generalization capabilities of LoRA fusion methods.

6.4 Few-Shot Sample Size Selection

We also investigate the impact of the number of few-shot learning examples (i.e., N). On the Zh math task, when trained with 50, 100, and 200 examples, LoRA-Flow achieves scores of 26.8, 31.6, and 33.2, respectively. This trend indicates that using more data can yield better fusion results.

7 Related Work

Module Composition Exploring the reuse of existing modules for new tasks is an appealing research direction. Pfeiffer et al. (2021) combines adapters (Houlsby et al., 2019) that have been fine-tuned on various downstream tasks through an additional attention layer. Zhang et al. (2023) defines some specific operations, such as addition and subtraction, to combine existing LoRA modules. The combination weights are tuned on a validation set. Huang et al. (2024) further improves the LoRA combination by automatically optimizing the fusion weights in a few-shot manner. Chronopoulou et al. (2023) combines LoRA modules trained on the summarization task and multilingual unlabeled data to perform multilingual summarization, where the fusion weight is also a hyperparameter that should be tuned. Most previous LoRA fusion methods employ task-level static weights, while our proposed LoRA-Flow can dynamically combine different types of LoRA according to the current context.

Mixture-of-LoRA Some recent studies propose to improve the performance of LoRA with the mixture-of-LoRA (MoLoRA) architecture (Zadouri et al., 2023; Dou et al., 2023; Wang et al., 2022), which is similar to mixture-of-expert models. The primary objective of these efforts is to overcome the limitations of LoRA in certain downstream tasks, as the expressive capacity of a single LoRA may be restricted by its intermediate rank. The major difference between MoLoRA and our work is that MoLoRA mainly aims to train a better plugin for the backbone model, whereas we aim to leverage pre-trained LoRA modules for new tasks without training a new LoRA. These two approaches are complementary. Our method can also be extended to incorporate existing MoLoRAs or other advanced types of LoRA modules for unseen tasks.

8 Conclusion

In this work, we propose LoRA-Flow, a dynamic combination method for LoRA. By assigning dynamic weights to different LoRA modules based on the current context, LoRA-Flow can outperform representative baselines with static task-level fusion weights on six complex generative tasks.

9 Limitation

In this study, LoRA-Flow outperforms the fixed-weight linear combination method across tasks in various languages. However, our computing resources are limited, constraining us to models no larger than 13 billion parameters. We only conduct experiments combining two adapters, though our method is readily adaptable to incorporate additional adapters by simply integrating more into the model and adjusting the gate's shape accordingly. Nonetheless, the versatility of our method persists, and we leave the exploration of larger models and combining more adapters to future work.

Acknowledgements

This project was supported by the National Natural Science Foundation of China (No. 62106138, No. 62236004, No. 62236011), and a grant from the Guoqiang Institute, Tsinghua University. We thank all the reviewers for their constructive and insightful suggestions.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, Sebastian Ruder, Yi Tay, Kefan Xiao, Yuanzhong Xu, Yujing Zhang, Gustavo Hernandez Abrego, Junwhan Ahn, Jacob Austin, Paul Barham, Jan Botha, James Bradbury, Siddhartha Brahma, Kevin Brooks, Michele Catasta, Yong Cheng, Colin Cherry, Christopher A. Choquette-Choo, Aakanksha Chowdhery, Clément Crepy, Shachi Dave, Mostafa Dehghani, Sunipa Dev, Jacob Devlin, Mark Díaz, Nan Du, Ethan Dyer, Vlad Feinberg, Fangxiaoyu Feng, Vlad Fienber, Markus Freitag, Xavier Garcia, Sebastian Gehrmann, Lucas Gonzalez, Guy Gur-Ari, Steven Hand, Hadi Hashemi, Le Hou, Joshua Howland, Andrea Hu, Jeffrey Hui, Jeremy Hurwitz, Michael Isard, Abe Ittycheriah, Matthew Jagielski, Wenhao Jia, Kathleen Kenealy, Maxim Krikun, Sneha Kudugunta, Chang Lan, Katherine Lee, Benjamin Lee, Eric Li, Music Li, Wei Li, YaGuang Li, Jian Li, Hyeontaek Lim, Hanzhao Lin, Zhongtao Liu, Frederick Liu, Marcello Maggioni, Aroma Mahendru, Joshua Maynez, Vedant Misra, Maysam Moussalem, Zachary Nado, John Nham, Eric Ni, Andrew Nystrom, Alicia Parrish, Marie Pellat, Martin Polacek, Alex Polozov, Reiner Pope, Siyuan Qiao, Emily Reif, Bryan Richter, Parker Riley, Alex Castro Ros, Aurko Roy, Brennan Saeta, Rajkumar Samuel, Renee Shelby, Ambrose Slone, Daniel Smilkov, David R. So, Daniel Sohn, Simon Tokumine, Dasha Valter, Vijay Vasudevan, Kiran Vodrahalli, Xuezhi Wang, Pidong Wang, Zirui Wang, Tao Wang, John Wieting, Yuhuai Wu, Kelvin Xu, Yunhan Xu, Linting Xue, Pengcheng Yin, Jiahui Yu, Qiao Zhang, Steven Zheng, Ce Zheng, Weikang Zhou, Denny Zhou, Slav Petrov, and Yonghui Wu. 2023. Palm 2 technical report.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming

Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Alexandra Chronopoulou, Jonas Pfeiffer, Joshua Maynez, Xinyi Wang, Sebastian Ruder, and Priyanka Agrawal. 2023. Language and task arithmetic with parameter-efficient layers for zero-shot summarization.

Shihan Dou, Enyu Zhou, Yan Liu, Songyang Gao, Jun Zhao, Wei Shen, Yuhao Zhou, Zhiheng Xi, Xiao Wang, Xiaoran Fan, Shiliang Pu, Jiang Zhu, Rui Zheng, Tao Gui, Qi Zhang, and Xuanjing Huang. 2023. Loramoe: Revolutionizing mixture of experts for maintaining world knowledge in language model alignment.

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 *Proceedings of the 36th International Conference on Machine Learning*.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Chengsong Huang, Qian Liu, Bill Yuchen Lin, Tianyu Pang, Chao Du, and Min Lin. 2024. Lorahub: Efficient cross-task generalization via dynamic lora composition.

Viet Lai, Chien Nguyen, Nghia Ngo, Thuat Nguyen, Franck Dernoncourt, Ryan Rossi, and Thien Nguyen. 2023. Okapi: Instruction-tuned large language models in multiple languages with reinforcement learning from human feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

Xiang Lisa Li and Percy Liang. 2021. 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)*.

OpenAI. 2023. Gpt-4 technical report.

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*.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2023. Language models are multilingual chain-of-thought reasoners. In *The Eleventh International Conference on Learning Representations*.

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. 2023a. Llama: Open and efficient foundation language models.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Dan Bikel, Lukas Blecher, Cristian Canton Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, Brian Fuller, Cynthia Gao, Vedanuj Goswami, Naman Goyal, Anthony Hartshorn, Saghar Hosseini, Rui Hou, Hakan Inan, Marcin Kardas, Viktor Kerkez, Madian Khabsa, Isabel Kloumann, Artem Korenev, Punit Singh Koura, Marie-Anne Lachaux, Thibaut Lavril, Jenya Lee, Diana Liskovich, Yinghai Lu, Yuning Mao, Xavier Martinet, Todor Mihaylov, Pushkar Mishra, Igor Molybog, Yixin Nie, Andrew Poulton, Jeremy Reizenstein, Rashi Rungta, Kalyan Saladi, Alan Schelten, Ruan Silva, Eric Michael Smith, Ranjan Subramanian, Xiaoqing Ellen Tan, Binh Tang, Ross Taylor, Adina Williams, Jian Xiang Kuan, Puxin Xu, Zheng Yan, Iliyan Zarov, Yuchen Zhang, Angela Fan, Melanie Kambadur, Sharan Narang, Aurelien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023b. Llama 2: Open foundation and fine-tuned chat models.

Elena Voita, Rico Sennrich, and Ivan Titov. 2019. The bottom-up evolution of representations in the transformer: A study with machine translation and language modeling objectives. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Yaqing Wang, Sahaj Agarwal, Subhabrata Mukherjee, Xiaodong Liu, Jing Gao, Ahmed Hassan Awadallah, and Jianfeng Gao. 2022. AdaMix: Mixture-of-adaptations for parameter-efficient model tuning. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 5744–5760, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T. Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. Metamath: Bootstrap your own mathematical questions for large language models.

Ted Zadouri, Ahmet Üstün, Arash Ahmadian, Beyza Ermiş, Acyr Locatelli, and Sara Hooker. 2023. Pushing mixture of experts to the limit: Extremely parameter efficient moe for instruction tuning.

Jinghan Zhang, Shiqi Chen, Junteng Liu, and Junxian He. 2023. Composing parameter-efficient modules with arithmetic operations. In *Advances in Neural Information Processing Systems*.

A Best Hyperparameters

The best hyperparameters for LoRA fusion are listed in the Table 5.

B Translation Prompt

Figure 8 and Figure 9 show the prompts we used to translate the Metameth data from English to Chinese using GPT-3.5. The {system prompt} and {user prompt} are set by default. The first prompt is intended to translate the problem identified by the {instruction} of a sample. The second prompt is intended to translate the solution identified by the {answer} to a sample.

C Human Annotation Details

In the process of reviewing and polishing multilingual data, we employed two annotators for each language. For the construction of the English version of the code validation set, we hired two annotators with over two years of experience in NLP. We contacted them through social media and compensated them at market-standard rates. It has been communicated to all participating human annotators that the gathered data shall only be used for academic research purposes, with an assurance that their personal information will be kept confidential and not disclosed.

C.1 Details of Translated Data Verification

We employed college students majoring in the relevant languages to check and optimize the translated data. The primary areas of review and optimization include:

- Correctness of the mathematical problem-solving process.
- Correctness of the mathematical problem format: whether the expression of formulas and the organization of structures adhere to the original metamath template, such as answers following "####" and ending with "the answer is:" in various languages, etc.
- Whether the code part of the code generation problems remains consistent with that in the original data.
- Correctness of the code data format: whether the overall structure remains unchanged, and whether essential parts such as problem description, problem analysis, and code comments are translated into the corresponding language.
- Overall fluency of the translated sentence.

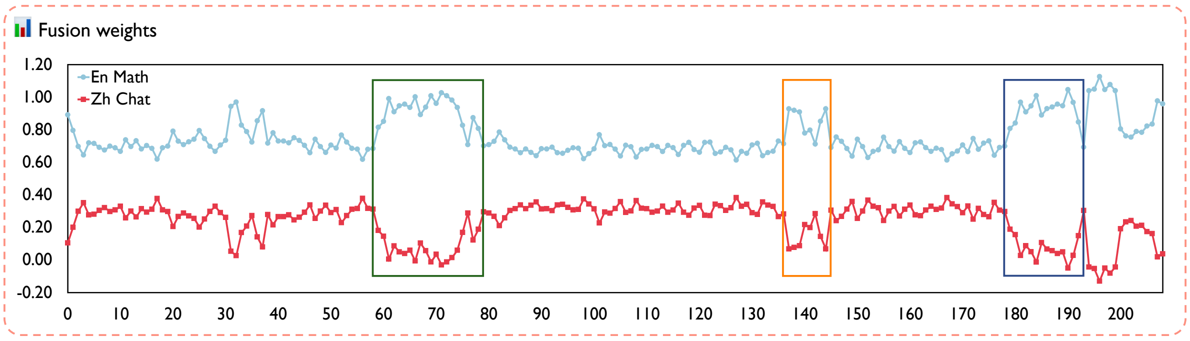
Each piece of data is sequentially reviewed by two independent annotators.

D Translated Case

To better understand the case study presented in Figure 5, we translate the Chinese question and answer into English, the results are shown in Figure 7.

Tasks	Zh Math	Ru Math	Es Math	Zh Code	Ru Code	Es Code
batch_size	4	4	8	4	4	4
learning_rate	1e-3	1e-3	1e-4	1e-3	1e-3	1e-3

Table 5: The best hyperparameters for LoRA fusion.



Input

贾马尔的手机能容纳的照片数量是布列塔尼手机的 6 倍。布列塔尼的手机能容纳的最大照片数量是贾马尔在动物园拍摄的鸭子照片中鸭子数量的 50 倍。如果贾马尔的手机能容纳 1,800 张照片，贾马尔在动物园拍摄的照片中能看到多少只鸭子？

English Translation: *Jamal's phone holds six times as many photos as Brittany's phone. The maximum number of photos Brittany's phone can hold is 50 times the number of ducks in Jamal's photo of ducks at the zoo. If Jamal's phone could hold 1,800 photos, how many ducks could Jamal see in the photo he took at the zoo?*

Output

贾马尔的手机能容纳的照片数量是布列塔尼手机的 6 倍，那么布列塔尼手机能容纳的照片数量是 1,800 / 6 = 300 张。\n布列塔尼的手机能容纳的最大照片数量是贾马尔在动物园拍摄的鸭子照片中鸭子数量的 50 倍，那么贾马尔在动物园拍摄的照片中能看到的鸭子数量是 300 / 50 = 6 只。\n#### 6\n答案是： 6

English Translation: *Jamal's phone can hold 6 times as many photos as Brittany's phone, so Brittany's phone can hold 1,800 / 6 = 300 photos. \nThe maximum number of photos that Brittany's phone can hold is 50 times the number of ducks in Jamal's photo of ducks at the zoo, so the number of ducks that Jamal can see in the photo of Jamal at the zoo is 300 / 50 = 6. \n#### 6\nThe answer is: 6*

Figure 7: Case study of LoRA-Flow with English translations.

{system prompt}:
将以下数学题中题干翻译为汉语，你必须要遵循以下几点要求：
1.保持文本中的数字、符号与LaTex转义字符不变。LaTex转义字符是指以反斜杠开头的字符串，如 \frac{1}{2}，\cdot 等，这些字符串通常被 '$' 符号包围，如 '$x = \frac{1}{2}$'。在翻译过程中将这些符号复制即可。
2.剩余文本内容全部翻译成汉语，特别是人名，人名翻译成汉语中最常见的对应形式。如将 "Joe"翻译成 "乔"。
3.在翻译过程中需要保证用词准确与严谨，需要将原文中的数学相关词语准确专业地翻译成汉语中相应的数学术语，并且尽可能地使得语言自然而流畅。
4.在翻译过程中需要保证内容完整，含义与原文一致，不允许添加或删除任何信息。
示例：
原文：
<text>Gracie and Joe are choosing numbers on the complex plane. Joe chooses the point $1+2i$. Gracie chooses $-1+i$. How far apart are Gracie and Joe's points?<\text>
翻译：
格蕾丝和乔在复平面上选择数字。乔选择了点$1+2i$。格蕾丝选择了$-1+i$。格蕾丝和乔选择的点有多远?

{user prompt}:
原文：
{instruction}
翻译:\n

Figure 8: Prompt for instruction translation.

{system prompt}:
将以下数学题中回答翻译为汉语，你必须要遵循以下几点要求：
1.保持文本中的数字、符号与LaTex转义字符不变。LaTex转义字符是指以反斜杠开头的字符串，如 \frac{1}{2}，\cdot 等，这些字符串通常被 '$' 符号包围，如 '$x = \frac{1}{2}$'。在翻译过程中将这些符号复制即可。
2.剩余文本内容全部翻译成汉语，特别是人名，人名翻译成汉语中最常见的对应形式。如将 "Joe"翻译成 "乔"。
3.在翻译过程中需要保证用词准确与严谨，需要将原文中的数学相关词语准确专业地翻译成汉语中相应的数学术语，并且尽可能地使得语言自然而流畅。
4.在翻译过程中需要保证内容完整，含义与原文一致，不允许添加或删除任何信息。
示例：
原文：
<text>Dave bought a total of 8 + 6 + 3 = 17 books.
Each book cost $6, so Dave spent a total of 17 x $6 = $102 on the books.
102
The answer is: 102<\text>
翻译：
戴夫买了总共8 + 6 + 3 = 17本书。
每本书的价格是6美元，所以戴夫在这些书上总共花费了17 x 6 = 102美元。
102
答案是：102

{user prompt}:
原文：
{answer}
翻译:\n

Figure 9: Prompt for answer translation.