arXiv:2508.04012v1 [cs.CL] 6 Aug 2025

Step More: Going Beyond Single Backpropagation in Meta Learning Based Model Editing

Xiaopeng Li, Shasha Li*, Xi Wang, Shezheng Song, Bin Ji, Shangwen Wang,
Jun Ma*, Xiaodong Liu, Mina Liu†, Jie Yu*
National University of Defense Technology
†KylinSoft
{xiaopengli,shashali,yj}@nudt.edu.cn, a506lm@126.com

Abstract

Large Language Models (LLMs) underpin many AI applications, but their static nature makes updating knowledge costly. Model editing offers an efficient alternative by injecting new information through targeted parameter modifications. In particular, meta-learning-based model editing (MLBME) methods have demonstrated notable advantages in both editing effectiveness and efficiency. Despite this, we find that MLBME exhibits suboptimal performance in low-data scenarios, and its training efficiency is bottlenecked by the computation of KL divergence. To address these, we propose **S**tep **M**ore **Edit** (**SMEdit**), a novel MLBME method that adopts **M**ultiple **B**ackpro**P**agation **S**teps (**MBPS**) to improve editing performance under limited supervision and a norm regularization on weight updates to improve training efficiency. Experimental results on two datasets and two LLMs demonstrate that SMEdit outperforms prior MLBME baselines and the MBPS strategy can be seamlessly integrated into existing methods to further boost their performance. Our code will be released soon.

1 Introduction

Large Language Models (LLMs) have emerged as foundational infrastructure for a wide range of AI applications [1–3]. Through extensive pretraining on diverse corpora followed by alignment, LLMs acquire rich world knowledge. However, the knowledge encoded in LLMs becomes fixed after training, making them unable to adapt to updates in the real world [4]. Re-training LLMs to reflect new knowledge is prohibitively expensive [5, 6].

To address this challenge, recent studies have introduced **model editing** as an efficient paradigm for updating knowledge in LLMs [7–15]. These approaches aim to make LLMs memorize new information by modifying only a small subset of parameters. Unlike vanilla fine-tuning, model editing typically computes a weight update that indirectly injects new knowledge into the model. For example, **locate-then-edit** methods perform least-squares updates on the feed-forward networks that are believed to store factual knowledge [16–19], while **meta-learning-based model editing (MLBME)** employs a lightweight hypernetwork to transform fine-tuning gradients and internal representations into effective weight updates [20, 11, 9]. Although locate-then-edit approaches achieve impressive editing performance, they suffer from low efficiency. For instance, in sequential editing tasks, AlphaEdit [10] (a representative locate-then-edit method) is approximately 27.8 $\times$ slower than RLEdit [11], a representative MLBME method. MLBME approaches exhibit promising performance both in terms of effectiveness and efficiency.

Despite the progress made in MLBME, we observe that it still has a couple of limitations. First, current MLBME methods perform well when abundant training data is available, but their performance is suboptimal under low-data scenarios (Section 4.1). This highlights a data efficiency limitation

Preprint. Under review.

in existing MLBME methods. Second, the training efficiency of MLBME is bottlenecked by the computation of KL divergence loss (Section 4.2), which requires the reference probability distribution from the original model, leading to double forward passes per iteration.

To tackle the above issues, we propose **Step More Edit (SMEdit)**, which performs model editing through **M**ultiple **B**ackpro**P**agation **S**teps (**MBPS**), leveraging multiple backpropagation (BP) steps to better learn editing patterns from the limited training data. This enables more effective editing with limited data, thereby addressing the first issue. The comparison between SMEdit and existing MLBME methods is illustrated in Figure 1. To improve training efficiency, SMEdit drops the KL loss in favor of an l_2 regularization on the weight update to preserve the original model's behavior, thus alleviating the second issue. Furthermore, SMEdit introduces a step-specific hypernetwork for sequential editing and a step-wise hypernetwork updating mechanism for batch editing, balancing editing effectiveness and efficiency.

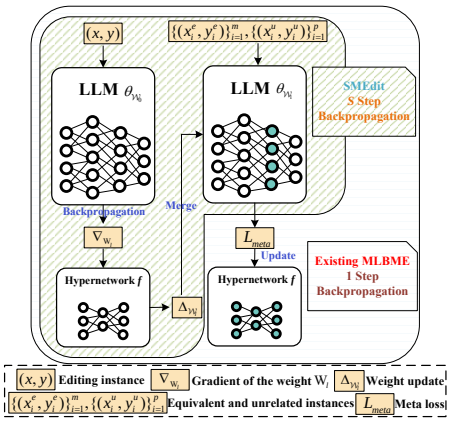

Figure 1: Existing MLBME performs model editing using a single BP, whereas SMEdit performs editing using MBPS.

We evaluate SMEdit on both batch and sequential editing tasks using GPT-J (6B) [21] and LLaMA-3 (8B) [22] on the ZsRE [7] and COUNTERFACT [8] datasets. Experimental results demonstrate the effectiveness of SMEdit. We also validate that incorporating MBPS consistently enhance the editing performance of existing MLBME baselines. In summary, our contributions are as follows:

- We reveal that current MLBME methods face limitations in both training data utilization and training efficiency. Their performance is suboptimal under low-data scenarios and training efficienc is bottlenecked by the computation of KL divergence.
- We propose using MBPS and norm constraints to address the limitations of current MLBME methods. Building on these insights, we introduce SMEdit, a novel MLBME approach that effectively balances editing performance and efficiency.
- We demonstrate the effectiveness of MBPS and SMEdit through batch and sequential editing experiments on two LLMs and two datasets.

2 Related Work

Model editing has recently emerged as a promising and efficient technique for updating the knowledge of large language models (LLMs) [23]. These approaches can be broadly categorized into two types based on whether they modify the original parameters of the model: **parameter-altering** methods and **parameter-preserving** methods [4].

Parameter-preserving methods store editing information externally—either by retrieving relevant knowledge from external sources [14], introducing auxiliary modules outside the model [24], or adding new internal parameters [25, 26]. These approaches have shown promising results on model editing tasks while mitigating the risks of altering the model's original behavior. However, the additional modules they require often increase system complexity and computational overhead, limiting their practicality in real-world applications.

In contrast, **parameter-altering** methods produce edited models that differ from the original ones only in their parameters [7–9]. This design makes them inherently more lightweight and flexible, allowing the edited models to be further edited or adapted to downstream tasks seamlessly. Such methods typically perform editing by modifying a small subset of the LLM's parameters. For example, locate-then-edit methods apply least-squares updates to key layers believed to store factual knowledge [16, 10, 17], achieving promising editing results. However, these methods often suffer from limited editing efficiency.

Meta-learning-based model editing (MLBME) represents a class of approaches that are competitive in both effectiveness and efficiency [20, 11]. These methods train a lightweight hypernetwork to transform single-step fine-tuning gradients and internal representations into effective weight updates

[9]. In this paper, we propose a novel MLBME method, SMEdit. Unlike existing MLBME methods, SMEdit leverages MBPS to make fuller use of the editing data, thereby enabling more expressive and effective model editing.

3 Preliminaries

3.1 Model Editing Problem

Model editing aims to efficiently modify a LLM θ to achieve specific editing goals without affecting its other capabilities. We define an editing sample as

$$T = (x, y) \cup \{(x_i^e, y_i^e)\}_{i=1}^m \cup \{(x_i^u, y_i^u)\}_{i=1}^p \tag{1}$$

where (x, y) is the editing instance used to measure the **efficacy** of model editing methods; $(x_i^e, y_i^e)i = 1^m$ are examples that are semantically equivalent to (x, y) and are used to evaluate the **generalization** ability; and $(x_i^u, y_i^u)i = 1^p$ are semantically unrelated examples used to assess the **specificity** of the model editing methods. Additionally, to assess whether the model retains its original capabilities after editing, existing works evaluate the performance of edited models on downstream natural language processing tasks (e.g., GLUE [27]). **Types of Model Editing:** Model editing methods can be classified into three scenarios: **sequential editing**, **batch editing**, and **sequential batch editing** [28]. Given n editing samples $\{T_1, T_2, \ldots, T_n\}$ to be edited, the three scenarios can be described as follows:

- **Sequential Editing:** This scenario continuously edits a single piece of knowledge, processing the n editing samples one by one.
- **Batch Editing:** This scenario performs a single edit operation on all n samples simultaneously, effectively editing multiple pieces of knowledge at once.
- **Sequential Batch Editing:** This scenario divides the n samples into batches of size b and performs $\lceil \frac{n}{b} \rceil$ sequential editing steps, each handling a batch.

After completing the editing process, the performance of the editing method is evaluated by measuring the success rate of the edited model across the n editing samples. The ultimate goal of all three scenarios is to effectively write the n knowledge samples into θ. However, their applicable scenarios, approaches to achieving the goal, and implementation challenges differ significantly.

3.2 Meta Learning Based Model Editing

Meta-learning-based model editing trains a hypernetwork to convert standard fine-tuning gradients into weight updates, reducing the computational burden of direct weight adjustment [9, 20, 11].

Editing Procedure: We denote the weight of θ as: $\mathcal{W} = \{W_l : l \in \mathcal{L}\}$ where W_l is the weight of linear layer l and $\mathcal{L}$ is the set of all linear layers. MEND [9] decomposes the single BP gradient ∇_{W_l} of the weight $W_l \in \mathbb{R}^{d' \times d}$ in layer l to be updated into a rank-1 product: $\nabla_{W_l} = \delta_{l+1} u_l^T$, where $\delta_{l+1} \in \mathbb{R}^{d' \times 1}$ is the gradient of L with respect to the preactivations to layer $l+1$ and $u_l \in \mathbb{R}^{d \times 1}$ are the inputs to layer l. The weight update $\Delta_{W_l} \in \mathbb{R}^{d' \times d}$ is obtained as the product of the *pseudogradient* and *pseudoactivations* transformed by a hypernetwork:

$$\Delta_{W_l} = \tilde{\delta}_{l+1} \tilde{u}_l^T, \text{where } \tilde{\delta}_{l+1}, \tilde{u}_l^T = f(\delta_{l+1}, u_l^T) \tag{2}$$

where $f : \mathbb{R}^d \times \mathbb{R}^{d'} \to \mathbb{R}^d \times \mathbb{R}^{d'}$ is the hypernetwork which transforms the gradient and inputs to *pseudogradient* $\tilde{\delta}_{l+1}$ and *pseudoactivations* $\tilde{u}_l^T$. MALMEN [20] improves the above process to accommodate massive batch editing scenarios by formulating a least squares problem that aggregates the weight updates corresponding to each editing sample into a single weight update. The weight update Δ_{W_l} of MALMEN is:

$$\Delta_{W_l} = D_l U_l^T (U_l U_l^T + \lambda_l I)^{-1} \text{ by solving: } \min_{\Delta_{W_l} \in \mathbb{R}^{d' \times d}} ||\Delta_{W_l} U_l - D_l||_2^2 + \lambda ||\Delta_{W_l}||_2^2 \tag{3}$$

where $D_l = [..., d_{l,k}, ...] = [..., \tilde{\delta}_{l+1,k} \tilde{u}_{l,k}^T u_{l,k}, ...] \in \mathbb{R}^{d' \times b}$ and $U_l = [..., u_{l,k}, ...] \in \mathbb{R}^{d \times b}$, with k representing the k-th editing sample in a batch of size b.

Training Procedure: The purpose of this process is to obtain a hypernetwork f that can transform gradients and activations into weight updates capable of achieving the model editing objective. The goal of model editing is to perform the edit while maintaining good generalization and specificity, which corresponds to the following two loss functions:

$$L_e = -\log \theta_{\mathcal{W}_1}(y^e|x^e), L_{loc} = \text{KL}\left[\theta_{\mathcal{W}_0}(\cdot|x^u)||\theta_{\mathcal{W}_1}(\cdot|x^u)\right] \tag{4}$$

where $\theta_{\mathcal{W}_0}$ and $\theta_{\mathcal{W}_1}$ denote the LLM with the original weights and the updated weights after one edit, respectively. The meta loss for training f is the sum of L_e and L_{loc}: $L_{meta} = L_e + \lambda_{loc} L_{loc}$ where λ_{loc} is a hyperparameter that balances generalization and specificity.

Based on the meta loss, different methods adopt various implementation details. MEND back-propagates L_{meta} into f via LLM θ [9], while MALMEN back-propagates on θ and f separately to avoid high memory costs [20]. RLEdit introduces a reinforcement learning framework into the above process, thus enabling the hypernetwork to perform sequential model editing in θ [11]. Specifically, RLEdit optimizes the following reward to obtain the optimal parameters of f:

$$J = \sum_{i=1}^{n} \gamma^i (L_{meta_i} + L_{back_i} + \eta||\Delta_{\mathcal{W}_i}||^2) \tag{5}$$

where the term $\eta||\Delta_{\mathcal{W}_i}||^2$ is the regularization term, with coefficient η, that accounts for the weight updates $\Delta_{\mathcal{W}_i}$ at edit step i. γ^i is the discount factor. The term L_{back_i} is the memory backtracking loss. RLEdit computes the meta loss of the current editing sample T_i and the meta loss of previous q editing samples $\{T_{i-1}, ..., T_{i-q}\}$ on $\theta_{\mathcal{W}_{i-1}}$ at edit step i. The backtracking loss is defined as:

$$L_{back_i} = \sum_{j=i-q}^{i-1} \mu^{i-j}(L_{e_j,\mathcal{W}_{i-1}} + \lambda_{loc} L_{loc_j,\mathcal{W}_{i-1}}) \tag{6}$$

where μ is a decay factor.

4 Limitations of MLBME

In this study, we use two state-of-the-art MLBME methods as research subjects and baselines: RLEdit [11] for sequential editing and MALMEN [20] for batch editing.

4.1 Data Efficiency Limitation

We observe that MLBME's performance follows a saturating growth trend with respect to the amount of training data it receives. In Figure 2, we show how MLBME's editing performance varies with the number of sequential edits and training steps in batch edits.

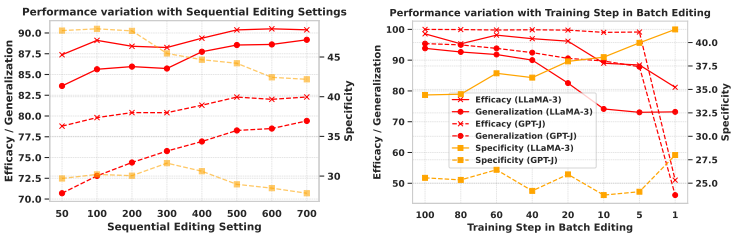

Figure 2: Performance Variation of MLBME in Sequential and Batch Editing. The batch size for sequential editing is 20, and the X-axis represents the number of sequences.

In the sequential editing setting[1], the overall editing performance increases as the number of sequential edits grows. This result is somewhat counterintuitive, as one might expect sequential model editing to become more challenging with a greater number of edits. Surprisingly, this suggests that RLEdit performs better on more complex sequential editing tasks. We hypothesize that longer editing sequences expose the model to more training instances, thereby enhancing its performance.

To test this hypothesis, we conduct batch editing experiments involving 1,000 batch edits, varying the number of training steps to simulate different amounts of training data. At each step, the number of

[1]In our sequential editing experiments with RLEdit, the number of training samples is matched to the number of sequential edits, following the same setup as in [11].

training samples is equal to the batch size, and the training data differed across steps. As shown in Figure 2 (right), editing performance decreases as the number of training steps is reduced. Especially, when the number of training steps is 1—meaning the amount of training data is equal to the batch size—the editing performance of MLBME drops significantly. This further supports our hypothesis, leading to the conclusion that **the performance of MLBME improves with increased training data, and that when training data is scarce, MLBME suffers a substantial performance degradation.** Given that training data is often scarce or hard to acquire in real-world applications [29], the **current design of MLBME yields suboptimal editing performance in low-resource settings**, thereby constraining its practical utility.

4.1.1 Solutions to Mitigate the Data Efficiency Limitation

The most direct way to mitigate the limitations of MLBME is to increase the amount of training data. However, this approach is difficult to implement in data-scarce scenarios. Another option is to modify the training strategy of MLBME. We observe that the current MLBME performs model editing using only a single-step gradient, which is equivalent to performing single BP. As a result, MLBME may not fully capture the editing patterns in the training process. To verify this, we modify MLBME to adopt a multi-step gradient-based model editing approach, where MBPS are performed on a batch of data to achieve model editing.

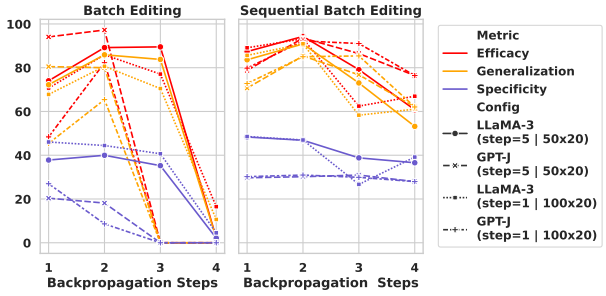

Figure 3: Performance Variation of MLBME in Sequential and Batch Editing Across Different BP Steps. The experiments are conducted on the ZsRE dataset.

We show the performance variation of MLBME in both batch and sequential batch editing settings across different numbers of BP steps in Figure 3. In the batch editing setting (left), we set the number of batches as 8,192 and "step=5/1" indicates that the training step of MLBME is 5 or 1. We limit the number of training steps to simulate the scarcity of training data in real-world scenarios. The efficacy and generalization of MLBME generally exhibit a trend of first increasing and then decreasing, while specificity consistently decreases as the number of BP steps increases. In the sequential batch editing (right figure in Figure 3), the trends observed here are similar to those in batch editing, but the best editing performance is achieved when the step size is 2. This result is intuitive: more BP steps mean more updates, and performing multiple updates on a single batch can lead to overfitting to that batch, thereby increasing efficacy and generalization but reducing specificity. Moreover, when the number of BP steps exceeds a certain threshold, the editing performance drops sharply, which may be due to excessive updates causing the editing collapse. This suggests that **MBPS can indeed achieve better editing performance**, and the choice of step size is a critical factor, with a step size of 2 yielding the overall best results.

4.2 Training Efficiency Limitation of MLBME

Although MLBME demonstrates strong editing efficiency, its training efficiency remains suboptimal. To investigate this issue, we conduct an in-depth analysis of the training time distribution for two representative MLBME methods, MALMEN and RLEdit, under both batch editing and sequential batch editing settings. Specifically, we perform batch editing with a batch size of 8,192 and sequential batch editing with 30×200 updates using MALMEN and RLEdit on the ZsRE and COUNTERFACT datasets, employing GPT-J and LLaMA-3 (8B) as the underlying models. We measure the average time consumption of the five steps in the MLBME training process: 1. Caching ∇_{W_l}; 2. Computing Δ_{W_l}; 3. Computing L_e and BP; 4. Computing L_{loc} and BP; 5. Updating f.

Figure 4 shows the time distribution of MLBME. In both the Batch and Sequential Batch experiments, Step 4 (computing L_{loc} and BP) accounts for a significant portion of the total training time—29.1% and 43.2%, respectively. This step is designed to protect the model's original knowledge. This overhead is expected because calculating the KL loss requires two forward passes through the LLMs (one for the original model output and one for the edited model output),

leading to increased computational cost. Although Step 1 (Cache ∇_{W_l}) consumes more time than Step 4 in the Batch Editing setting, it involves collecting the original fine-tuning gradients and internal representations, which is a fundamental step in MLBME. Notably, its time cost is the lowest in Sequential Batch Editing, indicating that it is not the primary bottleneck.

From the above analysis, we can conclude that **the training bottleneck of MLBME lies in Step 4: Compute L_{loc} and BP.** Therefore, we need to find an alternative way to efficiently preserve the model's original knowledge. To achieve this, we introduce the l_2-norm constraint as an alternative in Section 5.1, and demonstrate in the batch editing and sequential editing experiments (Section 6) that this alternative can achieve the aforementioned goal.

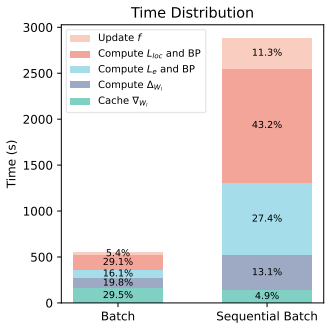

Figure 4: MLBME's Time Distribution.

5 Method

Inspired by the above analysis, we propose a novel MLBME method, **S**tep **M**ore **Edit** (**SMEdit**), which achieves more effective editing performance through **MBPS**. Following the same editing pipeline as MLBME, SMEdit consists of an ***Editing Procedure*** and a ***Training Procedure***. However, SMEdit incorporates *MBPS-based editing, step-specific hypernetwork for sequential editing, step-wise hypernetwork updating for batch editing, and an efficient and lightweight training process*. Due to the differences in the nature of sequential editing and batch editing tasks, the training process of SMEdit varies slightly. We describe them in detail below.

5.1 Lightweight Training

As shown in Section 3.2, previous MLBME methods typically use the weighted sum of the generalization loss L_e and the specificity loss L_{loc} as the meta-loss for training the hypernetwork. While this meta-loss effectively balances the trade-off between editing new knowledge and preserving original knowledge, we show in Section 4.2 that computing the loss L_{loc} leads to inefficient training. To improve training efficiency while preserving the original model's knowledge as much as possible, inspired by RLEdit, we replace the specificity loss with an l_2-norm constraint loss L_{cons} on the weight updates. Therefore, the final meta-loss of SMEdit is formulated as:

$$L_{meta} = L_e + \eta L_{cons} = L_e + \eta \sum_{s=0}^{S} \|\Delta_{s,\mathcal{W}_t}\|^2 \tag{7}$$

5.2 Step-specific Hypernetwork for Sequential Editing

SMEdit achieves a multi-step optimization process similar to traditional deep learning training through MBPS. Considering that the gradient at each step is different in sequential editing and that, under a rational assumption, both the training loss and gradient decrease as the number of steps increases, we design different hypernetworks for each step of SMEdit's editing. Specifically, SMEdit performs model editing in S BP steps, where s-th step corresponds to a hypernetwork f_s. These hypernetworks collectively form a hypernetwork set $\mathcal{F} = \{f_1, \ldots, f_S\}$. The pseudo-code of SMEdit hypernetwork training for sequential editing is shown in Algorithm 1.

Algorithm 1 SMEdit Hypernetwork Training for Sequential Editing

Input: LLM $\theta_{\mathcal{W}_0}$, hyperparameters S and η, set of hypernetworks $\mathcal{F} = \{f_1, ..., f_S\}$
Output: Optimized hypernetworks $\mathcal{F}$
Sample $(T_i = \{x_i, y_i, x_i^e, y_i^e\})_{i=1}^n$
// Efficient and lightweight training process
for $t = 1$ **to** n **do**
  // MBPS
  for $s = 1$ **to** S **do**
    $L \leftarrow -\log \theta_{\mathcal{W}_{t-1}^{s-1}}(y_t \mid x_t)$
    Back-propagate L and cache $\nabla_{\mathcal{W}_{t-1}}$
    // Step-specific hypernetwork
    $\Delta_{s-1,\mathcal{W}_{t-1}} \leftarrow f_{s-1}(\nabla_{\mathcal{W}_{t-1}})$
    $\mathcal{W}_t^s \leftarrow \mathcal{W}_{t-1}^{s-1} + \Delta_{s-1,\mathcal{W}_{t-1}}$
  $L_{cons} \leftarrow \|\mathcal{W}_t^S - \mathcal{W}_0\|^2$
  $L_{e,i} \leftarrow -\log \theta_{\mathcal{W}_t}(y_i^e \mid x_i^e)$
  $L_i \leftarrow L_{e,i} + \eta L_{cons}$
  Back-propagate L_i
Update $\mathcal{F}$
return $\mathcal{F}$

Each hyper-network follows the same structure as in previous MLBME methods [9, 20]. However, since the gradient decreases as the number of steps increases, learning the transformation from gradient to pseudo-gradient becomes easier in later steps. Therefore,

we employ a rank decay strategy to alleviate the editing burden. Specifically, the rank of the hypernetwork at the s-th step is defined as $r_s = r \mid s$, where r is the predefined rank.

5.3 Step-wise Hypernetwork Updating for Batch Editing

Unlike sequential editing training, which uses a step-specific hypernetwork, batch editing training employs a single hypernetwork f to learn multi-step gradient transformations. This is based on our observation that a single hypernetwork can better capture the gradient transformation patterns in batch editing. Moreover, we observe that gradient accumulation during sequential editing requires a large amount of memory, especially when the batch size is large. Therefore, we adopt step-wise hypernetwork updating to reduce memory consumption, where the hypernetwork is updated in every backpropagation step. The pseudo-code of SMEdit hypernetwork training for batch editing is shown in Algorithm 2.

Algorithm 2 SMEdit Hypernetwork Training for Batch Editing

Input: LLM $\theta_{\mathcal{W}_0}$, hyperparameters S and η, a hypernetwork f
Output: An optimized hypernetwork f
Sample $(T_i = \{x_i, y_i, x_i^e, y_i^e\})_{i=1}^n$
// Efficient and lightweight training process
for $t = 1$ **to** n **do**
  // MBPS
  for $s = 1$ **to** S **do**
    // Step-wise hypernetwork updating
    $L \leftarrow -\log \theta_{\mathcal{W}_{t-1}^{s-1}}(y_t \mid x_t)$
    Back-propagate L and cache $\nabla_{\mathcal{W}_{t-1}}$
    $\Delta_{s-1,\mathcal{W}_{t-1}} \leftarrow f(\nabla_{\mathcal{W}_{t-1}})$
    $\mathcal{W}_t^s \leftarrow \mathcal{W}_{t-1}^{s-1} + \Delta_{s-1,\mathcal{W}_{t-1}}$
    $L_{cons} \leftarrow \|\mathcal{W}_t^S - \mathcal{W}_0\|^2$
    $L_{e,i} \leftarrow -\log \theta_{\mathcal{W}_t}(y_i^e \mid x_i^e)$
    $L_i \leftarrow L_{e,i} + \eta L_{cons}$
    Back-propagate L_i
    Update f
return f

6 Experiments

6.1 Experimental Settings

LLM We consider two LLMs: LLaMA-3-instruct (8B) [22] and GPT-J (6B) [21].

DataSets & Metrics We consider two common datesets in model editing: COUNTERFACT [8] and ZsRE [30]. In line with prior work [11, 20], we evaluate editing performance using three metrics: Efficacy, Generalization, and Specificity. Details of these metrics can be found in Appendix C.

Baselines Our work primarily addresses the limitations of MLBME. Therefore, we adopt the most advanced batch and sequential MLBME editing methods as our baselines: MALMEN [20] (batch) and RLEdit [11] (sequential), along with their naive improved variants introduced in Section 4.1.1, denoted as MALMEN^2 and RLEdit^2, where the superscript 2 indicates that they are enhanced variants of MLBME using two-step backpropagation.

Experimental Details We show all the experimental details in Appendix C.

6.2 Sequential Batch Model Editing

We first evaluate the performance of SMEdit on the sequential batch editing task. We compare SMEdit with RLEdit and RLEdit^2. Experiments are conducted on the ZsRE and COUNTERFACT datasets by editing GPT-J (6B) and LLaMA-3 (8B), respectively. The editing results under different combinations of batch sizes and sequence lengths are shown in Table 1. RLEdit^2 and SMEdit achieve comparable editing performance across datasets, models, and sequential batch editing settings, both significantly outperforming RLEdit. This demonstrates that **the MBPS strategy can effectively enhance the editing performance of RLEdit.**

In terms of efficiency, RLEdit achieves the highest valuation efficiency, which is intuitive as it performs editing using only single-step gradients, while SMEdit and RLEdit^2 rely on multi-step gradients. Regarding training efficiency, SMEdit performs best, followed by RLEdit, with RLEdit^2 performing the worst. The low training efficiency of RLEdit and RLEdit^2 primarily stems from the computation of the KL loss. Since SMEdit eliminates the use of KL loss, it achieves the best training efficiency despite adopting a multi-step gradient strategy. While RLEdit^2 delivers strong editing performance, its training efficiency is suboptimal. In contrast, **our proposed SMEdit achieves a favorable balance between efficiency and performance.** We demonstrate that the introduction of MBPS have no impact on the model's original capability retention in the editing method, with details provided in the Appendix B.

Batch×Seq	Method	GPT-J (6B)					Llama-3 (8B)				
		Eff.↑	Gen.↑	Spe.↑	Time (s) / Edit		Eff.↑	Gen.↑	Spe.↑	Time (s) / Edit	
					Train↓	Val↓				Train↓	Val↓
ZsRE											
30×600	**RLEdit**	83.76	81.47	29.13	0.50	**0.05**	90.16	88.78	42.04	0.67	**0.09**
	RLEdit^2	**93.88**	**90.54**	30.40	1.12	0.15	93.77	92.03	40.41	1.04	0.11
	SMEdit	93.61	90.34	**30.64**	**0.17**	0.16	**95.40**	**93.91**	**42.18**	**0.19**	0.11
30×500	**RLEdit**	82.07	79.11	29.64	0.50	**0.06**	89.87	88.47	41.33	0.53	**0.06**
	RLEdit^2	**93.43**	**90.14**	30.69	0.99	0.10	92.83	91.34	43.67	1.10	0.12
	SMEdit	93.11	89.66	**30.91**	**0.17**	0.10	**95.67**	**94.10**	**43.96**	**0.21**	0.11
30×400	**RLEdit**	81.31	77.87	30.60	0.49	**0.05**	89.17	87.79	43.26	0.52	**0.06**
	RLEdit^2	92.71	**89.00**	**31.28**	0.91	0.10	92.60	88.29	30.82	0.73	0.08
	SMEdit	**92.81**	88.83	31.10	**0.17**	0.10	**95.61**	**94.09**	**44.94**	**0.18**	0.11
30×300	**RLEdit**	80.63	76.51	31.09	0.48	**0.05**	88.38	86.44	44.06	0.53	**0.06**
	RLEdit^2	**92.60**	**88.29**	30.82	0.97	0.10	93.07	91.10	**45.93**	1.02	0.11
	SMEdit	92.18	87.65	**31.28**	**0.16**	0.10	**95.25**	**93.39**	45.10	**0.18**	0.11
COUNTERFACT											
20×400	**RLEdit**	61.34	45.76	**54.06**	0.53	0.15	63.95	57.56	41.72	0.54	**0.07**
	RLEdit^2	89.63	67.77	41.35	1.01	0.27	88.19	75.63	**42.06**	1.04	0.12
	SMEdit	**90.29**	**68.06**	40.64	**0.26**	**0.12**	**94.24**	**79.49**	40.56	**0.30**	0.14
20×300	**RLEdit**	61.58	47.28	**48.80**	0.49	**0.06**	64.59	58.04	**41.95**	0.50	**0.07**
	RLEdit^2	88.70	66.03	41.17	0.94	0.11	**93.83**	**82.82**	37.59	1.01	0.13
	SMEdit	**89.73**	**67.28**	42.30	**0.19**	0.11	93.53	78.60	41.46	**0.22**	0.13
20×200	**RLEdit**	64.33	46.56	**50.08**	0.33	**0.04**	65.71	55.54	**43.30**	0.34	**0.05**
	RLEdit^2	87.40	**61.84**	45.68	0.65	0.07	**93.79**	**78.86**	42.73	0.67	0.08
	SMEdit	**88.43**	60.45	44.41	**0.13**	0.07	92.13	76.94	42.15	**0.22**	0.09

Table 1: Comparison of Editing Methods across GPT-J (6B) and Llama-3 (8B) on ZsRE and COUNTERFACT datasets. Eff.: Effectiveness, Gen.: Generalization, Spe.: Specificity.

6.3 Batch Model Editing

We next evaluate the performance of SMEdit on the batch editing task. We compare SMEdit with MALMEN and MALMEN^2. Specifically, we perform 2^{10} to 2^{14} batch edits on the ZsRE dataset and 2^{10} to 2^{13} batch edits on the COUNTERFACT dataset. As shown in Figure 5, similar to the results in sequential editing, SMEdit and MALMEN^2 achieve comparable performance, both outperforming MALMEN. However, their specificity on the COUNTERFACT dataset is significantly lower than that of MALMEN. Despite this, the overall results still **demonstrate that the MBPS strategy can effectively enhance the editing performance of MALMEN.**

In terms of efficiency, MALMEN achieves the best training and validation efficiency, while MALMEN^2 performs the worst in both aspects. **SMEdit once again strikes a favorable balance between editing effectiveness and efficiency in the batch editing setting.**

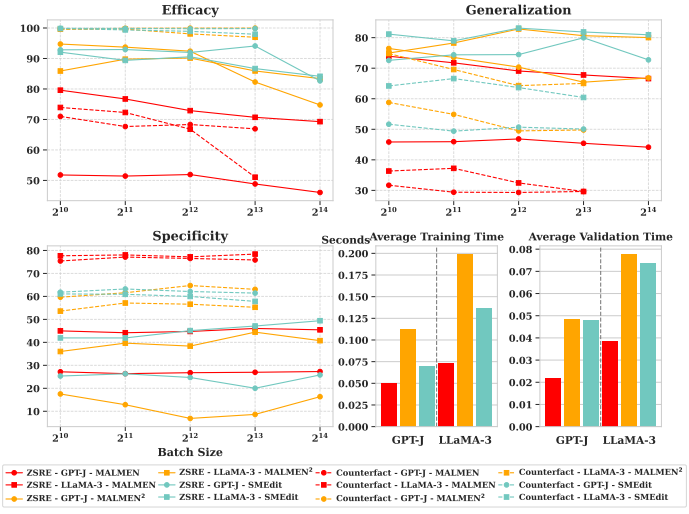

Figure 5: Batch editing results.

6.4 Ablation Study

In our ablation study, we verify the number of BP steps selected by SMEdit, as well as the effectiveness of *Step-specific Hypernetwork* (corresponding to *identical hypernetwork*) and *rank decay* (corresponding to *consis-*

tent rank) for sequential editing, and *Step-wise Hypernetwork Updating* (corresponding to *No step-wise*) and *Identical Hypernetwork* (corresponding to *diff hypernetwork*) for batch editing. We conduct 8192-sized batch editing and 30×400 sequential batch editing on the LLaMA-3 (8B) model using the ZsRE dataset. The results are shown in Table 2, with SMEdit results highlighted in cyan.

Regarding the number of BP steps, although a BP step of 3 achieves better editing performance in batch editing, its training and validation time nearly doubles. Choosing 2 steps achieves a better balance between performance and efficiency. Notably, when the BP step is set to 4, the performance drops to nearly zero, which is consistent with the results in Section 4.1.1, suggesting that excessive BP steps may lead to editing collapse. For sequential batch editing, using 2 BP steps also yields the best balance between performance and efficiency.

Table 2: Ablation Study.

Bath Size: 2^{13}					
Metrics	**Eff.**	**Gen.**	**Spe.**	**Train (s) / Edit**	**Val (s) / Edit**
BP Step 1	70.18	67.22	46.01	0.05	0.04
BP Step 2 (SMEdit)	85.91	80.64	44.44	0.11	0.07
BP Step 3	91.19	84.50	47.6	0.15	0.11
BP Step 4	0.62	0.45	0.46	0.20	0.14
Diff HyperNetwork	83.57	78.13	46.88	0.12	0.08
No Step-wise	Out of Memory				
Sequence Length×Bath Size: 30×400					
BP Step 1	89.31	87.57	43.47	0.11	0.06
BP Step 2 (SMEdit)	95.61	94.09	44.94	0.18	0.11
BP Step 3	88.08	85.86	36.88	0.29	0.19
BP Step 4	55.19	49.39	16.42	0.56	0.21
Identical HyperNetwork	93.57	91.83	38.66	0.35	0.12
Consistent Rank	94.52	92.53	44.02	0.20	0.12

In terms of the hypernetwork design, the results indicate that using different hypernetworks in batch editing and the same hypernetwork in sequential batch editing both lead to worse editing performance. This may be due to the differing nature of batch and sequential editing tasks. Additionally, maintaining a consistent hypernetwork rank in sequential batch editing does not yield better editing results. Furthermore, in batch editing, not using step-wise updating requires substantial memory for gradient accumulation—we also experimented with a batch size of 1024 and still encountered out-of-memory errors.

6.5 Loss Variation Across Different BP Steps

In this part, we analyze the trend of training loss across different BP steps in SMEdit's MBPS process. Since the differences in loss across BP steps are more pronounced in the batch editing experiments, we focus our analysis on this setting. Specifically, we record the loss at each BP step in the batch editing experiment. We show the loss trend in Figure 6. It can be observed that for both datasets and models, the loss at BP Step 2 is consistently lower than at BP Step 1. This indicates that the introduction of MBPS in MLBME has a similar effect to multi-epoch backpropagation in standard deep learning training, where the loss gradually decreases with more iterations. We presented more details in Appendix A

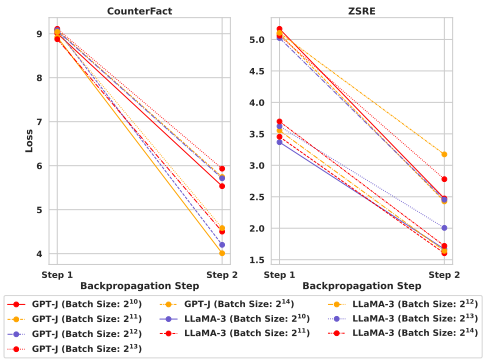

Figure 6: The variation of loss across different BP steps for different batch edit sizes.

7 Conlusion

We introduce SMEdit, a new meta-learning-based model editing method that addresses critical limitations of existing approaches in data and training efficiency. By leveraging MBPS, SMEdit enables effective editing even in low-data regimes. Furthermore, replacing KL divergence with l_2 regularization significantly reduces computational overhead without sacrificing performance. Our step-specific and step-wise hypernetwork designs improve editing performance and efficiency across sequential and batch settings. Extensive experiments on GPT-J and LLaMA-3 across ZsRE and COUNTERFACT demonstrate that SMEdit achieves state-of-the-art performance in both effectiveness and efficiency. Our findings also reveal that the MBPS strategy can enhance MLBME methods, offering a promising direction for future research in efficient knowledge editing.

References

[1] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[2] Bang Liu, Xinfeng Li, Jiayi Zhang, Jinlin Wang, Tanjin He, Sirui Hong, Hongzhang Liu, Shaokun Zhang, Kaitao Song, Kunlun Zhu, et al. Advances and challenges in foundation agents: From brain-inspired intelligence to evolutionary, collaborative, and safe systems. *arXiv preprint arXiv:2504.01990*, 2025.

[3] Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345, 2024.

[4] Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 10222–10240, Singapore, December 2023. Association for Computational Linguistics.

[5] 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. Llama 2: Open foundation and fine-tuned chat models, 2023.

[6] Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. Pythia: A suite for analyzing large language models across training and scaling. In *International Conference on Machine Learning*, pages 2397–2430. PMLR, 2023.

[7] Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. Modifying memories in transformer models. *arXiv preprint arXiv:2012.00363*, 2020.

[8] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 17359–17372. Curran Associates, Inc., 2022.

[9] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022.

[10] Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Xiang Wang, Xiangnan He, and Tat-seng Chua. Alphaedit: Null-space constrained knowledge editing for language models. *arXiv preprint arXiv:2410.02355*, 2024.

[11] Zherui Li, Houcheng Jiang, Hao Chen, Baolong Bi, Zhenhong Zhou, Fei Sun, Junfeng Fang, and Xiang Wang. Reinforced lifelong editing for language models. *arXiv preprint arXiv:2502.05759*, 2025.

[12] Zexuan Zhong, Zhengxuan Wu, Christopher Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15686–15702, Singapore, December 2023. Association for Computational Linguistics.

[13] Xiaohan Wang, Shengyu Mao, Ningyu Zhang, Shumin Deng, Yunzhi Yao, Yue Shen, Lei Liang, Jinjie Gu, and Huajun Chen. Editing conceptual knowledge for large language models. *arXiv preprint arXiv:2403.06259*, 2024.

[14] Ce Zheng, Lei Li, Qingxiu Dong, Yuxuan Fan, Zhiyong Wu, Jingjing Xu, and Baobao Chang. Can we edit factual knowledge by in-context learning? In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4862–4876, Singapore, December 2023. Association for Computational Linguistics.

[15] Andrew L Maas, Awni Y Hannun, Andrew Y Ng, et al. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3. Atlanta, GA, 2013.

[16] Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer, 2022.

[17] Xiaopeng Li, Shanwen Wang, Shasha Li, Shezheng Song, Bin Ji, Jun Ma, and Jie Yu. Rethinking the residual distribution of locate-then-editing methods in model editing. *arXiv preprint arXiv:2502.03748*, 2025.

[18] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics.

[19] Mor Geva, Jasmijn Bastings, Katja Filippova, and Amir Globerson. Dissecting recall of factual associations in auto-regressive language models. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 12216–12235, Singapore, December 2023. Association for Computational Linguistics.

[20] Chenmien Tan, Ge Zhang, and Jie Fu. Massive editing for large language models via meta learning, 2024.

[21] Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. `https://github.com/kingoflolz/mesh-transformer-jax`, May 2021.

[22] AI@Meta. Llama 3 model card. 2024.

[23] Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, Siyuan Cheng, Ziwen Xu, Xin Xu, Jia-Chen Gu, Yong Jiang, Pengjun Xie, Fei Huang, Lei Liang, Zhiqiang Zhang, Xiaowei Zhu, Jun Zhou, and Huajun Chen. A comprehensive study of knowledge editing for large language models, 2024.

[24] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR, 2022.

[25] Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models, 2024.

[26] Tom Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with grace: Lifelong model editing with discrete key-value adaptors. *Advances in Neural Information Processing Systems*, 36, 2024.

[27] Alex Wang. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018.

[28] Vittorio Mazzia, Alessandro Pedrani, Andrea Caciolai, Kay Rottmann, and Davide Bernardi. A survey on knowledge editing of neural networks. *arXiv preprint arXiv:2310.19704*, 2023.

[29] Hang Li. Deep learning for natural language processing: advantages and challenges. *National Science Review*, 5(1):24–26, 09 2017.

[30] Zero-shot relation extraction via reading comprehension. In *CoNLL 2017 - 21st Conference on Computational Natural Language Learning, Proceedings*, CoNLL 2017 - 21st Conference on Computational Natural Language Learning, Proceedings, pages 333–342. Association for Computational Linguistics (ACL), 2017.

[31] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013.

[32] Bill Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Third international workshop on paraphrasing (IWP2005)*, 2005.

[33] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[34] Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. The fifth pascal recognizing textual entailment challenge. *TAC*, 7(8):1, 2009.

[35] A Warstadt. Neural network acceptability judgments. *arXiv preprint arXiv:1805.12471*, 2019.

[36] Adina Williams, Nikita Nangia, and Samuel R Bowman. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

[37] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, et al. Kilt: a benchmark for knowledge intensive language tasks. *arXiv preprint arXiv:2009.02252*, 2020.

[38] Yanai Elazar, Nora Kassner, Shauli Ravfogel, Abhilasha Ravichander, Eduard Hovy, Hinrich Schütze, and Yoav Goldberg. Measuring and improving consistency in pretrained language models. *Transactions of the Association for Computational Linguistics*, 9:1012–1031, 2021.

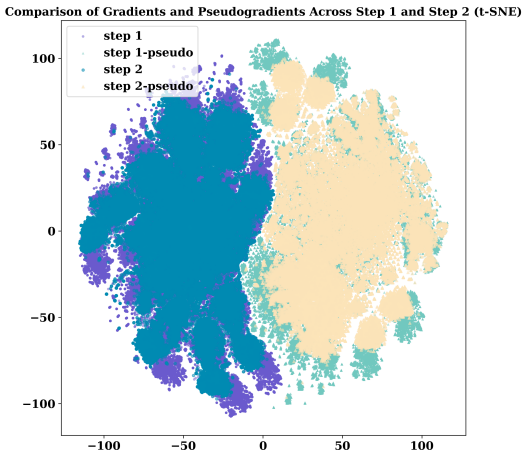

Figure 8: t-SNE visualization of gradients and pseudo-gradients after dimensionality reduction.

A More Details of the Loss Variation

We present the variation of the loss norm across different BP steps with respect to the editing batch size. As shown in Figure 7, it can also be observed that for both datasets and models, the loss at BP Step 2 is consistently lower than at BP Step 1. Moreover, the overall increase in loss with larger batch sizes aligns with intuition. This indicates that the introduction of MBPS in MLBME has a similar effect to multi-epoch backpropagation in standard deep learning training, where the loss gradually decreases with more iterations.

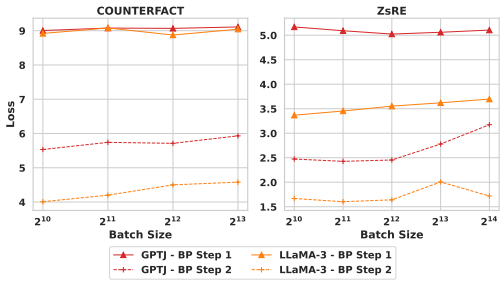

Figure 7: The loss at different BP steps varies with the number of batch edits.

Additionally, we visualize the gradients and pseudo-gradients at different BP steps during the validation of 2^{13} batch edits, as shown in Figure 8. It can be observed that there are clear differences between the gradients and pseudo-gradients at both Step 0 and Step 1, which are attributed to the transformation effect of the trained hypernetwork. On the other hand, the distributions of gradients and pseudo-gradients at Step 1 are more dispersed compared to those at Step 2. This indicates that MBPS reduce the spread of the gradient distribution, which is consistent with our previous analysis of the loss across different BP steps.

B General Capability Test on ZsRE

As mentioned in Section 3.1, the effectiveness of model editing methods can also be reflected in their ability to preserve the original capabilities of the model. Following the work of [10, 11], we evaluate the post-edit performance of LLMs using six natural language tasks from the General Language Understanding Evaluation (GLUE) benchmark [27]. The six tasks are: SST (The Stanford Sentiment Treebank) [31], MRPC (Microsoft Research Paraphrase Corpus) [32], MMLU (Massive Multi-task Language Understanding) [33], RTE (Recognizing Textual Entailment) [34], CoLA (Corpus of Linguistic Acceptability) [35], and NLI (Natural Language Inference) [36].

We perform 30 sequential edits with a batch size of 100 on both GPT-J and LLaMA-3. The performance of the LLMs before and after editing is evaluated every 5 edits. The results are shown in Figure 9. It can be observed that the overall performance of the edited LLMs is comparable across the three methods: SMEdit, RLEdit, and RLEdit^2. In some cases, SMEdit outperforms the other two methods—for example, on the CoLA and RTE tasks with GPT-J, and on the RTE, MMLU, and NLI tasks with LLaMA-3. However, on the SST task with GPT-J, SMEdit underperforms compared to

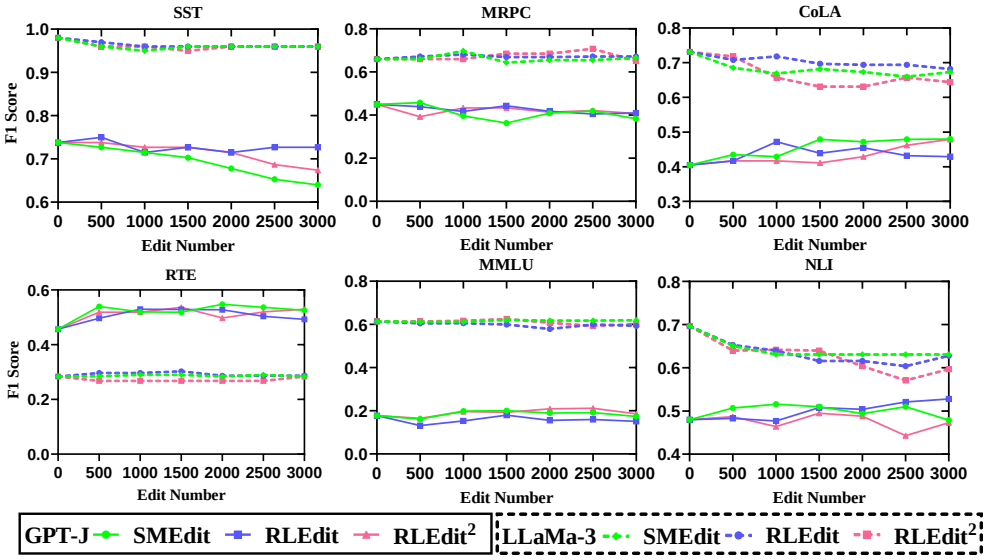

Figure 9: F1 scores of the post-edited Llama3 (8B) on six tasks.

the other two methods. Despite this, these results suggest that **the introduction of MBPS do not significantly impact the ability of editing methods to preserve the original capabilities of LLMs.**

C Experimental Details

Table 3: Hyperparameter Settings of Model Editing Methods

Hyperparameter	SMEdit	MALMEN	RLEdit	MALMEN^2	RLEdit^2
Rank of linear transformation in hyper-network	1024	1024	1024	1024	1024
Number of blocks in hyper-network	4	4	4	4	4
Initial learning rate	1e-6	1e-6	1e-6	1e-6	1e-6
Meta-learning rate	1e-5	1e-5	1e-5	1e-5	1e-5
Locality coefficient	-	0.6	0.6	0.6	0.6
Maximum meta gradient norm	1	1	1	1	1
Regularization coefficient	0.5	-	1e-4	-	1e-4
Discount factor	-	-	1	-	1
Backtracking depth	-	-	10	-	10
Memory backtracking decay factor	-	-	0.95	-	0.95
BP Steps	2	1	1	2	2

In this section, we provide the experimental details, including the experimental environment, hyper-parameter settings, datasets, and evaluation metrics. All experiments are conducted on a single A800 (80G) GPU and a server equipped with two Intel(R) Xeon(R) Platinum 8358 CPUs @ 2.60GHz.

C.1 Hyperparameter Setting

To ensure a fair comparison, the hyperparameter settings of **SMEdit**, **MALMEN**, and **RLEdit** are kept largely consistent. For the variant methods **MALMEN^2** and **RLEdit^2**, all hyperparameters remain the same as their original versions, except for the BP steps. We present the hyperparameters of each method in Table 3.

In our study, for **sequential editing**, all methods edit the following modules:

- **GPT-J:** `mlp.fc_in[10-26]`
- **LLaMA-3:** `mlp.gate_proj[11-15]` and `mlp.up_proj[18-24]`

For **batch editing**, all methods edit:

- **GPT-J:** `mlp.fc_out[22-27]`
- **LLaMA-3:** the same modules as in the sequential editing setting

C.2 Details of Dataset

ZsRE (Zero-shot Relation Extraction) [7] ZsRE is originally a relation extraction dataset. Zhu et al. [7] processes the training set from the KILT [37] version of ZsRE by evenly distributing the questions corresponding to each fact into the training and test sets, and filtering out samples that exceed the token limit. Each sample in this dataset consists of a question, a paraphrased version of the question, and a semantically inconsistent local question. The question and its paraphrase are used to evaluate the efficacy and generalization of the editing method, respectively, while the local question is used to assess specificity. The metrics are computed as follows:

- Efficacy: It evaluates the ability of edited model θ_{W_1} to correctly predict the exact editing instance (x, y):
$$\mathbb{E}[y = \arg\max_{\boldsymbol{y}} \mathbb{P}_{\theta_{W_1}}(\boldsymbol{y}|x)] \tag{8}$$
- Generalization: It evaluates the ability of edited model θ_{W_1} to correctly predict the semantically consistent instance (x^e, y^e) of the exact editing instance (i.e., paraphrased version):
$$\mathbb{E}[y^e = \arg\max_{\boldsymbol{y}} \mathbb{P}_{\theta_{W_1}}(\boldsymbol{y}|x^e)] \tag{9}$$
- Specificity: It evaluates the ability of edited model θ_{W_1} to correctly predict the unrelated instance (x^u, y^u):
$$\mathbb{E}[y^u = \arg\max_{\boldsymbol{y}} \mathbb{P}_{\theta_{W_1}}(\boldsymbol{y}|x^u)] \tag{10}$$

COUNTERFACT [8] The COUNTERFACT dataset originates from the PARAREL [38] dataset, with additional paraphrase prompts and neighborhood prompts added to evaluate generalization and specificity, respectively. Following previous work [11], we use efficacy, generalization, and specificity to evaluate the performance of editing methods on this dataset.

- Efficacy: It evaluates whether the edited model θ_{W_1} assigns a higher probability to the edited instance (x, y) compared to the old knowledge $(x, \boldsymbol{y})$:
$$\mathbb{E}[\mathbb{P}_{\theta_{W_1}}(y|x) > \mathbb{P}_{\theta_{W_1}}(\boldsymbol{y}|x)] \tag{11}$$
- Generalization: It evaluates whether the edited model θ_{W_1} assigns a higher probability to the semantically consistent instance (x^e, y^e) of the edited instance compared to the old knowledge $(x, \boldsymbol{y})$:
$$\mathbb{E}[\mathbb{P}_{\theta_{W_1}}(y^e|x^e) > \mathbb{P}_{\theta_{W_1}}(\boldsymbol{y}|x^e)] \tag{12}$$
- Specificity: It evaluates whether the edited model θ_{W_1} assigns a higher probability to the unrelated instance (x^u, y^u) instead of the edited instance:
$$\mathbb{E}[\mathbb{P}_{\theta_{W_1}}(y^u|x^u) > \mathbb{P}_{\theta_{W_1}}(y|x^u)] \tag{13}$$

D Limitations

Although this work demonstrates that MBPS can enhance the performance of MLBME in both batch and sequential editing, our evaluation is currently limited to simple fact-based knowledge datasets. Whether this improvement extends to more complex scenarios, such as reasoning-based knowledge and long-form knowledge, remains to be further explored. We leave this for future work.

E Impact Statements

Model editing is an emerging field that has gained traction following the advent of pretrained language models. It aims to efficiently correct errors, outdated information, toxic content, and sensitive knowledge within large language models using minimal resources. However, like any powerful technology, model editing is a double-edged sword. While our intention in developing model editing techniques is to enable more efficient knowledge updates in language models, these methods could also be misused by malicious actors to inject harmful information into models. We strongly oppose any malicious use of model editing technologies and urge the research community to develop methods that can defend against such misuse.