Wasserstein Distance Constraint and Parameter Sparsification for Batched and Iterative Knowledge Editing

Shanbao Qiao, Xuebing Liu, Seung-Hoon Na*

Center for Advanced Image and Information Technology,
Department of Computer Science and Artificial Intelligence,
Jeonbuk National University
{joe,liuxuebing,nash}@jbnu.ac.kr

Abstract

Model knowledge editing has become a widely researched topic because it enables the efficient and rapid injection of new knowledge into language models, as well as the correction of erroneous or outdated knowledge. Existing model knowledge editing methods are typically classified into two categories: single-instance sequential editing and massive one-time editing. However, the batched and iterative editing method is better aligned with model updating patterns in practical applications. In this work, we examine the performance of parameter-update-based models on a novel batched iterative editing benchmark. Our findings indicate that as the number of editing iterations increases, the accumulation of updated parameters leads to a greater shift in the model's parameter distribution, making it harder to maintain both the editing performance and the stability of the model. To address this degradation issue, we propose two methods: the *Wasserstein distance constraint* and the *parameter update sparsification*, where the Wasserstein distance constraint optimizes the transition of parameter distribution before and after editing, and the parameter update sparsification significantly reduces the number of updated parameters, thereby alleviating instability in the parameter distribution caused by their accumulation over iterations. Our methods are generally applicable to various parameter-update-based knowledge editing models. Experiments on the zsRE and CounterFact datasets demonstrate that our methods improve editing performance and enhance the stability of batched iterative editing in the later stages across different models.

Code — https://github.com/JoveReCode/WDSP.git

Introduction

Large language models (LLMs) have demonstrated strong capabilities across a wide range of natural language processing (NLP) tasks (Touvron et al. 2023; OpenAI 2023; Petroni et al. 2020). However, as world knowledge continuously evolves and changes, the knowledge stored in the model parameters can become outdated over time (Onoe et al. 2022; Dhingra et al. 2022; Liška et al. 2022). Additionally, the models may inherently contain erroneous knowledge (Zhao et al. 2023; Ji et al. 2023; Lazaridou et al. 2021; Agarwal and Nenkova 2022; Gallegos et al. 2023), making iterative updates to LLMs both necessary and crucial. Retraining or fine-tuning LLMs requires substantial computational resources and time. To achieve time-efficient model updates, the *knowledge editing* paradigm has emerged, gaining increasing research attention. Knowledge editing aims to timely inject new knowledge or correct erroneous knowledge by updating a small subset of model parameters or leveraging in-context learning (ICL) techniques (Brown et al. 2020; Dong et al. 2022). Developing efficient and stable knowledge editing methods is critical for the future maintenance of language models.

Existing knowledge editing methods can be broadly categorized into parameter updating methods, meta-learning-based methods, and ICL-based methods. Parameter updating approaches (Cao, Aziz, and Titov 2021; Meng et al. 2022a,b; Li et al. 2023) typically leverage mechanistic interpretability (MI) to identify specific layers in the model that store new knowledge and rewrite the relevant parameters. Meta-learning methods (Mitchell et al. 2022a; Tan, Zhang, and Fu 2024) involve training a hypernetwork to alter the model's output predictions, whereas ICL-based methods (Zheng et al. 2023; Zhong et al. 2023) temporarily adjust the model's output by appending the constructed prompts to the input query. Most existing work on editing focuses on single-instance sequential editing or massive one-time editing. However, in practice, *batched and iterative* editing is preferred in the maintenance of LLMs (e.g., the model's knowledge needs to be updated weekly or monthly as world knowledge changes rapidly). This primarily requires that editing methods are performed at the batch level. In this regard, ICL-based methods are less practical for batch execution, as they require lengthy prompts for each editing instance, which significantly reduces inference efficiency. Moreover, the effects of knowledge editing achieved through ICL are only temporary, lasting only within the current model run, and do not genuinely integrate the knowledge into the model. Meta-learning-based methods also train mainly for individual instances at a time, which limits their ability to achieve satisfactory performance in batched editing.

In contrast, parameter updating methods can perform massive knowledge editing in a single step, with the number of edits meeting the typical requirements for model updates

*Corresponding author

and maintenance. However, *can parameter updating methods achieve iterative and stable editing?* We found this to be a challenging issue for current popular parameter updating methods. We tested representative parameter updating methods, namely MEMIT(Meng et al. 2022b) and PMET(Li et al. 2023), on a benchmark involving batched and iterative editing. Our findings indicate that as the number of iterations increases, the model is prone to rapid deterioration. This is mainly due to the accumulation of parameters that need to be updated with each iteration, which leads to increasingly significant changes in the parameter distribution of the model, thus resulting in instability and collapse. To address the challenging task of batch and iterative knowledge editing, we propose methods based on *Wasserstein distance constraint* (Rubner, Tomasi, and Guibas 2000) and *parameter update sparsification*, aiming to mitigate the accumulation of adverse effects from parameter rewriting (that is, parameter updating) and to maintain the long-term stability of the model's parameter distribution. Wasserstein distance is a metric that measures the similarity between two probability distributions by calculating the minimal cost required to transform one into the other. We constrain the learning process of parameter updates by establishing a Wasserstein distance loss between the model's initial parameter distribution and the post-edited parameter distribution. This approach aims to minimize changes in the model's parameter distribution after the knowledge editing process, thereby maintaining stability. Additionally, in parameter updating methods, not all parameter updates contribute to editing performance. Parameter update sparsification prunes the parameters with small magnitudes and applies *random dropout* to make the updates sparser in each editing batch. We found that discarding approximately 50% of the parameters does not degrade the editing performance (and may even improve it), which reduces the accumulation of new parameters in each batch, allowing the model to remain effectively updateable after multiple iterative editing cycles. The contributions of our work are summarized as follows:

1. We examine the performance of popular parameter updating methods for knowledge editing methods on a batched iterative editing benchmark, revealing that these methods often lose editing capability or cause the model to rapidly collapse after multiple iterations. To address this, we design a loss function based on the Wasserstein distance to constrain distributional changes during the parameter update process, thereby maintaining the stability of the model's parameter distribution.
2. We further apply sparsification to the parameters that need updating, discarding redundant ones that do not contribute to editing performance. This significantly reduces the number of parameters to be updated in each iteration, thereby mitigating changes in the model's parameter distribution caused by the accumulation of updates. This approach can be used independently or in combination with the Wasserstein distance constraint, both of which help prevent the model from collapsing rapidly after multiple editing iterations, while also improving knowledge editing performance.

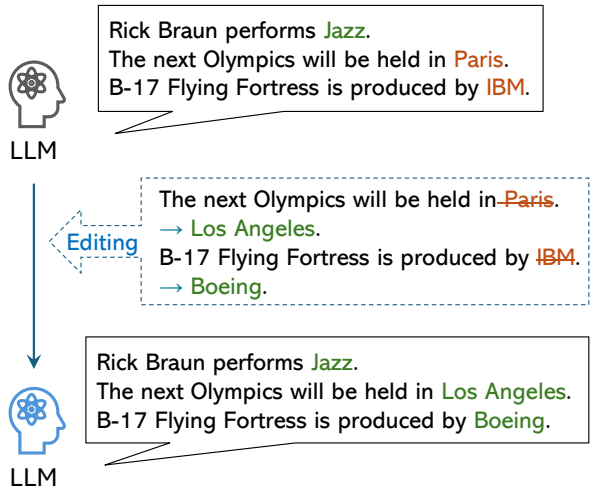

Figure 1: Illustration of knowledge editing: The language model stores a vast amount of knowledge, including outdated or erroneous information that needs to be edited. After applying the knowledge methods the model should correctly generate updated answers (e.g., for the query *"The next Olympics will be held in"*, the model should respond with *"Los Angeles"*) while preserving unedited knowledge (e.g., for the query *"Rick Braun performs"*, the model should still respond with *"Jazz"*).

3. Extensive experiments on the zsRE and CounterFact datasets demonstrate the effectiveness of our proposed methods.

Related Works

Knowledge editing methods can be categorized into several types from a technical perspective, including meta-learning, parameter updating[1], and in-context learning (ICL) methods. However, from an alternative perspective on their editing capacity, they can be classified into two main categories: single-instance editing and batched (or massive) editing. Single-instance editing is typically performed sequentially, and if model parameters are not restored, it can edit a few dozen pieces of knowledge at most. In contrast, batch or massive editing can handle hundreds to thousands of edits simultaneously, resulting in significant differences in their applicability.

Single Editing Methods MEND (Mitchell et al. 2022a) trains a hyper-network to convert the fine-tuning gradient into a simplified representation using low-rank decomposition, and then applies this updated gradient to the model. SERAC (Mitchell et al. 2022b) stores the edits in a separate memory and uses a retrieval-augmented counterfactual model to incorporate the edits to influence the model's predictions. ROME (Meng et al. 2022a) locates the model parameters corresponding to new factual knowledge and up-

[1]In this paper, parameter updating methods are often referred to as the update-based editing methods.

dates the key-value pairs in the MLP module with newly computed residual vectors. T-Patcher (Huang et al. 2023) adds a single neuron to the final layer of the feed-forward network (FNN) to handle a specific edit request.

GRACE (Hartvigsen et al. 2023) is a recent work towards lifelong knowledge editing, which employs a discrete codebook to store editing information and uses a deferral mechanism to determine whether to apply it during inference. This enables knowledge editing without modifying the model's internal parameters, thus avoiding potential model degradation. While the goal of this work aligns closely with ours, there are notable differences in the underlying concepts and applicable scenarios. GRACE is designed for instance-level sequential editing scenarios (i.e., editing one piece of knowledge at a time), whereas our approach aims to enable the sequential editing of batches of new knowledge. Furthermore, GRACE requires maintaining an external codebook, the size of which grows as the amount of edited knowledge increases. In contrast, our work focuses on achieving knowledge editing solely through the model's own parameters, ensuring stable performance in extensive knowledge editing scenarios without introducing any external components.

Batched/Massive Editing Methods To enable massive editing, MEMIT (Meng et al. 2022b) improves upon ROME by distributing the computed residual vectors across multiple model layers. MALMEN (Tan, Zhang, and Fu 2024) extends MEND to handle massive editing tasks by aggregating batched parameter updates into a single update, thereby achieving improved scalability. PMET (Li et al. 2023) builds on MEMIT by considering the knowledge-storing role of the multi-head self-attention (MHSA) layers, thus preventing the overestimation of the parameter updates required for the FFN layers.

In our tested batched iterative editing benchmark, since single-instance editing methods cannot handle such tasks due to their limited capacity, we focus primarily on researching and testing methods for massive editing.

Method

In this section, we describe the knowledge editing task and introduce our method for facilitating batched and iterative editing. An overview of our method is illustrated in Figure 2.

Task Definition

The objective of the knowledge editing task is to inject new factual knowledge into the model or modify outdated or incorrect knowledge, while preserving the existing knowledge. An example is shown in Figure 1. Suppose that $\mathcal{M}$ is an auto-regressive language model with parameters θ. Given factual knowledge represented by a tuple (s, r, o), where s, r, and o denote the *subject, relation* and *object* of the fact, respectively (e.g., s:"B-17 Flying Fortress", r: "is produced by", o: "IBM"), the language model $\mathcal{M}_\theta$ takes the prefix (s, r) as the input and predicts the answer o through the decoding process:

$$o = \arg\max_{x=(s,r)} P_{\mathcal{M}_\theta}(y|x) \tag{1}$$

where $P_{\mathcal{M}_\theta}(y|x)$ denotes the generative probability of y. In cases where the factual knowledge needs to be edited, given the prefix (s, r), the knowledge editing methods should steer the model's prediction towards the new answer o^* (e.g., o^*:"Boeing") by modifying the model parameters to θ^* or by using in-context learning prompts, i.e.:

$$o^* = \arg\max_{x=(s,r)} P_{\mathcal{M}_{\theta^*}}(y|x) \tag{2}$$

Single-instance knowledge editing methods (Mitchell et al. 2022a; Meng et al. 2022a; Huang et al. 2023) perform such editing process sequentially for different facts, with some requiring the model parameters to be restored to their initial state between each edit, which makes it challenging to handle large-scale editing scenarios.

For the batched or massive editing task, the editing requests are given as a set of new knowledge $\mathcal{E} = \{e_i\}_{i=1}^N$, where $e_i = (s_i, r_i, o_i^*)$. The editing methods compute the parameter updates (or changes) for all new knowledge in the editing set and apply them to the model simultaneously. In this work, we further investigate whether these methods can be performed iteratively without restoring the updated parameters in each iteration, thereby enabling batched and iterative editing that is suitable for practical model maintenance. To formally describe this editing process, consider a collection of editing request sets with batch size B denoted as $\{\mathcal{E}_j\}_{j=1}^T$, where $\mathcal{E}_j = \left\{e_i^j\right\}_{i=1}^B$, and T is the total number of iterations for the test. Given an editing method $\mathcal{G}$, the model parameters are updated as follows:

$$\mathcal{M}_{\theta_t^*} = \mathcal{G}(\mathcal{M}_{\theta_{t-1}^*}, \mathcal{E}_t), t \in \{1, 2, ..., T\} \tag{3}$$

where $\theta_0^* = \theta$ represents the initial model parameters. We evaluate the editing performance for each batch of edits.

Wasserstein Distance Constraint

Although parameter updating methods can achieve batched and permanent knowledge editing, they face challenges in iterative editing tasks. For simplicity, we describe the process of update-based knowledge editing methods (i.e., parameter updating methods) as follows: compute the residual vectors δ (parameter updates) needed for the model to predict the new facts o^*, and then add δ to the hidden states of the previous layer, which can be simply represented as $\theta^* = \theta + \delta$. For the overall model update, compute the update matrix Δ using the collection of residual vectors δ and the model's FFN keys, then add the update matrix Δ to the original model's FFN weights W, i.e. $W^* = W + \Delta$. In the iterative editing process, each update accumulates a new Δ on top of the previous iteration, causing the model parameter distribution to undergo increasing changes as the number of iterations grows. In the later stages of the iteration, this can rapidly deteriorate performance (as will be demonstrated by the experimental data in subsequent sections). To address this issue, we introduce methods based on Wasserstein distance (Rubner, Tomasi, and Guibas 2000) constraints and parameter sparsification, with the goal of maintaining the stability of the model's parameter distribution and preventing degradation.

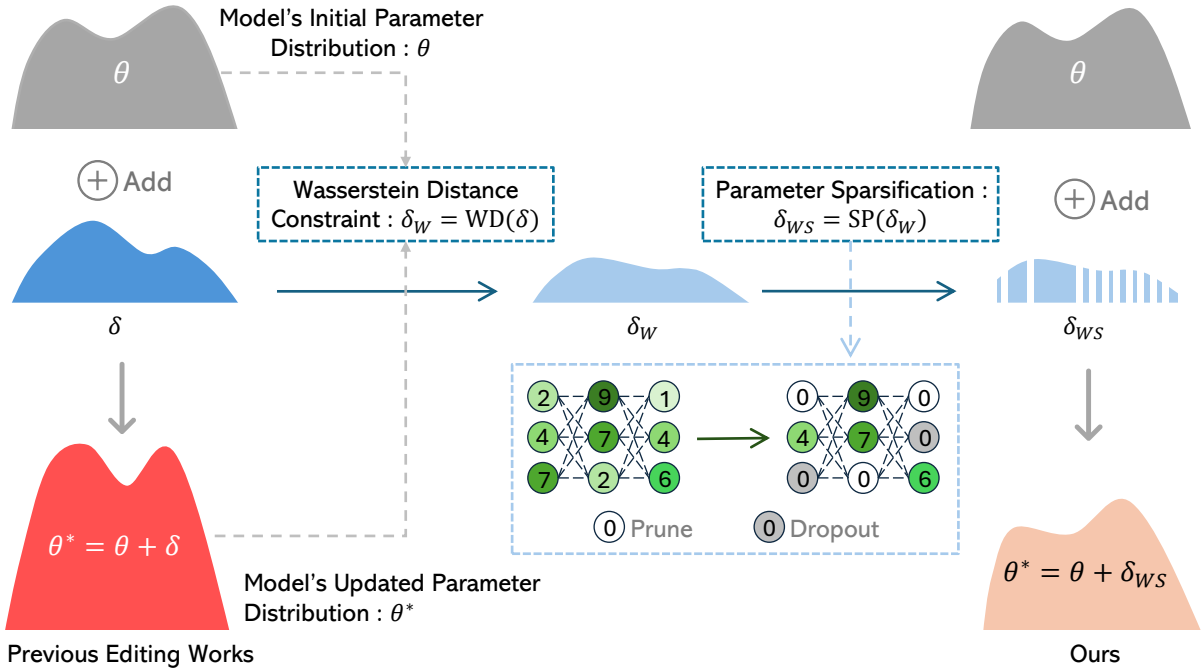

Figure 2: An overview of the process of our proposed method. Given a language model $\mathcal{M}$ with its initial parameters θ, parameter updating methods for knowledge editing compute residual values δ for the new requested editing knowledge and add them to the initial parameters θ to achieve the editing. This results in significant changes to the parameter distribution, and after multiple iterations, it can cause severe instability or even lead to model collapse. Our proposed method consists of two key processes. First, we apply a Wasserstein distance constraint ($\mathrm{WD}(\cdot)$) between the updated and initial parameter distributions to obtain smoothed residual values $\delta_W = \mathrm{WD}(\delta)$. Second, we apply parameter (update) sparsification, which performs pruning and dropout to sparsify the δ_W, resulting in $\delta_{WS} = \mathrm{SP}(\delta_W)$. The final δ_{WS} enables smoothed parameter updates for batched editing, which significantly helps maintain model stability during subsequent editing iterations.

Wasserstein distance is a metric that measures the discrepancy between two probability distributions. It quantifies the minimum cost required to transport one distribution into another, taking into account both the distances and the amount of mass being moved. Given two probability measures μ and ν, the Wasserstein p-distance[2] between them is defined as:

$$W_p(\mu,\nu) = \inf_{\gamma\in\Pi(\mu,\nu)} \mathbb{E}_{(x,y)\sim\gamma}\left[||x-y||_p^p\right] \tag{4}$$

where $\Pi(\mu,\nu)$ is the set of all joint distributions whose marginal distributions are μ and ν. In our task, we aim to constrain the Wasserstein distance between the initial model parameters θ and the updated model parameters θ^*. However, computing $W(\theta,\theta_t^*)$ is intractable due to the high dimensionality of the model parameters. We use the *Sinkhorn* algorithm (Sinkhorn and Knopp 1967; Luise et al. 2018) to compute the approximate solution, by introducing entropy regularization, where the computation of Wasserstein distance is transformed into the convex optimization problem. With the transport plan γ and the regularization term being differentiable, the gradient of the approximated Wasserstein distance can be efficiently computed by back-propagating through the iterative updates of the Sinkhorn algorithm. The Sinkhorn iteration can be expressed as follows:

$$\begin{aligned}\mathbf{u}^{(i+1)} &= \frac{\theta}{\mathbf{K}\mathbf{v}^{(\mathbf{i})}} \\ \mathbf{v}^{(i+1)} &= \frac{\theta^*}{\mathbf{K}^T\mathbf{u}^{(i+1)}}\end{aligned} \tag{5}$$

where $\mathbf{v}$ and $\mathbf{u}$ start as uniform distributions and iterate to convergence. The kernel matrix $\mathbf{K} = e^{\frac{-\mathbf{C}}{\epsilon}}$ is computed by the L_2 distance matrix $\mathbf{C}$, where $\mathbf{C_{i,j}} = ||\beta_i - \zeta_j||_2^2, (\beta \in \theta, \zeta \in \theta^*)$, ϵ is a hyper-parameter that controls the intensity of entropy regularization. After the Sinkhorn iteration converges, the solution $\mathbf{P}$ can be computed by:

$$\mathbf{P} = \mathrm{diag}(\mathbf{u})\mathbf{K}\mathrm{diag}(\mathbf{v}) \tag{6}$$

here $\mathbf{P}$ is the matrix of $\gamma(x,y)$, which specifies the optimal transport plan between two distributions, enabling the computation of the Wasserstein distance $W(\theta,\theta^*)$.

Given the computed Wasserstein distance, we use it as a loss function during the computing of update parameters δ (the residual vector, which can be represented as $\delta = \theta^* - \theta$), and it is defined as:

$$L_{WD} = W(\theta,\theta^*). \tag{7}$$

In the update-based knowledge editing methods such as MEMIT (Meng et al. 2022b) or PMET (Li et al. 2023), the

[2]We used the Wasserstein 2-distance in our experiments and denote it as $W(\mu,\nu)$ in the paper for simplicity.

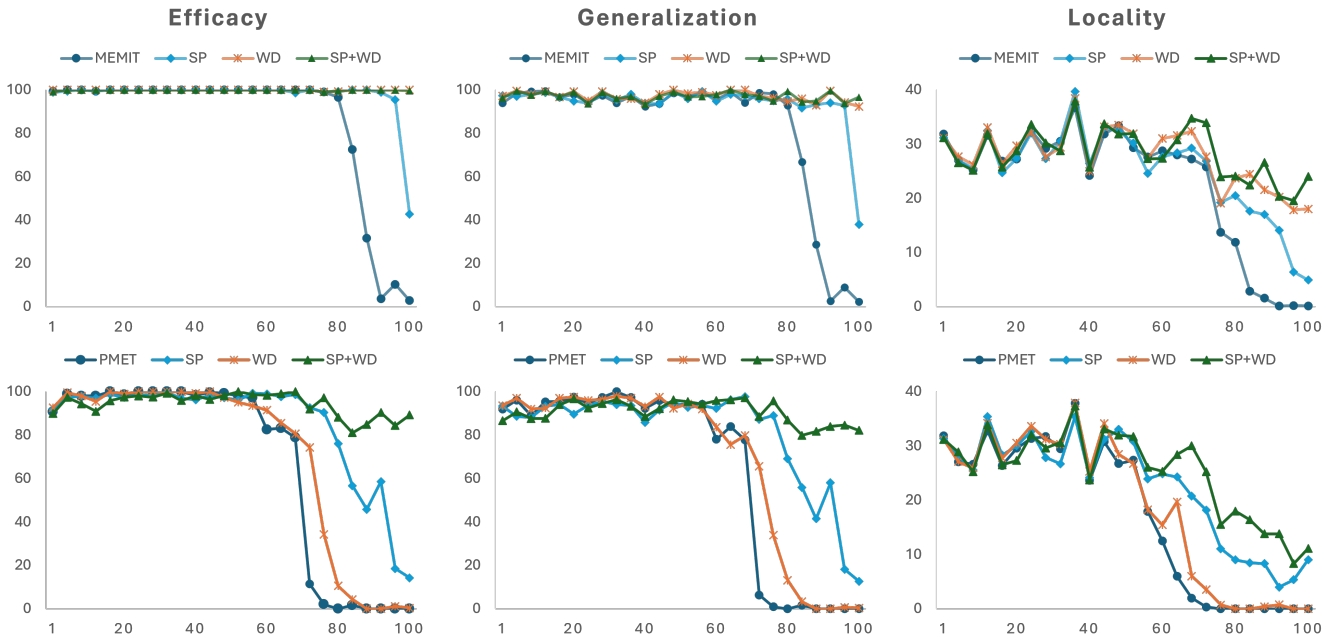

Figure 3: Detailed performance of Efficacy, Generalization and Locality on the zsRE dataset (batch size = 50). The vertical axis represents the metric scores for the current editing iteration, while the horizontal axis represents the number of iterations. **SP** denotes the parameter sparsification, **WD** denotes the Wasserstein distance constraint, and **WD+SP** denotes the combined method.

residual vector δ for each editing request is computed by the negative log-likelihood loss[3]:

$$L_{NLL} = -\log \mathcal{P}_{\mathcal{M}(\theta+\delta)}[o^*|(s,r)] \tag{8}$$

Along with the Wasserstein distance loss, our loss function for computing the residual vector δ is defined as follows:

$$L = L_{NLL} + \lambda(L_{WD}) \tag{9}$$

Unlike conventional hyper-parameters, λ here is a dynamic scaling function. The value of the Wasserstein distance varies across different samples, with some results being quite large, which could lead to an overemphasis of the Wasserstein distance constraint. Additionally, as iterative editing progresses and the parameter distribution changes, the Wasserstein distance between the updated and the original parameter distributions may also increase. To prevent the Wasserstein distance from becoming excessively large and dominating the overall loss function, we introduce a dynamic scaling factor λ to adjust the contribution of the Wasserstein loss. Since the value of L_{NLL} is relatively stable, we dynamically normalize the L_{WD} value to match the scale of L_{NLL}.

Parameter Update Sparsification

Given the loss function in equation (9), for a batched editing with batch size B, the residual vector δ for each editing request is computed and collected as $R = \{\delta_i\}_{i=1}^{B}$, and R is then used to compute the update matrix Δ, which is applied to the model layers to be updated:

$$\Delta \leftarrow RK^T(C + KK^T)^{-1} \tag{10}$$

where K denotes the FFN keys of the model layer and C is a constant proportional to the uncentered covariance of the pre-existing keys (pre-computed from a sample of Wikipedia text, Meng et al. (2022a)). With the overall model update defined as $W^* = W + \Delta$, the sparsification of the update matrix Δ has been used in recent work (Gu et al. 2024) to preserve the general capabilities of the model after editing. We consider whether the sparsification of Δ can be used to preserve the model's capabilities after multiple iterations of editing. However, directly sparsifying the update matrix Δ incurs significant computational costs, as it shares the large dimensions as the FFN layer, $M_{in} \times M_{out}$. In batched iterative editing, assuming we need to perform T rounds of editing, with l being the number of layers to be edited and Q representing the computational complexity of sparsification, the total complexity required for sparsifying Δ would be $\mathcal{O}(M_{in} \times M_{out} \times l \times T \times Q)$. To reduce the computational complexity to an acceptable level, we apply the *parameter update sparsification* which sparsifies the residual vectors δ in R and allows this sparsity to propagate into the update matrix Δ. This reduces the computational complexity to $\mathcal{O}(M_{in} \times B \times l \times T \times Q)$, where B is the editing batch size, which is significantly smaller than M_{out}. This also enables our parameter sparsification method to scale more efficiently to larger models, as the computational cost primarily depends on M_{in}, making it more efficient than ex-

[3]Here, we simplify the symbolic representation for uniform description. In practice, the residual vector δ_i is computed individually for each editing request $e_i = (s_i, r_i, o_i^*)$.

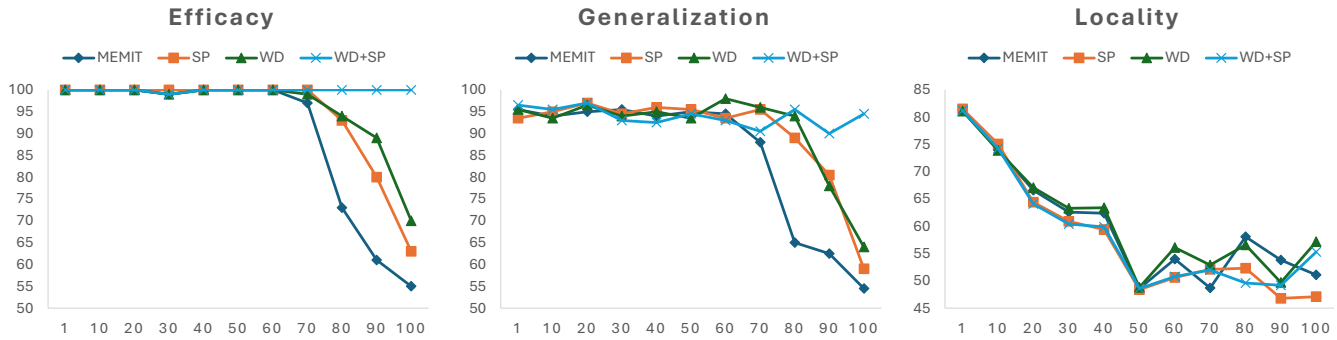

Figure 4: Detailed performance of Efficacy, Generalization and Locality on the CounterFact dataset (batch size = 100). The vertical axis represents the metric scores, while the horizontal axis represents the number of iterations. **SP** denotes the parameter sparsification, **WD** denotes the Wasserstein distance constraint, and **WD+SP** denotes their combination.

Method	Number of Batched Iteration											Avg
	@1	@12	@24	@36	@48	@60	@72	@80	@88	@96	@100	
MEMIT	**57.61**	58.02	58.40	63.11	59.91	54.32	50.97	28.53	4.29	0.66	0.45	39.66
+WD	57.33	**59.65**	58.17	64.63	**60.21**	**57.18**	53.17	47.80	44.71	39.18	39.35	52.85
+SP	57.24	58.92	57.87	**66.11**	58.97	52.95	52.07	43.30	37.79	16.93	11.88	46.73
+WD+SP	57.15	58.36	**59.52**	64.29	58.28	52.87	**60.29**	**48.71**	**51.69**	**41.81**	**48.35**	**54.67**
PMET	**56.31**	58.58	57.24	**64.26**	51.66	28.63	0.92	0.00	0.00	0.00	0.00	28.87
+WD	56.06	59.38	**59.86**	**64.26**	53.32	34.25	9.69	0.00	0.00	0.00	0.00	30.62
+SP	56.08	**60.91**	58.40	60.91	**58.79**	49.02	38.86	21.65	18.07	10.16	11.56	40.40
+WD+SP	54.83	57.79	57.44	62.62	57.86	**49.91**	**48.56**	**38.25**	**31.11**	**20.85**	**26.48**	**45.98**

Table 1: The overall editing **Score** on the zsRE dataset (batch size = 50). **@k** denotes the k-th iteration of batched editing. **+WD** denotes the Wasserstein distance constraint, **+SP** denotes the parameter sparsification, and **+WD+SP** denotes the combined method.

Method	Avg.Eff	Avg.Gen	Acg.Loc
MEMIT	76.41	73.69	21.33
+WD	**99.94**	96.05	28.04
+SP	94.18	90.73	24.67
+WD+SP	99.86	**96.85**	**29.29**
PMET	52.91	50.72	15.77
+WD	59.25	57.73	17.87
+SP	74.08	71.83	23.27
+WD+SP	**91.84**	**88.74**	**24.39**

Table 2: Average metric scores on the zsRE dataset (batch size = 50). **Avg.Eff**, **Avg.Gen**, and **Avg.Loc** denotes the average Efficacy, Generalization and Locality across all batched editing iterations, respectively.

isting methods that rely on both M_{in} and M_{out}.

For the details of sparsification, we use two operations: *pruning* and *dropout*. The pruning operation retains the top-k parameters with the largest magnitudes in δ, discarding the smaller ones, as they contribute less to the editing performance. We control the degree of sparsification by setting a pruning ratio $pr(0 < pr < 1)$, and define a mask matrix $\mathbf{Pr}$ as follows:

$$\mathbf{Pr}_{i,j} = \begin{cases} 1, & \text{if } \delta_{i,j} \text{ in top-pr}(\delta) \\ 0, & \text{else} \end{cases} \tag{11}$$

therefore the pruned residual vector is given by: $\delta \leftarrow \mathbf{Pr} \odot \delta$. Additionally, we apply dropout with a rate of dr to the pruned δ, which can be expressed as:

$$\delta \leftarrow \frac{(\mathbf{1} - \mathbf{Dr}) \odot \delta}{1 - dr} \tag{12}$$

where $\mathbf{Dr} \sim \text{Bern}(dr)$ is the dropout mask matrix.

After applying the Wasserstein distance constraint and parameter sparsification, the updated parameter δ exhibits a relatively smooth and sparse distribution. This ensures that the parameter updates from knowledge editing do not cause drastic changes to the model's parameter distribution, thereby maintaining stability even after multiple iterations.

Experiments

Datasets and Baselines

We evaluate our method on two widely used knowledge editing datasets, **zsRE**(Levy et al. 2017) and **CounterFact**(Meng et al. 2022a). In the zsRE dataset, each sample consists of a factual knowledge statement (editing request), its paraphrase, and an irrelevant natural question. In

Method	Number of Batched Iteration											Avg
	@1	@8	@16	@20	@24	@28	@32	@36	@42	@46	@50	
MEMIT	91.78	89.73	88.29	86.57	84.15	82.56	81.37	81.49	81.01	82.53	76.43	84.17
+WD	**92.25**	89.64	88.31	85.63	83.71	81.95	80.41	82.05	80.42	**83.25**	**77.73**	84.12
+SP	91.12	**90.65**	**89.26**	85.82	**85.09**	81.16	**81.47**	81.13	80.41	81.50	74.47	83.83
+WD+SP	91.56	89.82	88.68	**87.06**	84.07	**84.66**	80.53	**82.44**	**81.38**	82.09	76.12	**84.40**
PMET	92.74	**90.93**	89.15	84.03	81.49	82.32	79.03	78.73	75.44	76.35	66.97	81.56
+WD	92.42	89.96	**89.91**	85.63	83.09	**83.90**	79.56	78.77	74.43	77.37	68.18	82.11
+SP	**92.80**	90.75	89.63	**86.14**	83.51	83.14	79.89	79.24	76.03	75.46	69.65	82.39
+WD+SP	92.76	87.97	88.68	85.83	**84.04**	82.91	**80.90**	**79.38**	**77.26**	**82.19**	**75.90**	**83.44**

Table 3: Overall editing **Score** on the CounterFact dataset (batch size = 50). **@k** denotes the k-th iteration of batched editing. **+WD**, **+SP**, and **+WD+SP** denote the Wasserstein distance constraint, the parameter sparsification, and the combined method, respectively.

Method	Avg.Eff	Avg.Gen	Acg.Loc
MEMIT	**100**	95.55	66.42
+WD	**100**	94.55	**66.85**
+SP	99.82	94.18	66.53
+WD+SP	**100**	**95.91**	66.65
PMET	**100**	**97.00**	61.84
+WD	99.82	95.18	63.64
+SP	99.63	**97.00**	63.22
+WD+SP	99.27	93.64	**66.65**

Table 4: Average metric scores on the CounterFact dataset (batch size = 50). **Avg.Eff**, **Avg.Gen**, and **Avg.Loc** denotes the average Efficacy, Generalization and Locality across all batched editing iterations, respectively.

the CounterFact dataset, each sample consists of a factual knowledge statement, two paraphrased sentences, and ten neighborhood questions.

Given that our proposed method is a general approach that can be applied to any update-based editing method, we selected the representative parameter updating methods for massive editing tasks, MEMIT (Meng et al. 2022b) and PMET (Li et al. 2023), on the GPT-J 6B (Wang and Komatsuzaki 2021) model as experimental baselines. We evaluated the impact of the Wasserstein distance constraint, parameter sparsification, and their combination on the performance of these baselines in the batched iterative knowledge editing task.

Metrics and Settings

We use the standard evaluation metrics for knowledge editing, which are defined as follows:

- **Efficacy** measures the editing accuracy; it corresponds to the factual knowledge statement in the dataset being successfully edited.
- **Generalization** evaluates whether the edit can be extended to paraphrased or contextually relevant sentences included in the data set.
- **Locality** refers to the preservation of original knowledge that is not related to editing requests, ensuring it remains intact. This is evaluated using irrelevant natural questions or neighborhood questions in the dataset.

Formally, the metrics for **Efficacy** and **Generalization** are defined as follows:

$$\mathbb{E}[\mathcal{P}_{\mathcal{M}(\theta^*)}(o^*|(s,r)) > \mathcal{P}_{\mathcal{M}(\theta^*)}(o|(s,r))], \quad (13)$$

and the metric on **Locality** is defined as follows:

$$\mathbb{E}[\mathcal{P}_{\mathcal{M}(\theta^*)}(o^*|(s,r)) < \mathcal{P}_{\mathcal{M}(\theta^*)}(o|(s,r))]. \quad (14)$$

On the batched iterative editing benchmark, we evaluate the knowledge editing methods based on these three metrics in each batch of edits and calculate their harmonic mean to obtain the overall editing **Score**. On the zsRE dataset, we set the batch size for iterative editing to 50 and conducted 100 iterations, resulting in a total of 5,000 knowledge edits. On the CounterFact dataset, we evaluated the setting with a batch size of 50 over 50 iterations, resulting in a total of 2,500 edits. We then extended this to a more challenging setting with a batch size of 100 over 100 iterations, totaling 10,000 edits. For hyper-parameters, we set the prune rate pr to 0.7 and the dropout rate dr to 0.3. All of our experiments were conducted on NVIDIA RTX A6000 48G GPUs.

Results

We present all our experimental results as ablation studies to clearly highlight the contributions of the proposed Wasserstein distance constraint and parameter sparsification to editing performance.

Results on zsRE

For the zsRE dataset, we evaluated the performance of MEMIT and PMET in iterative editing with a batch size of 50 over 100 iterations. As observed in Table 1, the editing performance gradually declines with successive iterations, indicating that the accumulation of updated parameters makes the model increasingly unstable, thereby complicating future edits. Between the 70-80th iterations, the model begins to deteriorate rapidly, eventually collapsing and completely losing its editing capability in subsequent iterations.

After incorporating the Wasserstein distance constraint, the MEMIT-based model was able to retain most of its editing capabilities, even during the later stages of iteration. The

Method	Number of Batched Iteration											Avg
	@1	@10	@20	@30	@40	@50	@60	@70	@80	@90	@100	
MEMIT	91.58	87.84	84.04	82.08	81.82	**72.98**	76.72	71.07	64.8	58.84	53.47	75.02
+WD	91.45	87.65	**85.06**	**82.10**	**82.64**	72.83	**78.88**	**76.10**	**77.03**	67.90	63.30	78.63
+SP	91.00	**88.64**	83.71	81.07	80.53	72.93	74.15	75.63	72.97	64.80	55.50	76.44
+WD+SP	**91.80**	88.42	83.54	80.19	79.99	72.89	74.19	74.48	73.83	**72.40**	**77.59**	**79.03**

Table 5: The overall editing **Score** on the CounterFact dataset (batch size = 100). **@k** denotes the k-th iteration of batched editing. **+WD**, **+SP**, and **+WD+SP** denote the Wasserstein distance constraint, the parameter sparsification, and the combined method, respectively.

PMET-based model deteriorated more slowly than the baseline during the intermediate stages but still collapsed in the later stages. While solely applying parameter sparsification to the baselines provides some improvement, the model still lost most of its capability in the later stages. The combined method further improved the model's stability in later iterations and achieved the best overall performance. Figure 3 presents the details of the Efficacy, Generalization, and Locality metrics, clearly demonstrating the effectiveness of our method.

Table 2 additionally presents the average Efficacy, Generalization, and Locality scores across all iterations and settings. The baseline exhibited significantly lower average scores, mainly due to model collapse in the later stages. All of our methods demonstrate substantially higher average performance compared to the baseline, with the combined approach achieving the best results.

Results on CounterFact

For the CounterFact dataset, we first evaluated MEMIT and PMET with a batch size of 50 over 50 iterations. As shown in Table 3, all methods show a decreasing trend in editing performance as the number of iterations increases, while the model did not experience a rapid collapse within 50 iterations. Our method shows noticeable improvement over MEMIT. Employing the Wasserstein distance constraint, parameter sparsification independently, or their combination enhances performance in specific iterations, with the combined approach achieving the highest overall average performance overall. For PMET, the combined approach consistently yields the best performance during the later stages of iteration, and its effectiveness becomes more evident as the number of iterations increases. Using WD or SP independently also improves average performance compared to the baseline, and the combined approach achieves the highest average performance.

It is also observed that during the early stages of iteration (e.g., before the 28th iteration), our independent methods outperform the combined approach in certain iterations under both baselines. This is primarily because the model maintains relatively stable performance in the early stages due to minor parameter shifts, thus making the sole application of either the Wasserstein distance constraint or parameter sparsification sufficient to effectively ensure stability. Table 4 further presents the average performance of the Efficacy, Generalization, and Locality metrics. We found that all our approaches improve Locality. For PMET, the combined method sacrifices some Generalization but achieves a more significant improvement in Locality. This trade-off between Generalization and Locality remains a significant challenge in knowledge editing tasks. Nevertheless, these results indicate that our method helps maintain model stability, as a higher Locality implies better preservation of knowledge unrelated to the edits.

To further assess the effectiveness of our method, we conducted tests in a more challenging setting. Table 5 presents the results obtained after 100 iterations with a batch size of 100. Our method clearly demonstrates its effectiveness in maintaining stability during the later stages of batched iterative editing. While applying the Wasserstein distance constraint or parameter sparsification individually offers considerable improvements, the combined method yields the best average results, consistent with the findings on the zsRE dataset. Figure 4 presents the details on the Efficacy, Generalization, and Locality metrics. Our method ensures that both Efficacy and Generalization remain stable in the later stages of iteration, whereas Locality shows no significant differences across all methods.

Conclusion

In this work, we addressed the challenging task of batched iterative knowledge editing and demonstrated the instability of parameter updating methods (capable of batch editing) in the later stages of iterative editing. To prevent the rapid collapse of the model during later iterations, we proposed the use of the Wasserstein distance constraint and parameter sparsification to mitigate changes in the model's parameter distribution after multiple rounds of batched iterative editing, thereby preserving stability. These two techniques can be applied independently or in combination, and notably, the combined approach yielded the best results in most experiments.

For future work, we will focus on achieving lifelong batched iterative editing, aiming to completely eliminate the side effects of each batch iteration, thereby fully mitigating the distribution shifts and the severe instability caused by the accumulation of parameter updates during the iterative process. Furthermore, exploring improved methods that can support larger batch sizes during iterative editing is another important direction for future research. This will provide valuable insights and contributions to the practical updating and maintenance of language models.

Acknowledgments

This work was supported by Institute of Information & Communications Technology Planning & Evaluation(IITP) grant funded by the Korea government(MSIT) (RS-2023-00216011, Development of artificial complex intelligence for conceptually understanding and inferring like human). Shanbao Qiao and Xuebing Liu were also supported by the China Scholarship Council (CSC).

References

Agarwal, O.; and Nenkova, A. 2022. Temporal Effects on Pre-trained Models for Language Processing Tasks. *Transactions of the Association for Computational Linguistics*, 10: 904–921.

Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; Agarwal, S.; Herbert-Voss, A.; Krueger, G.; Henighan, T.; Child, R.; Ramesh, A.; Ziegler, D.; Wu, J.; Winter, C.; Hesse, C.; Chen, M.; Sigler, E.; Litwin, M.; Gray, S.; Chess, B.; Clark, J.; Berner, C.; McCandlish, S.; Radford, A.; Sutskever, I.; and Amodei, D. 2020. Language Models are Few-Shot Learners. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 1877–1901. Curran Associates, Inc.

Cao, N. D.; Aziz, W.; and Titov, I. 2021. Editing Factual Knowledge in Language Models.

Dhingra, B.; Cole, J. R.; Eisenschlos, J. M.; Gillick, D.; Eisenstein, J.; and Cohen, W. W. 2022. Time-Aware Language Models as Temporal Knowledge Bases. *Transactions of the Association for Computational Linguistics*, 10: 257–273.

Dong, Q.; Li, L.; Dai, D.; Zheng, C.; Wu, Z.; Chang, B.; Sun, X.; Xu, J.; and Sui, Z. 2022. A survey for in-context learning. *arXiv preprint arXiv:2301.00234*.

Gallegos, I. O.; Rossi, R. A.; Barrow, J.; Tanjim, M. M.; Kim, S.; Dernoncourt, F.; Yu, T.; Zhang, R.; and Ahmed, N. K. 2023. Bias and Fairness in Large Language Models: A Survey. arXiv:2309.00770.

Gu, J.; Xu, H.; Ma, J.; Lu, P.; Ling, Z.; Chang, K.; and Peng, N. 2024. Model Editing Can Hurt General Abilities of Large Language Models. *CoRR*.

Hartvigsen, T.; Sankaranarayanan, S.; Palangi, H.; Kim, Y.; and Ghassemi, M. 2023. Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors. In *Advances in Neural Information Processing Systems*.

Huang, Z.; Shen, Y.; Zhang, X.; Zhou, J.; Rong, W.; and Xiong, Z. 2023. Transformer-Patcher: One Mistake worth One Neuron. *arXiv preprint arXiv:2301.09785*.

Ji, Z.; Lee, N.; Frieske, R.; Yu, T.; Su, D.; Xu, Y.; Ishii, E.; Bang, Y. J.; Madotto, A.; and Fung, P. 2023. Survey of Hallucination in Natural Language Generation. *ACM Comput. Surv.*, 55(12).

Lazaridou, A.; Kuncoro, A.; Gribovskaya, E.; Agrawal, D.; Liska, A.; Terzi, T.; Gimenez, M.; de Masson d'Autume, C.; Kocisky, T.; Ruder, S.; et al. 2021. Mind the gap: Assessing temporal generalization in neural language models. *Advances in Neural Information Processing Systems*, 34: 29348–29363.

Levy, O.; Seo, M.; Choi, E.; and Zettlemoyer, L. 2017. Zero-Shot Relation Extraction via Reading Comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, 333–342.

Li, X.; Li, S.; Song, S.; Yang, J.; Ma, J.; and Yu, J. 2023. PMET: Precise Model Editing in a Transformer. *arXiv preprint arXiv:2308.08742*.

Liška, A.; Kočiský, T.; Gribovskaya, E.; Terzi, T.; Sezener, E.; Agrawal, D.; de Masson d'Autume, C.; Scholtes, T.; Zaheer, M.; Young, S.; Austin, E. G.-M. S.; Blunsom, P.; and Lazaridou, A. 2022. StreamingQA: A Benchmark for Adaptation to New Knowledge over Time in Question Answering Models. *arXiv preprint arXiv:2205.11388*.

Luise, G.; Rudi, A.; Pontil, M.; and Ciliberto, C. 2018. Differential properties of sinkhorn approximation for learning with wasserstein distance. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, 5864–5874. Red Hook, NY, USA: Curran Associates Inc.

Meng, K.; Bau, D.; Andonian, A.; and Belinkov, Y. 2022a. Locating and Editing Factual Associations in GPT. *Advances in Neural Information Processing Systems*, 35.

Meng, K.; Sen Sharma, A.; Andonian, A.; Belinkov, Y.; and Bau, D. 2022b. Mass Editing Memory in a Transformer. *arXiv preprint arXiv:2210.07229*.

Mitchell, E.; Lin, C.; Bosselut, A.; Finn, C.; and Manning, C. D. 2022a. Fast Model Editing at Scale. In *International Conference on Learning Representations*.

Mitchell, E.; Lin, C.; Bosselut, A.; Manning, C. D.; and Finn, C. 2022b. Memory-Based Model Editing at Scale. In Chaudhuri, K.; Jegelka, S.; Song, L.; Szepesvari, C.; Niu, G.; and Sabato, S., eds., *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, 15817–15831. PMLR.

Onoe, Y.; Zhang, M.; Choi, E.; and Durrett, G. 2022. Entity Cloze By Date: What LMs Know About Unseen Entities. In Carpuat, M.; de Marneffe, M.-C.; and Meza Ruiz, I. V., eds., *Findings of the Association for Computational Linguistics: NAACL 2022*, 693–702. Seattle, United States: Association for Computational Linguistics.

OpenAI. 2023. GPT-4 Technical Report. *ArXiv*, abs/2303.08774.

Petroni, F.; Lewis, P.; Piktus, A.; Rocktäschel, T.; Wu, Y.; Miller, A. H.; and Riedel, S. 2020. How context affects language models' factual predictions. *arXiv preprint arXiv:2005.04611*.

Rubner, Y.; Tomasi, C.; and Guibas, L. J. 2000. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40: 99–121.

Sinkhorn, R.; and Knopp, P. 1967. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2): 343–348.

Tan, C.; Zhang, G.; and Fu, J. 2024. Massive Editing for Large Language Model via Meta Learning. In *The Twelfth International Conference on Learning Representations*.

Touvron, H.; Lavril, T.; Izacard, G.; Martinet, X.; Lachaux, M.-A.; Lacroix, T.; Rozière, B.; Goyal, N.; Hambro, E.; Azhar, F.; et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Wang, B.; and Komatsuzaki, A. 2021. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax.

Zhao, W. X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. 2023. A survey of large language models. *arXiv preprint arXiv:2303.18223*.

Zheng, C.; Li, L.; Dong, Q.; Fan, Y.; Wu, Z.; Xu, J.; and Chang, B. 2023. Can We Edit Factual Knowledge by In-Context Learning? *arXiv preprint arXiv:2305.12740*.

Zhong, Z.; Wu, Z.; Manning, C. D.; Potts, C.; and Chen, D. 2023. MQuAKE: Assessing Knowledge Editing in Language Models via Multi-Hop Questions. *arXiv preprint arXiv:2305.14795*.