Consecutive Batch Model Editing with HooK Layers

Anonymous ACL submission

Abstract

As the typical retraining paradigm is unacceptably time- and resource-consuming, researchers are turning to model editing to find an effective way that supports both consecutive and batch scenarios to edit the model behavior directly. Despite all these practical expectations, existing model editing methods fail to realize all of them. Furthermore, the memory demands for such sequential model editing approaches tend to be prohibitive, frequently necessitating an external memory that grows incrementally over time. To cope with these challenges, we propose CoachHooK, a model editing method that simultaneously supports sequential and batch editing. CoachHooK is memory-friendly as it only needs a small amount of it to store several hook layers whose size remains unchanged over time. Experimental results demonstrate the superiority of our method over other batch-supportive model editing methods under both single-round and consecutive batch editing scenarios. Extensive analyses of CoachHooK have been conducted to verify the stability of our method over 1) the number of consecutive steps and 2) the number of editing instances. Our code will be released via `https://github.com/anonymous`.

1 Introduction

Large Language Models (LLMs) (Chung et al., 2022; OpenAI, 2023; Black et al., 2022; Touvron et al., 2023) have been demonstrated to be capable of recalling factual knowledge about the real world (Brown et al., 2020; Petroni et al., 2020). Nevertheless, researchers also reveal that LLMs often fail to recall the most up-to-date knowledge or information and some specialized knowledge if they are not periodically updated (Liska et al., 2022; Agarwal and Nenkova, 2022; Lazaridou et al., 2021). Despite the fact that fresh and customizable knowledge is highly desired in many areas, such as text generation, question-answering, reasoning, etc., updating the model via retraining is both time- and resource-consuming. Additionally, researchers have uncovered that well-trained LLMs do make mistakes. One popular sort of mistake is called hallucination (Tonmoy et al., 2024), which means that LLMs generate text based on "hallucinated" fake knowledge. Although many researchers have tried to mitigate this issue (Qiu et al., 2023; Mündler et al., 2023; Kang et al., 2023; Varshney et al., 2023), the strategy to fix this bug remains unclear. Therefore, researchers have started to seek an efficient approach that could edit LLMs in a customizable, cost-effective way.

To this end, recent years have witnessed many efforts in investigating the model-editing techniques to bypass the retraining paradigm and edit the LLMs directly (Meng et al., 2022; Hartvigsen et al., 2022; Li et al., 2023; Mitchell et al., 2022a,b). Accordingly, several new datasets (*e.g.*, ZsRE (Levy et al., 2017) and COUNTERFACT (Meng et al., 2023)) and corresponding metrics (*e.g.*, reliability, generality, locality, portability (Yao et al., 2023)) are proposed to facilitate the development in this field. However, these methods either require extra expensive training of a meta-network (Mitchell et al., 2022a; De Cao et al., 2021), or a classifier (Mitchell et al., 2022b), which causes time and resources overhead, or demands an external memory of explicit edit instances for reference (Mitchell et al., 2022b; Hartvigsen et al., 2022), which inevitably escalates the memory requirement. Further, most of existing methods are only evaluated on single-round editing, where the model is rolled back to the initial state after each edit step. This deviates from the application scenario in reality since most users anticipate an editing approach that allows sequential and batch editing.

In light of these issues, we propose a novel method, named **CoachHooK**, which performs **Co**nsecutive **Batch** Model Editing with **HooK** layers. Specifically, CoachHooK supports consecutive batch editing and utilizes the hook layers to separate weight change from the original model weight. CoachHook does not need any training on parameters or large explicit external memory that stores editing instances. It only needs a reasonable amount of memory to collect the optimized weight in the hook layer. To achieve this, we propose a new transformer memory updating mechanism that supports consecutive batch editing settings and design a simple yet effective local editing scope identification technique used in the hook layer that can accurately detect the inputs in the local editing scope. We demonstrate the effectiveness of our method via extensive experiments on ZsRE and COUNTERFACT datasets using two popular autoregressive language models, GPT2-XL and GPT-J (6B). Both the single-round batch settings and consecutive batch settings are included, with the total number of editing instances ranging from 1k to 10k. An analysis of the editing scope identification has also been conducted to validate the method. Beyond all these, we implement comprehensive ablation studies to verify the validity of each component and discuss the optimal hyperparameter settings in the method.

2 Preliminaries of Model Editing

As defined by Yao et al. (2023), the task of model editing is to efficiently modify an initial base model f_θ into an edited model $f_{\theta'}$ whose responses to a particular set of input instances $\mathcal{X}_t$ are adjusted as desired without affecting the responses of the model to other instances. The intended edit descriptor is denoted as (x_t, y_t), where $x_t \in \mathcal{X}_t$ and $f_\theta(x_t) \neq y_t$. The post-edit model $f_{\theta'}$ is supposed to produce the expected output to an intended edit instance x_t, while preserving the original output to other instances:

$$f_{\theta'}(x) = \begin{cases} y_t & \text{if } x \in \mathcal{X}_t \\ f_\theta(x) & \text{if } x \notin \mathcal{X}_t \end{cases} \qquad (1)$$

In particular, there are three standard criteria for model editing, namely Reliability, Generality, and Locality (Yao et al., 2023; Mitchell et al., 2022a,b). Suppose the prediction of the original model to the prompt "*What is the native language of Joe Biden?*" is "*French*", and the expected post-edit model prediction is "*English*". To verify the Reliability, we use the same original prompt as input and then assess whether the post-edit model predicts "*English*" as desired. For Generality, a rephrased prompt "*The mother tongue of Joe Biden is*" could be inputted into the edited model to assess whether the output of the model remains as "*English*". Locality suggests that the model output of an irrelevant prompt like "*What is the native language of Donald Trump?*" should remain unaffected, which means that the post-edit model should output whatever the initial model output to this prompt.

The current problem settings of model editing can be generally categorized into three groups (Yao et al., 2023):

1) **Single instance Editing** evaluates the post-edit model performance when only one single knowledge update is performed:

$$\theta' \leftarrow \text{argmin}_\theta(\| f_\theta(x_t) - y_t \|) \qquad (2)$$

2) **Batch Editing** evaluates the post-edit model performance in a more realistic scenario where multiple knowledge pieces are modified simultaneously:

$$\theta' \leftarrow \text{argmin}_\theta \sum\nolimits_{t=1}^{n}(\| f_\theta(x_t) - y_t \|) \qquad (3)$$

where $n \leq | \mathcal{X}_t |$ is the batch size and it varies for different methods (Meng et al., 2023; Mitchell et al., 2022a,b; Meng et al., 2022).

3) **Sequential Editing** requires every single edit to be performed successively, and evaluation has to be conducted after a series of knowledge updates (Hartvigsen et al., 2022):

$$\theta_{|\mathcal{X}_t|}' \leftarrow \text{argmin}_{\theta_t} \sum\nolimits_{t=1}^{|\mathcal{X}_t|}(\| f_{\theta_t}(x_t) - y_t \|) \qquad (4)$$

In this work, we investigate a new and more practical setting for model editing, namely **Consecutive Batch Editing**, which aims at executing the editing in a consecutive batch editing way:

$$\theta_{\lceil |\mathcal{X}_t|/n \rceil}' \leftarrow \underset{\theta_s}{\text{argmin}} \sum_{s=0}^{\lceil |\mathcal{X}_t|/n \rceil} \sum_{t=s\times n}^{\min((s+1)\times n, |\mathcal{X}_t|)} (\| f_{\theta_s}(x_t) - y_t \|) \qquad (5)$$

where s represents the consecutive editing step.

3 Method

We first discuss our method under the single-layer consecutive batch editing setting. Explicitly, we first discuss the process of extending the single-layer updating mechanism in MEMIT (Meng et al., 2023) from a scenario of single-round batch editing to consecutive batch editing. Then, we introduce the hook layer and the local editing scope identification operation employed in the hook layer. The practicality of the operation is also clarified. Finally, we broadening the method from single-layer to multi-layer scenarios.

3.1 Single-Layer Consecutive Batch Editing

3.1.1 Batch Editing Menchanism

Meng et al. (2023) demonstrate an effective single-layer editing method using minimal squared error. Although it supports multiple edits on a single round, the updates do not account for scenarios involving consecutive updates. In this section, we extend this approach to include consecutive scenarios. Following (Meng et al., 2023, 2022), we analyse the model layer weights W_0 as a linear associative memory (Kohonen, 1972; Anderson, 1972) that stores associations between a set of keys k_i and values v_i using minimal squared error:

$$W_0 = \text{argmin}_W \sum_{i=1}^{n} \|Wk_i - v_i\|^2 \quad (6)$$

In this work, W_0 is the weight of the second layer of the model's FFN part (denoted as W^l_{proj}). For simplicity, we stack keys and values into matrices $K_0 = [k_1|k_2|...|k_n]$ and $V_0 = [v_1|v_2|...|v_n]$, then Eq.6 can be optimized by solving (Strang, 2022):

$$W_0 K_0 K_0^T = V_0 K_0^T \quad (7)$$

Thanks to the well-conducted pre-training procedure for most of the available LLMs, we can assume that the pre-trained weight W_0 satisfies Eq.7, *i.e.*, serves as the optimal solution for Eq.6.

Unlike Meng et al. (2023), we define a successive mass-editing objective:

$$\hat{W}_1 = \text{argmin}_W (\sum_{i=1}^{r} \|Wk_i - v_i\|^2 + \sum_{i=r+1}^{r+u} \|Wk_i - v_i\|^2) \quad (8)$$

Following Eq.7, we conclude that Eq.8 can be optimized if we can solve:

$$\hat{W}_1 [K_1\ K_2][K_1\ K_2]^T = [V_1\ V_2][K_1\ K_2]^T \quad (9)$$

where $K_1 = [k_1|k_2|\dots|k_r] (r \geq n)$ and $V_1 = [v_1|v_2|\dots|v_r]$ is the set of key-value pairs that have been updated and $K_2 = [k_{r+1}|k_{r+2}|...|k_{r+u}]$ and $V_2 = [v_{r+1}|v_{r+2}|...|v_{r+u}]$ is the set of key-values that are going to be edited. Therefore, the objective (Eq.8) indicates that we want an optimal $\hat{W}_1$ that successfully updates the new associations while maintaining the old key-value pairs.

Further expanding Eq.9:

$$(W_1 + \Delta)(K_1K_1^T + K_2K_2^T) = (V_1K_1^T + V_2K_2^T) \quad (10)$$

$$W_1K_1K_1^T + \Delta K_1K_1^T + W_1K_2K_2^T + \Delta K_2K_2^T = V_1K_1^T + V_2K_2^T \quad (11)$$

The Δ means the desired weight change to update the new associations K_2, V_2 and W_1 is the weight that has been updated for the associations K_1, V_1 (Note that $W_1 = W_0$ if and only if $r = n$). In a real consecutive editing scenario, r increases and starts with n, and each batch-editing iteration is optimized through the objective (Eq.8). Hence, we can conclude that $W_1K_1K_1^T = V_1K_1^T$. Subtracting it from Eq.11, we get:

$$\Delta K_1K_1^T + W_1K_2K_2^T + \Delta K_2K_2^T = V_2K_2^T \quad (12)$$

Further rearranging it, we have:

$$\Delta = RK_2^TC_{accu}^{-1} \quad (13)$$

where $R = (V_2 - W_1K_2)$ is the residual error evaluated on the most recent updated weights. $C_{accu} = (K_1K_1^T + K_2K_2^T)$ is the accumulation sum of editing keys' outer product, and we have

$$K_1K_1^T = K_0K_0^T + K'K'^T \quad (14)$$

where K_0 is the set of pre-training keys that have been contained in the pre-training weight, $K' = [k_{n+1}|k_{n+2}|...|k_r]$ denotes the updated keys proceeding to current editing step. We follow (Meng et al., 2023) to model $K_0K_0^T$ as the uncentered covariance of some randomly sampled inputs:

$$K_0K_0^T = \lambda E[kk^T] \quad (15)$$

Note that the λ represents a factor that balances the pre-trained and the whole updated associations. We follow the definitions of keys and values in (Meng et al., 2023, 2022), where keys are the activations at the last token of the subject (such as "Joe Biden" for example provided in §1) and values are gradient-descent optimized vectors that maximize the model's prediction for the target object.

3.1.2 Hook Layer

Yao et al. (2023) demonstrate that those editing methods that directly modify the model parameter in place struggle with sequential editing. Specifically, the locality decreases drastically when the number of iterations increases. Meanwhile, those methods that freeze the model parameters show more stable performance over iterations. This indicates that it might be helpful to separate the editing change from the model itself. However, directly applying an external memory (Mitchell et al., 2022b; Hartvigsen et al., 2022) that grows over time for a consecutive batch editing scenario is too memory-costly. Therefore, we aim to seek an approach that

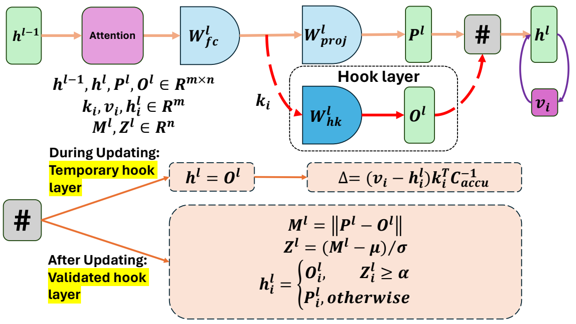

Figure 1: Single layer update with hook layer (residual connections are omitted). $\| \, . \, \|$ means calculate the L2-norm over the keys' dimension (m). During updating the weights, the temporary hook layer is used to ensure Δ is computed based on W^l_{hk}. After the weights update, the validated hook layer is applied to determine whether to use the original layer or hook layer for each token.

could store associations without regularly increasing external memory while preserving the original model parameters.

In light of these motivations, we introduce the hook layer (Fig.1), which takes the original model layer weights as the weight initialization and is responsible for all editing weight alteration in the whole editing process of CoachHooK. It is similar to the forward hook function defined in popular ML libraries like PyTorch, which adjusts the original forward layer output based on predefined criteria. Theoretically, the hook layer can be hung on any target linear layer in the transformer. Nevertheless, we mainly focus on the critical path identified in (Meng et al., 2023, 2022) as they are verified to be crucial for fact association storage in the autoregressive language model.

As shown in Fig.1, there are generally two sorts of hook layers in this work, namely, the **Temporary hook layer** and the **Validated hook layer**. The temporary hook layer is temporarily applied during the weight updating process. It replaces the original output with the output from the hook layer so that the residual is computed on the basis of the hook layer weight. The hook layer weights are then updated (Eq.13) using the calculated residual and the accumulated sum of the keys' outer product. Validated hook layers are employed after the weight updating process at the layer, and inherit the updated weights from the temporary hook layer.

3.1.3 Local Editing Scope Identification

Outlier Detection Given the original outputs produced by the model layer weights and the edited outputs generated by hook layer weights, we need to decide when and which part of the original outputs to swap over. The ideal solution is only to switch those parts of outputs whose keys have been updated to the hook layer weights and leave other parts unchanged. To this end, we first detect the output parts that have their keys updated. Suppose $k_i \in K_1, v_i \in V_1$ is an association that has been updated to W_1, and $k_j \notin K_1$ is a key that is not included in the updated associations. We show empirically in section 4.4 that $\| \; W_1k_i - W_0k_i \; \| \gg \| \; W_1k_j - W_0k_j \; \|$ holds. This implies that when the hook layer with updated weight W_h receives an input $\hat{K} \in R^{m\times n}$ (batch dimension is ignored for simplicity) that contains an edited key $k_i \in \hat{K} \cap K_1$, $k_i \in R^m$, then we should have $\| \; W_hk_i - W_0k_i \; \| \gg \| \; W_hk_j - W_0k_j \; \|$ for all $k_j \in \hat{K} - \hat{K} \cap K_1$, which means that $\| \; W_hk_i - W_0k_i \; \|$ would be outliers among $\{\| \; W_hk_x - W_0k_x \; \|: \forall k_x \in \hat{K}\}$. Hence, detecting outputs of the updated keys can be transferred to detecting the outliers in the L2-norm distribution of inputs. We used the standardization to find the outliers (Fig.1), which applies the standardization technique to L2-norm vectors of inputs and determines outliers via a predefined threshold α. Concretely, for the inputs $\hat{K}$, we first compute the L2-norm vector $M^l \in R^n$:

$$P^l = W_0\hat{K} \quad O^l = W_h\hat{K} \tag{16}$$

$$M^l = \| \; (O^l - P^l) \; \| \tag{17}$$

Note that $\| \, . \, \|$ here means computing the L2-norm for each vector over the keys' dimension (m). Then, we standardize M^l to get the z-score vector Z^l and select the swap location by comparing it with α. The details of choosing α are discussed in the next paragraph. Specifically, we do:

$$h^l_i = \begin{cases} O^l_i & \text{if } Z^l_i \geq \alpha, \\ P^l_i & \text{orrtherwise.} \end{cases} \tag{18}$$

where i is the index over tokens.

Threshold α Determination We denote $Z^l_i = \max((M^l - \mu)/\sigma)$ as the maximum z-score entry of an input $\hat{K}$. Since the Z^l_i varies for different instances (§4.4) and is likely to shift as the consecutive editing steps grow, it is unreasonable to set α as a fixed real number. Therefore, we determine the α dynamically during the editing process:

$$\alpha_s = \begin{cases} \alpha_z & \text{if } s = 1, \\ \min(\alpha_c, \alpha_{s-1}) & \text{otherwise} \end{cases} \tag{19}$$

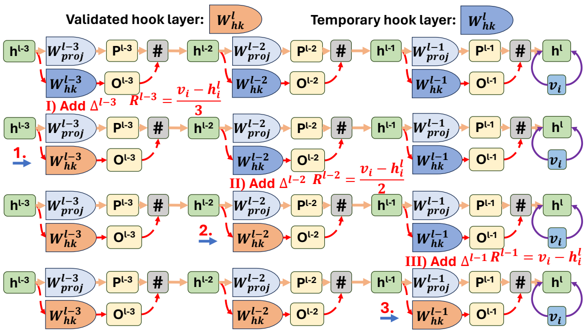

Figure 2: Multiple layer update with hook layer (Attention module and the first layer of FFN are omitted). The value vector v_i is first computed at the last editing layer, and then we iteratively insert a fraction of the residual to each editing layer (I, II, III) using Eq.13. Since changing one layer would affect the activations of downstream layers, recollection of the activations is conducted after each iteration. At the beginning, temporary hook layers are initialized to all editing layers. Once the hook layer weight is updated, it is replaced by the validated hook layer (1, 2, 3).

where $s \geq 1$. Specifically, the α is first initialized to a pre-selected value α_z. At each consecutive editing step s, for the batch of inputs in this step, we calculate Z_i^l (the maximal z-score entry) for each single instance and select the minimal Z_i^l in the batch (*i.e.*, the supremum) as the candidate α_c. The α_s is finally determined to be the minimum between the candidate α_c and the previous value α_{s-1}. In practice, we set $\alpha_z = 2.2$.

3.2 Multiple-layer Consecutive Batch Editing

Given the designed single-layer editing procedure, there exists a risk that the single-layer hook fails to detect the updated keys. Suppose k_i is an updated key; failure to detect k_i indicates that the output corresponds to k_i at this single layer would be the same as the original output $W_{proj}k_i$, which consequently leads to the failure update for k_i. To tackle this issue, one potential solution is to apply the hook to multiple model layers rather than a single model layer because the latter layer grasps the chance to capture the edited keys missed by proceeding layers. Furthermore, (Zhu et al., 2020) showcased that minimizing the magnitude of parameter change is helpful for improving the robustness of the model. Thus, we expand our work to multiple layers (Fig.2).

We first find the desired object vector v_i following a similar procedure in (Meng et al., 2023). However, the optimization is not based on the original model, but the model hung with the validated hook that inherits the most recently updated hook weights from the previous editing step. After v_i is found, the hook weight is updated at each layer.

At each batch editing step, all the hook layers are initialized to temporary hook layers, which substitute the entire original output to output from hook layers. The purpose of doing this is to ensure that the residual regarding the hook layer weights rather than the original model weights are calculated. Then, the residual is distributed evenly to each layer, and the alteration Δ^l to the parameter at each layer is found in a layer-increasing iterative manner with keys and residuals recomputed at each iteration (Fig.2). The reason for the recomputation of keys and residuals is that the layer-increasing alteration approach will affect the keys and residuals in the latter layer. For each layer, once the hook layer weight is updated, the hook layer is changed from a temporary hook layer to a validated hook layer to facilitate the computation of the keys and residuals in the latter layer. After the whole editing process is completed, the validated hook layers with the ultimately updated weights are hung on the model to shape the final edited model.

4 Experiments

4.1 Experiment Setups

Datasets & Evaluation Metrics We use the ZsRE (Levy et al., 2017) and COUNTERFACT (Meng et al., 2023) datasets with the split provided in EasyEdit[1] for evaluation. We employ three popular editing evaluation metrics defined in (Yao et al., 2023; Huang et al., 2023; Cao et al., 2021), *i.e.*, Reliability, Generality, and Locality, as well as the average scores[2] over the three metrics. Further details are provided in Appendix A.

Baselines & Implementation Details For baselines, we adopt several batch-supportive editing methods, including LoRA (Hu et al., 2022), SERAC (Mitchell et al., 2022b), MEND (Mitchell et al., 2022a), MEMIT (Meng et al., 2023) and fine-tuning with specific layer (FT-L) technique used in (Meng et al., 2022; Yao et al., 2023), which only fine-tune a specific layer identified by Rome (Meng et al., 2022) instead of all layers to ensure a fair comparison. We also include a small variation of FT-L called FT-M and a sequential supportive editing method GRACE (Hartvigsen et al., 2022). We choose large autoregressive language models

[1] https://github.com/zjunlp/EasyEdit/tree/main

[2] Most of the application scenarios of model editing require good performance in all three metrics.

Method	Model	ZsRE				COUNTERFACT			
		Reliability	Generality	Locality	Average	Reliability	Generality	Locality	Average
FT-L (Meng et al., 2022)	GPT2-XL	16.85	16.34	71.55	34.91	0.27	0.34	85.18	28.60
FT-M	GPT2-XL	17.95	17.32	71.26	35.51	0.36	0.42	82.81	27.86
LoRA (Hu et al., 2022)	GPT2-XL	30.10	29.08	80.54	46.57	5.64	3.46	69.45	26.18
MEND (Mitchell et al., 2022a)	GPT2-XL	2.16	2.11	20.34	8.20	0.13	0.03	4.22	1.46
SERAC (Mitchell et al., 2022b)	GPT2-XL	**98.64**	48.12	35.68	60.81	17.88	14.55	82.25	38.23
MEMIT (Meng et al., 2023)	GPT2-XL	61.19	49.97	97.51	**69.56**	81.01	27.67	95.80	**68.16**
CoachHooK	GPT2-XL	82.21	**66.61**	**99.40**	**82.74**	**88.28**	**40.38**	**97.66**	**75.44**
FT-L (Meng et al., 2022)	GPT-J	22.57	21.77	**99.19**	47.84	0.37	0.34	99.57	33.43
FT-M	GPT-J	99.96	80.31	43.35	74.54	**99.99**	35.29	17.04	50.77
LoRA (Hu et al., 2022)	GPT-J	**99.97**	**83.20**	17.64	66.93	99.87	**53.10**	2.50	51.82
SERAC (Mitchell et al., 2022b)	GPT-J	87.46	63.64	77.35	76.15	16.67	15.93	**99.99**	44.20
MEMIT (Meng et al., 2023)	GPT-J	93.40	70.45	96.47	**86.77**	99.57	42.29	95.25	**79.04**
CoachHooK	GPT-J	97.59	72.41	99.10	**89.70**	87.94	42.76	98.17	**76.29**

Table 1: Single round batch editing results. The best two average scores are highlighted.

GPT2-XL and GPT-J (6B) as our base models. Further details of the baselines and implementation are given in the Appendix B.

4.2 Evaluation on Single-round Batch Editing

We first test the effectiveness of our method under basic single-round batch editing settings with batch size 30, *i.e.*, the model is rolled back to the initial state after each batch editing. Both MEMIT and CoachHooK need to set the parameter λ, the balance factor between pre-trained and newly updated associations. According to (Meng et al., 2023), higher λ helps preserve the original model behavior (locality) but could harm reliability and generality, and the best overall performance is found at around $\lambda = 10^4$. However, with the intent to verify whether our method comprehensively improves the editing, that is, could accept lower λ to assign higher weight for new associations while not sacrificing the locality, we deliberately set $\lambda = 5 \times 10^3$ for CoachHooK and keep it as the optimized value for MEMIT, which are 2×10^4 and 1.5×10^4 for GPT2-XL and GPT-J respectively.

The evaluation results are shown in Table 1. For GPT2-XL, our method has the best result in almost every metric. Specifically, despite the relatively low λ, our method overwhelms other baselines in generality metrics while maintaining a better locality. This indicates that lowering λ or, in other words, increasing the weight of the new associations does not sacrifice the locality in CoachHooK. The improvement in GPT-J is less compared with that in GPT2-XL. However, our method still has the best average score for the ZsRE dataset and a comparable average score with the best in the COUNTERFACT dataset.

4.3 Evaluation on Consecutive Batch Editing

We evaluate our method's capability on 1k samples from both datasets for consecutive batch editing, *i.e.*, there is no roll-back. The evaluation is conducted after the end of the whole consecutive batch editing process. We set λ to $15,000$ as the scenario now is consecutive batch editing.

Results in Table 2 show that most of the methods suffer from a great performance drop contrasted to editing in a single round. Although our method's performance experiences a decrease as well, it surpasses other methods in 100 consecutive steps with an even larger improvement margin for almost all the metrics compared to the single-round batch editing. This demonstrates that our method does not depend on simple regurgitation of the editing samples nor rely heavily on the trade-offs of lowering the balancing factor λ to increase the reliability and locality. An interesting point is that the GRACE performs perfectly in reliability and locality but poorly in generality. As expected, GRACE is superior in reliability since it maintains a codebook to memorize the encountered editing instances. However, its inferiority in generality indicates that it suffers from the problem of regurgitation.

We extend the data scale of the consecutive batch editing experiment to 10k (1k consecutive steps) to explore the limit of our method. Results can be found in Fig.6. Surprisingly, the locality experiences a great fall from 100 to 200 steps but remains steady from 200 to 1k editing steps, which proves that the hook layer stably obstructs the out-scope samples. Reliability and generality consistently fall as the consecutive steps grow, indicating that there is still room for improvement in this field.

Method	Model	ZsRE				COUNTERFACT			
		Reliability	Generality	Locality	Average	Reliability	Generality	Locality	Average
FT-L (Meng et al., 2022)		3.79	2.48	6.60	4.29	1.00	1.00	6.00	2.67
FT-M		8.92	8.41	6.22	7.85	4.00	3.50	5.50	4.33
LoRA (Hu et al., 2022)		0.96	1.29	0.03	0.76	0.50	0.02	0.50	0.34
MEND (Mitchell et al., 2022a)	GPT2-XL	20.95	18.29	93.69	47.01	0.01	0.00	0.08	0.03
SERAC (Mitchell et al., 2022b)		100	36.03	35.95	57.33	15.41	12.96	81.00	36.46
GRACE (Hartvigsen et al., 2022)		**100**	0.04	**100**	**66.68**	**100**	0.40	**100**	**66.80**
MEMIT (Meng et al., 2023)		34.88	32.96	70.74	46.19	56.00	37.00	31.00	41.33
CoachHooK		66.91	**56.11**	97.23	**73.42**	86.00	**38.00**	59.00	**61.00**
FT-L (Meng et al., 2022)		23.53	21.70	55.27	33.5	2.00	2.00	72.00	25.33
FT-M		64.33	55.63	17.59	45.85	25.50	5.00	2.00	10.83
LoRA (Hu et al., 2022)		1.43	1.39	0.02	0.95	0.50	0.50	0.10	0.37
SERAC (Mitchell et al., 2022b)	GPT-J	86.91	55.36	79.07	73.78	18.49	14.56	98.89	43.98
GRACE (Hartvigsen et al., 2022)		**100**	0.04	**100**	**66.68**	**100**	0.50	**100**	**66.83**
MEMIT (Meng et al., 2023)		63.36	48.90	74.80	62.35	75.00	**45.00**	42.00	54.00
CoachHooK		79.89	**61.29**	96.52	**79.23**	95.00	41.00	80.00	**72.00**

Table 2: Consecutive batch editing results.

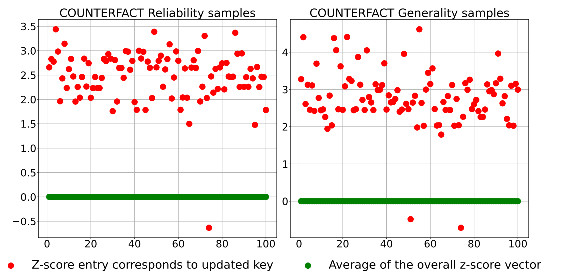

Figure 3: Difference between the z-score entry to the updated key Z^l_{key} and average of Z^l. The x-axis represents the sample index.

4.4 Validation of Local Editing Scope

Given an updated hook layer with the weight W_h, the original model weight W_0, an updated key k_i, and an out-of-scope key k_j, we conduct experiments to verify whether $\| W_h k_i - W_0 k_i \| \gg \| W_h k_j - W_0 k_j \|$ holds. We select 100 samples from the COUNTERFACT dataset to edit GPT2-XL using CoachHooK, then apply the edited model to these 100 samples and record the z-score entries of the L2-norm of the difference vector between update keys' response from the last hook layer and original model layer, namely, z-score entries of $\| W_h k_i - W_0 k_i \|$. Both reliability and generality prompts are included for comprehensiveness.

The result is shown in Fig.3. Almost all the z-scores of the responses from updated keys exhibit a great margin from the mean value, with the lowest around 1.5 in reliability samples and 2 in generality samples. The discriminative z-score demonstrates that the identification technique (section 3.1.3) can effectively filter editing-irrelevant instances and accept editing-relevant instances, which validates the local editing scope.

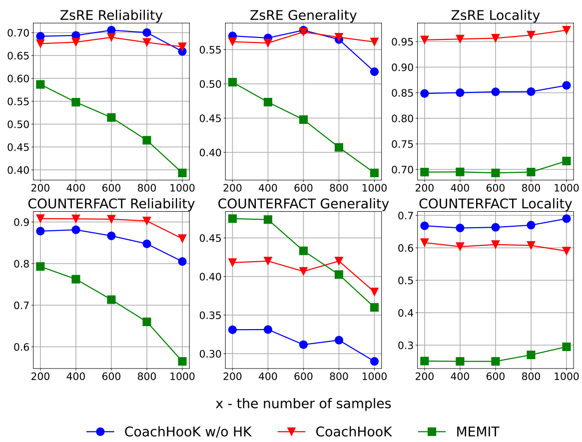

Figure 4: Ablation study.

4.5 Detailed Analysis and Discussions

Ablation Study of Update mechanism and Hook layers The effectiveness of the derived consecutive updating mechanism and the hook layers are discussed in this part. We run three cases using GPT2-XL, namely, MEMIT (no consecutive updating mechanism, no hook layers), CoachHooK without hook (CoachHooK w/o HK), and CoachHooK for consecutive batch editing on 1k samples from both ZsRE and COUNTERFACT datasets.

The results are demonstrated in Fig.4. In almost all metrics of the two datasets except the generality of COUNTERFACT, the CoachHooK w/o the hook performs better than the vanilla MEMIT, and the margin tends to increase as the consecutive steps ascend. This certifies the effectiveness of our derived consecutive updating mechanism in consecutive batch editing scenarios. For the ZsRE dataset, the method with hook layers considerably outperforms the one without hook in the locality without sacrificing reliability and generality. This verifies that the

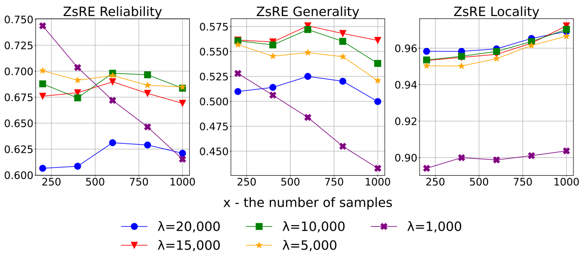

Figure 5: Performance comparisons on initial five different values of λ.

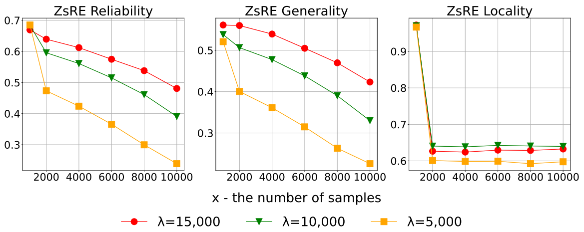

Figure 6: Extension on the best three values of λ.

hook layer can efficiently and accurately block the out-scope instances from the input without fraudulently missing in-scope instances. For the COUNTERFACT dataset, the reliability of CoachHooK is consistently higher than the other two, and the generality surpasses that of MEMIT after 80 editing steps. Besides, the hook layer causes some side effects in the locality of COUNTERFACT, but this circumstance is not found in the ZsRE dataset. It is worth noting that CoachHooK shows the most stable performance as the number of consecutive editing steps grows, showing the great potential of our method for consecutive editing.

Effect of the Balance Factor λ We test the effect of different λ, the balance factor between pre-training and newly updated associations. We first evaluate the CoachHooK with different λ on 1k samples from ZsRE (Fig.5). It seems that a small value of $\lambda = 1,000$ would cause significant damage to all three metrics, especially the reliability and generality, since they experience a great drop as the consecutive steps increase. This may result from the overly high magnitude of the weight change caused by the low value of λ, which severely distorts the previously updated associations. Meanwhile, a too-high value of $\lambda = 20,000$ also seems not to be a good choice, which gives rise to an overly small magnitude of the weight change so that it fails to deliver the new optimized values for keys. The cases of $\lambda = 5000, 10000, 15000$ do not show an apparent difference, so we extend further the sample size to 10k (Fig.6).

Extended results show that 5000 is not a good choice for large-consecutive editing steps, though it performs no worse than the other two in early 1k samples. The case of $\lambda = 15,000$ ranks first in reliability and generality. Although it performs worse in locality compared to $\lambda = 10,000$, the margin between them gradually narrows as the consecutive steps rise. Overall, we conclude that 15,000 would be a reasonable selection.

More Analyses Other detailed analyses of hyperparameters and inference time of the proposed method are presented in Appendix C

5 Related Work

Recent years have witnessed prosperous development in the field of model editing. According to (Yao et al., 2023), the proposed methods so far can be generally classified into two groups, *i.e.*, modify the model's weight or not. The methods that do not directly alter the model weights generally follow two directions: they either employ an external memory or introduce additional adjustable parameters. Methods like T-Patcher (Huang et al., 2023) and CaliNET (Dong et al., 2022) apply new neurons that are responsible for specific mistakes in the last layer of the FFN model. GRACE (Hartvigsen et al., 2022) introduces a timely adjusted code book to edit the model's behavior. Another group of methods like (Mitchell et al., 2022b) integrates an explicit external memory as edit descriptors to help editing scope recognition. On the other hand, those directly altering the model's weight either train a hyper-network to predict the change required by the edits (Mitchell et al., 2022a; De Cao et al., 2021) or first locate corresponding parameters that are responsible for specific knowledge and then edit the located parameters (Meng et al., 2023, 2022; Li et al., 2023; Dai et al., 2022).

6 Conclusion

This work introduces a novel model editing method, CoachHooK, which advocates the more practical consecutive batch model editing. CoachHooK uses an expanded editing mechanism to support consecutive editing and newly proposed hook layers to identify the editing scope. Compared to existing model editing methods, CoachHooK does not require large external memory nor extra training for meta-networks or classifiers. Instead, it adopts hook layers whose size remains fixed over time for storing associations. Comprehensive experiments are conducted to verify the method's effectiveness over single-round and consecutive batch editing.

Limitations

Several aspects remain to be further investigated.

Other types of tasks Notably, model editing techniques could be applied to various types of tasks. Specifically, besides factual knowledge editing, it can be applied to erase hallucinations, biases, privacy information, etc. However, the concentration of this paper is to explore the practicability of expanding the model editing application scenario to consecutive batch editing and investigate the potential bottleneck of corresponding methods under this scenario. Therefore, our experiment focuses on varying the scale of editing samples in factual knowledge editing tasks, as it is a relatively well-studied and universal evaluation task in model editing.

Model scale and architecture Due to the limited computational resources, we cannot verify our method's effectiveness in larger LLMs such as Llama-2 (Touvron et al., 2023), and GPT-NEOX-20B (Black et al., 2022). We focus on the decoder-only autoregressive models and do not include encoder-decoder structure models, as the autoregressive structures are the mainstream architecture nowadays (OpenAI, 2023; Touvron et al., 2023). Further, as stated by Yao et al. (2023), the weight matrix in some models like OPT-13B (Zhang et al., 2022) is not invertible. However, such an issue can be relieved by adding a term βI to the Eq.14, where β is a scalar expected to be small and I is the identity matrix.

The shrink of α As more and more associations are integrated into the hook layer, the dynamically determined hyperparameter α will gradually shrink, meaning that an increasing number of vector entries in the original layer output will be swapped by the output from the hook layer, which is likely to lead the drop in locality . Nevertheless, such a problem can be alleviated by the newly designed updated mechanism (Eq.13), which considers both previously updated and newly updated keys.

References

Oshin Agarwal and Ani Nenkova. 2022. Temporal effects on pre-trained models for language processing tasks. *Trans. Assoc. Comput. Linguistics*, 10:904–921.

James A Anderson. 1972. A simple neural network generating an interactive memory. *Mathematical biosciences*, 14(3-4):197–220.

Sid Black, Stella Biderman, Eric Hallahan, Quentin Anthony, Leo Gao, Laurence Golding, Horace He, Connor Leahy, Kyle McDonell, Jason Phang, Michael Pieler, USVSN Sai Prashanth, Shivanshu Purohit, Laria Reynolds, Jonathan Tow, Ben Wang, and Samuel Weinbach. 2022. Gpt-neox-20b: An open-source autoregressive language model. *CoRR*, abs/2204.06745.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 6491–6506. Association for Computational Linguistics.

Siyuan Cheng, Bozhong Tian, Qingbin Liu, Xi Chen, Yongheng Wang, Huajun Chen, and Ningyu Zhang. 2023. Can we edit multimodal large language models? *arXiv preprint arXiv:2310.08475*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Eric Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, Albert Webson, Shixiang Shane Gu, Zhuyun Dai, Mirac Suzgun, Xinyun Chen, Aakanksha Chowdhery, Sharan Narang, Gaurav Mishra, Adams Yu, Vincent Y. Zhao, Yanping Huang, Andrew M. Dai, Hongkun Yu, Slav Petrov, Ed H. Chi, Jeff Dean, Jacob Devlin, Adam Roberts, Denny Zhou, Quoc V. Le, and Jason Wei. 2022. Scaling instruction-finetuned language models. *CoRR*, abs/2210.11416.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8493–8502, Dublin, Ireland. Association for Computational Linguistics.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–

6506, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Qingxiu Dong, Damai Dai, Yifan Song, Jingjing Xu, Zhifang Sui, and Lei Li. 2022. Calibrating factual knowledge in pretrained language models. In *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, pages 5937–5947. Association for Computational Linguistics.

Thomas Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. 2022. Aging with GRACE: lifelong model editing with discrete key-value adaptors. *CoRR*, abs/2211.11031.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Zeyu Huang, Yikang Shen, Xiaofeng Zhang, Jie Zhou, Wenge Rong, and Zhang Xiong. 2023. Transformer-patcher: One mistake worth one neuron. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Haoqiang Kang, Juntong Ni, and Huaxiu Yao. 2023. Ever: Mitigating hallucination in large language models through real-time verification and rectification. *CoRR*, abs/2311.09114.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Teuvo Kohonen. 1972. Correlation matrix memories. *IEEE Trans. Computers*, 21(4):353–359.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomás Kociský, Sebastian Ruder, Dani Yogatama, Kris Cao, Susannah Young, and Phil Blunsom. 2021. Mind the gap: Assessing temporal generalization in neural language models. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 29348–29363.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 333–342, Vancouver, Canada. Association for Computational Linguistics.

Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. 2023. PMET: precise model editing in a transformer. *CoRR*, abs/2308.08742.

Adam Liska, Tomás Kociský, Elena Gribovskaya, Tayfun Terzi, Eren Sezener, Devang Agrawal, Cyprien de Masson d'Autume, Tim Scholtes, Manzil Zaheer, Susannah Young, Ellen Gilsenan-McMahon, Sophia Austin, Phil Blunsom, and Angeliki Lazaridou. 2022. Streamingqa: A benchmark for adaptation to new knowledge over time in question answering models. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 13604–13622. PMLR.

Shengyu Mao, Ningyu Zhang, Xiaohan Wang, Mengru Wang, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. 2023. Editing personality for llms. *arXiv preprint arXiv:2310.02168*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Kevin Meng, Arnab Sen Sharma, Alex J. Andonian, Yonatan Belinkov, and David Bau. 2023. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022a. Fast model editing at scale. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022b. Memory-based model editing at scale. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 15817–15831. PMLR.

Niels Mündler, Jingxuan He, Slobodan Jenko, and Martin T. Vechev. 2023. Self-contradictory hallucinations of large language models: Evaluation, detection and mitigation. *CoRR*, abs/2305.15852.

OpenAI. 2023. GPT-4 technical report. *CoRR*, abs/2303.08774.

Fabio Petroni, Patrick S. H. Lewis, Aleksandra Piktus, Tim Rocktäschel, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. 2020. How context affects language models' factual predictions. In *Conference on Automated Knowledge Base Construction, AKBC 2020, Virtual, June 22-24, 2020*.

Yifu Qiu, Yftah Ziser, Anna Korhonen, Edoardo Maria Ponti, and Shay B. Cohen. 2023. Detecting and mitigating hallucinations in multilingual summarisation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP*

2023, Singapore, December 6-10, 2023, pages 8914–8932. Association for Computational Linguistics.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Gilbert Strang. 2022. *Introduction to linear algebra*. SIAM.

S. M. Towhidul Islam Tonmoy, S. M. Mehedi Zaman, Vinija Jain, Anku Rani, Vipula Rawte, Aman Chadha, and Amitava Das. 2024. A comprehensive survey of hallucination mitigation techniques in large language models. *CoRR*, abs/2401.01313.

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, Aurélien Rodriguez, Robert Stojnic, Sergey Edunov, and Thomas Scialom. 2023. Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Neeraj Varshney, Wenlin Yao, Hongming Zhang, Jianshu Chen, and Dong Yu. 2023. A stitch in time saves nine: Detecting and mitigating hallucinations of llms by validating low-confidence generation. *CoRR*, abs/2307.03987.

Peng Wang, Ningyu Zhang, Xin Xie, Yunzhi Yao, Bozhong Tian, Mengru Wang, Zekun Xi, Siyuan Cheng, Kangwei Liu, Guozhou Zheng, et al. 2023. Easyedit: An easy-to-use knowledge editing framework for large language models. *arXiv preprint arXiv:2308.07269*.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. 2023. Editing large language models: Problems, methods, and opportunities. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 10222–10240. Association for Computational Linguistics.

Ningyu Zhang, Yunzhi Yao, Bozhong Tian, Peng Wang, Shumin Deng, Mengru Wang, Zekun Xi, Shengyu Mao, Jintian Zhang, Yuansheng Ni, et al. 2024. A comprehensive study of knowledge editing for large language models. *arXiv preprint arXiv:2401.01286*.

Ningyu Zhang, Jintian Zhang, Xiaohan Wang, Honghao Gui, Kangwei Liu, Yinuo Jiang, Xiang Chen, Shengyu Mao, Shuofei Qiao, Yuqi Zhu, Zhen Bi, Jing Chen, Xiaozhuan Liang, Yixin Ou, Runnan Fang, Zekun Xi, Xin Xu, Lei Li, Peng Wang, Mengru Wang, Yunzhi Yao, Bozhong Tian, Yin Fang, Guozhou Zheng, and Huajun Chen. 2023. Knowlm technical report.

Susan Zhang, Stephen Roller, Naman Goyal, Mikel Artetxe, Moya Chen, Shuohui Chen, Christopher Dewan, Mona T. Diab, Xian Li, Xi Victoria Lin, Todor Mihaylov, Myle Ott, Sam Shleifer, Kurt Shuster, Daniel Simig, Punit Singh Koura, Anjali Sridhar, Tianlu Wang, and Luke Zettlemoyer. 2022. OPT: open pre-trained transformer language models. *CoRR*, abs/2205.01068.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix X. Yu, and Sanjiv Kumar. 2020. Modifying memories in transformer models. *CoRR*, abs/2012.00363.

A Experiment Details

All baselines are implemented using the EasyEdit (Yao et al., 2023; Zhang et al., 2024; Wang et al., 2023; Cheng et al., 2023; Mao et al., 2023; Zhang et al., 2023) library.

Evaluation Metrics We employ three popular editing evaluation metrics defined in (Yao et al., 2023; Huang et al., 2023; Cao et al., 2021), *i.e.*, reliability, generality, and locality. Given an initial base model f_θ, a post-edit model $f_{\theta'}$, and a set of edit instances $\{(x_t, y_t)\}$, the reliability is computed as the average accuracy of the edit cases:

$$\mathbb{E}_{(x_t,y_t)\in\{(x_t,y_t)\}}\{\arg\max_y f_{\theta'}(y|x_t) = y_t\}\,. \tag{20}$$

The editing should also edit the equivalent neighbor of the instance $N(x_t, y_t)$ (*e.g.* rephrased descriptions). This metric is named generality and is evaluated by the average accuracy on the neighbors of the edit cases:

$$\mathbb{E}_{(x_t',y_t')\in\{N(x_t,y_t)\}}\{\arg\max_y f_{\theta'}(y|x_t') = y_t'\}\,. \tag{21}$$

Despite the editing, those instances that are irrelevant to the edit cases $\{O(x_t, y_t)\}$ should not be affected. This evaluation is called locality (also known as specificity) and is measured by the proportion of unchanged predictions between the initial model and the post-edit model:

$$\mathbb{E}_{(x_t',y_t')\in\{O(x_t,y_t)\}}\{f_{\theta'}(x_t') = f_\theta(x_t')\}\,. \tag{22}$$

"subject": "Barbara Legrand",
"src": "What is Barbara Legrand's position on the field while playing football?",
"pred": "midfielder",
"rephrase": "What is Barbara Legrand's position on the field during the football match?",
"alt": "defender",
"answers": ["goalkeeper"],
"loc": "nq question: who played donna in 2 pints of lager",
"loc_ans": "Natalie Casey",
"cond": "midfielder >> defender || What is Barbara Legrand's position on the field while playing football?"

Figure 7: A sample from ZsRE dataset.

"case_id": 0,
"prompt": "The mother tongue of Danielle Darrieux is",
"target_new": "English",
"subject": "Danielle Darrieux",
"ground_truth": "French",
"rephrase_prompt": "Where Danielle Darrieux is from, people speak the language of",
"locality_prompt": "Michel Rocard is a native speaker of",
"locality_ground_truth": "French"

Figure 8: A sample from COUNTERFACT dataset.

Datasets ZsRE is a question-answering dataset that uses back-translation to generate equivalent neighborhoods. It is initially used in factual knowledge evaluation and later adopted in model editing by (Mitchell et al., 2022a). COUNTERFACT is a challenging dataset focusing on counterfactual information with a low prediction score compared to correct facts. It builds out-of-scope data by replacing the subject entity with a similar description that shares the same predicate.

Fig.7 shows an example from the ZsRE dataset. Each record in ZsRE contains the subject string, the factual prompt used for testing reliability, the rephrase prompt used for generality evaluation, and the locality prompt used for evaluating the locality. Note that what the locality demands the post-edit model does is not to predict the ground truth but whatever the initial base model predicts. Similarly, the fact, rephrase, and locality prompts of each record in COUNTERFACT are applied to the evaluation of the three metrics respectively (Fig.8).

B Baselines and Implementation Details

For all consecutive editing experiments, the evaluation is conducted after the full set of consecutive steps are finished. For example, in Fig.4, we conduct experiments for sample sizes 200, 400, 600, 800, and 1000, so the evaluation is triggered right after the first 200, 400, 600, etc, samples are edited to the model. Unless specified, the batch size[3] for consecutive editing is selected to be 10.

Fine-tuning We implemented two fine-tuning methods in the experiments. For FT-L, we follow the procedure in (Meng et al., 2023, 2022) and fine-tune the mlp_{proj} parameter from layer 0 for GPT2-XL and layer 21 for GPT-J since they are found to have the optimal performance. FT-M[4] is a small variation of FT-L, and it uses a different loss computation procedure to optimize the parameters. For both models, we conduct 25 optimization steps using Adam optimizer (Kingma and Ba, 2015) and use learning rate $5e^{-4}$. All other parameters of both models follow default settings.

LoRA Hu et al. (2022) proposed a parameter-efficient fine-tuning method that decomposes the update gradient matrix into two small rank-n matrices, which reduces the required memory for LLM training to a large extent. In all experiments, we set the learning rate and the rank number to $1e^{-3}$ and 8, respectively. The α was chosen to be 32, and the dropout rate was 0.1. The number of update steps is 30 for GPT2-XL and 50 for GPT-J.

MEND MEND (Mitchell et al., 2022a) conducts the editing by manipulating the language models' gradient. It trains a meta-network that employs a rank-1 decomposition of the model gradients and predicts a new rank-1 update to the corresponding model weights. In this work, we train two meta-networks using corresponding training split in the ZsRE and COUNTERFACT datasets for GPT2-XL following the default hyperparameter settings. Due to the large required computation resource for training GPT-J (6B) meta-network, we do not perform training for GPT-J.

SERAC Mitchell et al. (2022b) designed a memory-augmented editing method, which requires an external cache to store explicit editing cases. It also adopts a scope classifier that determines whether an input sample falls in the editing scope and a small counterfactual model for editing the in-scope cases. For both GPT2-XL and GPT-J, we train two separate models for the two datasets, respectively. Following the original paper, we apply distilbert-base-cased (Sanh et al., 2019) for the scope classifier across all models. For the

[3] Since GRACE (Hartvigsen et al., 2022) does not support batch editing, we set the batch size to 1 for GRACE.

[4] https://github.com/zjunlp/EasyEdit/blob/main/hparams/FT/gpt2-xl.yaml

small counterfactual model, we employ GPT2 for GPT2-XL and gpt-j-tiny-random[5] for GPT-J. All hyperparameters follow default settings.

MEMIT MEMIT (Meng et al., 2023) treats the feedforward layer of transform as a linear associative memory and uses a minimum square error optimization to add new key-value associations to layer weights. We follow the original paper to edit the layers in the identified critical path and set the balance factor λ to the optimal value found in the original work. Other parameters for the two models are all set based on configurations in (Meng et al., 2023, 2022).

GRACE Hartvigsen et al. (2022) proposed an editing method that preserves the original model parameters and adopts a codebook maintained by adding, splitting, and expanding keys over time to store relevant edits. We follow the optimized settings in the original paper and set the value optimizing learning rate to 1. The number of iterations for optimizing the values is 100, and the initial ε value is chosen to be 1. The codebook is employed at layers 35 and 25, respectively.

CoachHooK CoachHooK expands the update mechanism in MEMIT to consecutive cases and applies hook layers to separate the weight change from the original model layer. For both models, we set $\lambda = 15,000$, $\alpha_z = 2.2$ for consecutive batch editing. Unless specified, we evaluate our method on full critical path layers identified in (Meng et al., 2023). We employ the same procedure in MEMIT (Meng et al., 2023) to compute the updating keys and the target values, except that the most recently updated model during the process of consecutive editing is applied for relevant computations. We applied "torch.float16" for the GPT-J model for all experiments.

C Detailed analysis and discussions

Effect of the Number of Editing Layers To investigate the necessity of applying the hook layer onto multiple transformer layers, we conduct the consecutive batch editing experiment on the ZsRE dataset for GPT2-XL (Fig.9). As the effect of choosing different layers has already been studied in (Meng et al., 2023), we focus only on the effect of the number of layers. We selected the last one, three, and all layers from the critical path identified in (Meng et al., 2023; Yao et al., 2023), respectively.

[5]https://huggingface.co/anton-l/gpt-j-tiny-random

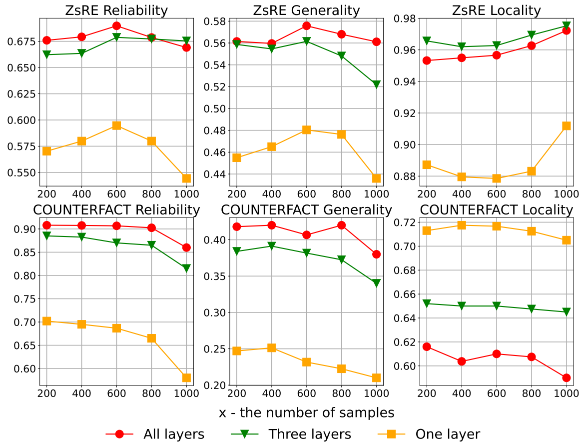

Figure 9: Performance comparisons on the different number of editing layers. Layers are selected from the critical path identified in (Meng et al., 2023).

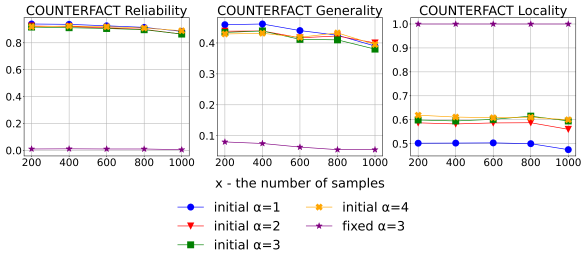

Figure 10: Performance comparisons on different α_z.

As shown in Fig.9, the one-layer case significantly underperforms the other two cases in most of the metrics for the two datasets, which directly certifies the necessity of the expansion. In ZsRE, the difference between the performance for one layer and multiple layers tends to enlarge in reliability and generality as the consecutive editing steps increase. This may serve as evidence of our assumption in section 3.2, which mentions that the latter hook layer may capture the in-scope instances missed in former hook layers. Additionally, the all-layer case has slightly better generality than the three-layer case, and they do not show a remarkable difference in locality and reliability. A similar situation could be found in the reliability and generality of the COUNTERFACT with an interesting exception in the locality, where an adverse performance order of the cases is shown. Nevertheless, the margin of the locality fall is not that manifest in contrast with the advancement in reliability and generality.

Model	Type	Inference Time (s)
GPT2-XL	Pre-edit	0.1187
	Post-edit	0.1297
GPT-J	Pre-edit	0.0762
	Post-edit	0.0863

Table 3: Inference time analysis.

Model	Granularity	Instances		
		Reliability	Generality	Locality
GPT-J	Instances	99.00	97.50	-
	Overall tokens	9.69	12.51	11.19
	Unwanted tokens	0.38	0.13	11.19

Table 4: Percentage of instances/tokens that used the hook layer.

Effect of the Initial Threshold α_z α_z is the initialization value of α used in the identification of local editing scope (section 3.1.3). We study its influence in this part. According to Fig.10, although the $\alpha_z = 1$ case ranks the highest in the first 60 editing steps in generality, it consistently performs the worst in locality, indicating that it fails to intercept many out-scope inputs. This implies that 1 may be too low for the initialization. Other cases do not show noticeable differences in the three metrics since α_z is just the initial value and α is determined dynamically. It seems that overly low α_z would damage the hook layer's capacity to discriminate in-scope and out-scope samples. Considering the unpredictable consecutive steps that our method may be applied, we select a relatively low value between 2 and 3, namely, $\alpha_z = 2.2$.

To verify the significance of the dynamical determination process, we also test the fix α case. We chose the value of 3, the standard threshold used in standardization to detect outliers. The results reveal a dramatic decline in reliability and generality and perfect fulfillment in the locality, indicating that almost all instances are indiscriminately obstructed by the hook layers regardless of the editing scope. Besides, choosing an optimal fixed α before editing is practically unrealistic. Therefore, it would be more reasonable to decide α dynamically.

Effect of Editing Batch Size Does the batch size parameter affect the performance of our method? We investigate the effect of batch size by conducting single-round editing on 1k samples from ZsRE. We tested batch sizes 10, 100, and 1000 (Fig.11).

Deferral Radius	Model	COUNTERFACT			
		Reliability	Generality	Locality	Average
$\varepsilon = 1$	GPT2-XL	100	0.40	100	66.80
$\varepsilon = 3$		100	0.42	100	66.81
$\varepsilon = 5$		100	0.65	99.50	66.72
$\varepsilon = 10$		100	1.80	93.70	65.17
$\varepsilon = 20$		100	18.30	56.60	58.30
$\varepsilon = 30$		100	83.90	7.40	63.77
$\varepsilon = 1$	GPT-J	100	0.50	100	66.83
$\varepsilon = 3$		100	0.54	100	66.85
$\varepsilon = 5$		100	0.57	100	66.86
$\varepsilon = 10$		100	0.68	99.60	66.76
$\varepsilon = 20$		100	5.00	93.20	66.07
$\varepsilon = 30$		100	31.30	58.90	63.40

Table 5: Results of GRACE with increased ε.

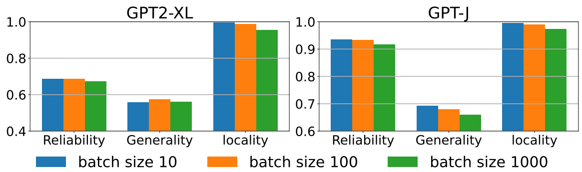

Figure 11: Performance comparisons on different editing batch sizes.

The results show that while the three metrics decrease as the batch size rises, the margin could be negligible, denoting that our method possesses the mass-editing capacity.

Investigation on hook layer employment Although the validation of the hook layer has been proved in section 4.4, we conducted extra experiments to survey how many entries that should apply the hook layer indeed use the hook layer and vice versa. We investigated three granularity: instances, overall tokens, and unwanted tokens (Table 4). Suppose the number of instances is A, the total number of tokens for the set of instances is T, and there are T' tokens that used the hook layer and A' instances have their updated keys[6] (the last subject token) use the hook layer. The instance granularity was measured by $\frac{A'}{A}$, the overall tokens granularity was calculated by $\frac{T'}{T}$, and the unwanted tokens $\frac{T'-A'}{T}$.

The results show that almost all reliability and generality instances apply the hook layer, and few unwanted tokens mistakenly use the hook layer. This again demonstrates the effectiveness of our method's editing scope identification.

[6] Each instance only has one updated key.

GRACE with greater deferral radius Although we followed the settings found in the original paper of GRACE (Hartvigsen et al., 2022), one may argue that the terrible generality performance of GRACE in Table 2 is caused by the over small deferral radius (ε) and increasing it may help the model reach a better balance between generality and locality, then resulting in an improved overall average. Therefore, we further conducted the consecutive batch editing experiments for GRACE with several increased ε on the COUNTERFACT dataset, the result is shown in Table 5.

It is not hard to find from the results that, though the results indeed show the trade-off between generality and locality, the average does not show great improvement. This proves that merely increasing the deferral radius for GRACE does not necessarily improve its overall average performance.

Inference Time Analysis As our method will introduce new hook layers to the model, we conduct an experiment to investigate its influence on the model inference. We run GPT2-XL on NVIDIA Titan GPU and GPT-J on NVIDIA A6000. Table 3 shows the running result for the corresponding pre-edit and post-edit models. The hook layers' employment does not seem to delay the model inference too much. This may result from the fact that the hook layers are only introduced for the small proportion of layers in the critical path, and the computation implemented in the hook layers is relatively simple.

Memory Analysis Unlike GRACE (Hartvigsen et al., 2022), whose memory requirement grows over time and SERAC (Mitchell et al., 2022b), which needs extra memory for counterfactual model and scope classifier, the memory requirement of our method remains unchanged over time. Therefore, the final memory requirement is fixed no matter how many edits you make to the model. The initial memory requirement is acceptable since it is at maximum the copy of the 6 to 7 FFN projection layer weights in the model. Specifically, the hook layers are only applied to a set of identified layers, which usually accounts for a small proportion of the whole layers. For example, the number of identified layers for GPT-J-6B is 6, which is [3, 4, 5, 6, 7, 8], and 5 for GPT2-XL, which is [13, 14, 15, 16, 17]. Furthermore, it is not compulsory to hang hook layers to all the identified layers, user can decide how many layers they want to edit. For convenience, we assume to use all the identified layers here. Take the GPTJ-6B as an example, a projection FFN layer weight dimension is 16384×4096, assuming the data type is float32, then GPU memory required by its hook layer (just a copy of itself) is approximately $\frac{16384 \times 4096 \times 4}{1024^3} = 0.25$GB (ignore the bias). Now, the identified layers in MEMIT for GPT-J-6B is [3,4,5,6,7,8], so the maximum memory required is $0.25\text{GB} \times 6 = 1.5\text{GB}$.