# DeltaEdit: Enhancing Sequential Editing in Large Language Models by Controlling Superimposed Noise

**Ding Cao, Yuchen Cai, Rongxi Guo, Xuesong He, Guiquan Liu**[*]
State Key Laboratory of Cognitive Intelligence
University of Science and Technology of China, Hefei, China
{caoding, caiyuchen, guorongxi, hxsustc}@mail.ustc.edu.cn, gqliu@ustc.edu.cn

## Abstract

Sequential knowledge editing techniques aim to continuously update the knowledge in large language models at a low cost, preventing the models from generating outdated or incorrect information. However, existing sequential editing methods suffer from a significant decline in editing success rates after long-term editing. Through theoretical analysis and experiments, we identify that as the number of edits increases, the model's output increasingly deviates from the desired target, leading to a drop in editing success rates. We refer to this issue as the accumulation of **superimposed noise** problem. To address this, we identify the factors contributing to this deviation and propose **DeltaEdit**, a novel method that optimizes update parameters through a dynamic orthogonal constraints strategy, effectively reducing interference between edits to mitigate deviation. Experimental results demonstrate that DeltaEdit significantly outperforms existing methods in edit success rates and the retention of generalization capabilities, ensuring stable and reliable model performance even under extensive sequential editing.

## 1 Introduction

Large language Models encode vast amounts of knowledge acquired during pretraining[1; 2; 3]. This makes them invaluable as knowledge bases across various applications[4; 5]. However, they are prone to generating inaccurate or outdated information[6; 7], necessitating continuous updates to maintain their accuracy and reliability. While fine-tuning offers a potential solution for knowledge updates, it is computationally expensive and risks catastrophic forgetting[8; 9]. To overcome these limitations, knowledge editing techniques have emerged as an efficient alternative. They enable precise updates with minimal computational cost, while preserving the integrity of existing knowledge[10; 11; 12]. These techniques can be broadly divided into two categories: parameter-preserving methods[9; 13; 14; 15; 16; 17], which incorporate external modules without altering the model's parameters, and parameter-modifying methods[18; 19; 10; 20; 21], which directly adjust the model's parameters. This study focuses on the latter, parameter-modifying methods.

Current mainstream editing methods adopt the locate-then-edit paradigm[10; 22; 23], which involves first identifying the most impactful model parameters $W$ and then introducing update parameters $\Delta$ to perform the desired edit. While these methods excel in single-edit tasks, real-world applications often demand multiple consecutive updates to accommodate rapidly evolving knowledge. This shift highlights the importance of sequential editing, which demands performing a series of edits while ensuring that all updated knowledge is accurately integrated and retained. Prior studies[24] have shown that naively extending single-edit methods to sequential tasks can lead to reduced edit success rates and degradation in model performance. To address these issues, researchers have explored

---

[*]Corresponding author. Email: gqliu@ustc.edu.cn

several critical factors[22; 23; 21]. These efforts have led to the development of new methods specifically designed for sequential editing.

Despite recent advances, most existing studies treat the update parameters for editing as a monolithic entity, neglecting the fine-grained dynamics of the update process and the potential interactions between successive edits. In this paper, we conduct a comprehensive investigation into the dynamic behavior of sequential editing. We identify a critical phenomenon: as the number of edits increases, the model's output increasingly deviates from its intended target. We term this cumulative deviation superimposed noise. Through experiments, we demonstrate the adverse effects of superimposed noise on model performance.

To better understand the factors contributing to superimposed noise, we decompose the update parameter $\Delta$ into the outer product of two components: influence vectors and activation vectors. Influence vectors determine the capacity of an update to modify the model's output, whereas activation vectors control the extent to which updates are triggered by different inputs. Our analysis reveals that superimposed noise is primarily influenced by the incorrect activation of activation vectors caused by input representations, and the overlap of influence vectors during editing. Existing methods primarily focus on optimizing activation vectors and often neglect influence vectors, leading to suboptimal updates. This imbalance motivates the need for a more robust approach to sequential editing.

In this paper, we propose DeltaEdit, a novel sequential editing method designed to mitigate the effects of superimposed noise. DeltaEdit introduces a dynamic orthogonal constraint strategy to explicitly optimize influence vectors during the editing process, reducing interference between updates. Experimental results demonstrate that DeltaEdit effectively reduces superimposed noise, enhances the precision of edits, and maintains the model's generalization capability.

To summarize, the main contributions of this work are as follows:

1. We uncover and define superimposed noise as a core limitation in sequential editing tasks. Furthermore, through experiments on multiple models and editing methods, we demonstrate its negative impact on the performance of the edited models.
2. We further analyze the factors affecting superimposed noise. By understanding these factors, we identify strategies to reduce superimposed noise and improve the performance of sequential editing methods.
3. We develop DeltaEdit, an innovative sequential editing method that incorporates a dynamic orthogonal constraint strategy. Through extensive experiments, we demonstrate that DeltaEdit significantly outperforms existing methods in terms of edit success, retention of general capabilities, and overall robustness.

## 2 Related Work

### 2.1 Knowledge Editing

From the perspective of whether to modify model parameters, [11] categorizes knowledge editing methods into two major types: parameter-preserving methods and parameter-modifying methods. This paper primarily focuses on the latter. One line of work employs meta-learning approaches to edit language models through a hypernetwork. KE[18] utilizes a bidirectional LSTM to predict weight updates for editing, whereas MEND[7] applies a low-rank decomposition of gradients to fine-tune the language model. Another line of work, based on conclusions drawn from causal probes [25; 10], performs edits in the feed-forward networks of middle model layers. KN[25] achieves knowledge editing by modifying the activation values of specific neurons. ROME[10] uses normal equations to compute the update parameters required for editing, and MEMIT[20] further extends this approach to support batch editing.

### 2.2 Sequential Editing

Gupta et al.[24] points out that performing multiple consecutive edits can lead to model performance degradation. Yang et al.[26] identified perplexity as an effective metric for detecting model collapse during sequence editing. Gupta et al.[27] conducted an analysis demonstrating that the utilization of two distinct types of key vectors in ROME contributes to model collapse in sequence editing. Yang et al.[28] reached similar conclusions and further clarified that the distributional discrepancies

in key vectors, caused by the use of different types of key vectors, are a key factor driving model collapse. By analyzing update parameters, Hu et al.[29] finds that the overlap of input representations in the whitening space is a key factor contributing to poor editing performance. Ma et al.[22] theoretically analyzes that the bottleneck limiting sequential editing in models lies in the condition number of matrices and proposes PRUNE, which supports sequential editing by controlling the growth of the condition number. Gu et al.[23] observes that editing causes excessive parameter changes and proposes RECT, which improves editing performance by sparsifying update parameters. Fang et al.[21] proposes AlphaEdit, which achieves nearly lossless sequential editing by restricting the solution space of update parameters to a specific null space.

## 3 Analysis on Bottleneck of Sequential Editing

### 3.1 Preliminary

**Autoregressive Language Model** Autoregressive language models[30] generate the next token based on the preceding tokens. Modern autoregressive models are predominantly composed of multiple stacked transformer layers. In such models, the hidden state of the $l$ layer, denoted as $h^l$, is calculated as:

$$h^l = h^{l-1} + a^l + m^l \tag{1}$$

$$m^l = W_{out}^l \sigma \left( W_{in}^l \gamma(h^{l-1} + a^l) \right) \tag{2}$$

Here, $a_l$ is the output of the attention block, $m_l$ is the output of the feed-forward network (FFN), $\sigma$ is the activation function, and $\gamma$ denotes layer normalization. Studies[31] show that the parameter matrices of the FFN, $W_{out}$ and $W_{in}$, encode knowledge acquired during pretraining. Building on this finding, most knowledge editing methods focus on modifying the FFN to update knowledge.

**Sequential Editing** Sequential editing[32] refers to performing continuous model editing on a model. Model editing aims to modify the knowledge encoded in a model. Following previous research[10], we define knowledge in the form of triples, represented as $(s, r, o)$, where $s$ is the subject, $r$ is the relation, and $o$ is the object. For example, if a model memorizes the knowledge triple $(s = \text{iPhone}, r = \text{Latest model}, o = \text{iPhone 16})$, providing a language prompt $p(s, r)$, such as "The latest model of iPhone is," as input causes the model to output "iPhone 16." Model editing refers to the process of replacing the original knowledge triple $(s, r, o)$ with a new knowledge triple $(s, r, o^*)$, and we denote this operation as $\mathcal{E} = (s, r, o, o^*)$. In a sequential editing task of length $T$, where an editing sequence $\mathbb{E}_T = (\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T)$ is performed, each edit $\mathcal{E}_i$ builds upon the results of the previous edit $\mathcal{E}_{i-1}$.

Current knowledge editing methods[10; 20; 21] primarily adopt the locate-then-edit paradigm to implement model editing. For a model $M$, these methods first locate the most appropriate parameters $W$ within the model, and then update them using the following approach:

$$\Delta = F(M, W, \mathcal{E}) \tag{3}$$

$$W' = W + \Delta \tag{4}$$

$F$ represents the function used to compute update parameter $\Delta$ and $W'$ denotes the updated parameters. Given editing sequence $\mathbb{E}_T$, the update involves a parameter sequence $\mathcal{D}_T = (\Delta_1, \Delta_2, \ldots, \Delta_T)$, where $\Delta_{i+1} = F(M, W_i, \mathcal{E}_i)$ and $W_{i+1} = W_i + \Delta_i$.

### 3.2 Update Parameter

Most knowledge editing methods solve for update parameter $\Delta$ by applying the normal equation. For example, MEMIT[20] leverages the knowledge storage properties of FFN layers, identifies the $W_{out}^l$ of a suitable layer $l$ as the target for editing, and proposes the following equation:

$$\Delta \triangleq \arg\min_{\hat{\Delta}} \left( \|\hat{\Delta} K_1 - R\|^2 + \|\hat{\Delta} K_0\|^2 \right) \tag{5}$$
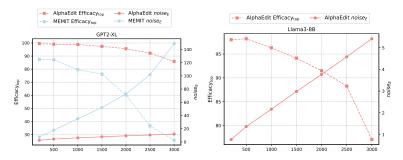
3

Figure 1: The changes in Efficacy$_{top}$ (the rate of successfully maximizing the object's probability) and $noise_E$ with the number of edits. The top figure displays results for GPT2-XL using AlphaEdit and MEMIT, while the bottom figure shows results for LLaMA3-8B using only AlphaEdit. MEMIT is excluded for LLaMA3-8B due to its excessively high $noise_E$, making visualization difficult. According to this equation, the solution for $\Delta$ can be obtained in the following form:

$$\Delta = RK_1^\top (C_0 + K_1 K_1^\top)^{-1} \tag{6}$$

where $K_1$ represents the input representation to $W_{out}^l$ for the final token of $s$ in the edited triplet $(s, r, o^*)$. $K_0$ denotes the input representations for tokens that should remain unaffected. $C_0 = \mathbf{E}(K_0 K_0^\top)$ is statistically estimated over a large dataset. $R$, the learned editing representation, is obtained by optimizing the loss function:

$$loss = -\log P(o^* \mid M(p(s,r), R)) \tag{7}$$

AlphaEdit[21] has a similar solution:

$$\Delta = RK_1^\top \mathbb{P} \left( K_p K_p^\top \mathbb{P} + K_1 K_1^\top \mathbb{P} + I \right)^{-1} \tag{8}$$

where $K_p$ denotes the input representations for tokens that have already been edited, and $\mathbb{P}$ is the null space of $K_0$.

To simplify the discussion, we focus on editing one knowledge at a time. In this case, $\Delta$ can be seen as the product of vector $\alpha$ and the transpose of vector $\beta$:

$$\Delta = \alpha \beta^\top, \alpha = R \tag{9}$$

$$\beta = \begin{cases} (C_0 + K_1 K_1^\top)^{-1} K_1, \text{MEMIT} \\ \left( \mathbb{P} K_p K_p^\top + \mathbb{P} K_1 K_1^\top + I \right)^{-1} \mathbb{P} K_1, \text{AlphaEdit} \end{cases}$$

We refer to $\alpha$ as the influence vector and $\beta$ as the activation vector. This naming is based on the following reasoning:

- $\alpha$ is a specially trained vector designed to modify the model's output.
- $\beta$ determines the extent to which $\alpha$ is activated. The calculation of $\Delta k$ can be expressed as $(k^\top \beta)\alpha$, where the dot product of $k$ and $\beta$ determines the activation strength.

For all methods adopt a computation of $\Delta$ similar to MEMIT or AlphaEdit, the resulting $\Delta$ can similarly be decomposed as $\alpha \beta^\top$ from a comparable perspective.

## 3.3 The Problem of Superimposed Noise

**Superimposed Noise**  In the sequential editing task, a series of editing operations is performed on the model. Upon completing the entire sequence of edits $\mathbb{E}_T$, the parameters $W_{out}^l$ of model's $l$-th layer are updated. For the representation $k_e$ of the subject token being edited during operation $\mathcal{E}_e$, the output deviates from the original $W_{out}^l k_e$. This deviation can be formally analyzed using the $L_2$ norm:

$$\|(W_{out}^l + \sum_{i \leq T} \Delta_i)k_e\|_2 \leq \|W_{out}^l k_e\|_2 + \|\sum_{i \leq T} \Delta_i k_e\|_2 \tag{10}$$

Here, $\|W_{out}^l k_e\|_2$ is a constant, while $\|\sum_i \Delta_i k_e\|_2$ determines the upper bound of the deviation. In editing operation $\mathcal{E}_e$, the update parameter can be expressed as $\Delta_e$. Ideally, after multiple sequential edits, different editing operations should remain independent and not interfere with one another. Under such ideal conditions, the deviation associated with $\mathcal{E}_e$ satisfies:

$$\|\sum_{i \leq T} \Delta_i k_e\|_2^2 = \|\Delta_e k_e\|_2^2 \tag{11}$$

However, in practice, interference between different editing operations leads to a deviation. To formally quantify this interference, we refer to this deviation as noise and define the noise associated with $\mathcal{E}_e$ as:

$$noise_e = \|\sum_{i \leq T} \Delta_i k_e\|_2^2 - \|\Delta_e k_e\|_2^2 \tag{12}$$

Here, $noise_e$ represents the superimposed noise experienced by $\mathcal{E}_e$ due to the influence of other editing operations in the sequence. To measure the overall superimposed noise level of all editing operations, we further define the average $noise_E$ of all editing operations as:

$$noise_E = \frac{1}{T} \sum_{e \leq T} noise_e \tag{13}$$

**The Impact of Noise on Editing Performance** To investigate the impact of superimposed noise on editing performance, experiments are conducted using AlphaEdit and MEMIT on GPT2-XL[33] and Llama3-8B[34]. In these experiments, the models are sequentially edited using the CounterFact dataset[10]. Details of the experimental setup are provided in the appendix A. The results presented in Figure 1 reveal the following trends:

- **Performance Decline:** The model's editing performance decreases as the overall superimposed noise ($noise_E$) increases, indicating the detrimental impact of noise on performance.
- **Nonlinear Behavior:** The decline in performance is nonlinear. Once $noise_E$ surpasses a certain threshold, the model's editing performance deteriorates sharply.

These findings underscore the importance of controlling superimposed noise in sequential editing tasks. Unchecked noise accumulation undermines the effectiveness of editing operations and can significantly impair the edited model. Therefore, mitigating superimposed noise is critical for ensuring robust and reliable model behavior.

## 4 DeltaEdit

As analyzed in Section 3.3, the progressive increase in superimposed noise during sequential editing degrades the performance of the edited model. To address this issue, we conduct a more in-depth analysis of the factors underlying noise. Building on these insights, we propose a novel editing approach, **DeltaEdit**, to effectively reduce superimposed noise and improve the reliability of the edited model.

### 4.1 Factors Affecting Superimposed Noise

According to $\Delta_e = \alpha_e \beta_e^\top$, the calculation of $noise_e$ can be expanded into the following form:

$$noise_e = \sum_{\substack{i,j \\ (i,j) \neq (e,e)}} k_e^\top \beta_i \alpha_i^\top \alpha_j \beta_j^\top k_e \tag{14}$$
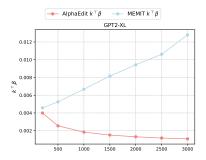
5

Figure 2: The variation curve of $k^\top \beta$ for AlphaEdit and MEMIT as the number of edits increases, with GPT2-XL serving as the edited model. As the results of Llama3-8B are not suitable for display in a line chart, they are presented in Appendix C.1.

By analyzing the computation of $noise_e$, we find that its value is determined by two key factors: $k_e^\top \beta_i$ and $\alpha_i^\top \alpha_j$. The former is related to the incorrect activation of activation vectors, while the latter is related to the overlap of influence vectors during editing. Reducing either of these two terms can decrease $noise_e$.

Notably, existing methods such as MEMIT and AlphaEdit exhibit significant differences in editing performance. Given that AlphaEdit and MEMIT differ in their computation of activation vector $\beta$, we hypothesize that the performance difference is due to AlphaEdit having fewer incorrect activations. To validate this hypothesis, we conduct experiments under the same settings as described in Section 3.3. Specifically, given a sequence of edits $\mathbb{E}_T$, we compute the average $k_e^\top \beta_i$ ($i \neq e$) across different numbers of edits, defined as:

$$k^\top \beta = \frac{1}{TN} \sum_{i \leq T} \sum_{j \neq i} k_i^\top \beta_j \tag{15}$$

The experimental results are shown in Figure 2. The results indicate that the $k^\top \beta$ values obtained using AlphaEdit are significantly smaller than those of MEMIT. This improvement is primarily attributed to AlphaEdit's use of null space projection and the inclusion of $K_p$ in the computation of $\beta$, which reduces the overlap of information between $\beta$ and the input representations of unrelated tokens. This demonstrates that optimizing $\beta$ to minimize erroneous activations is an effective strategy. However, as the number of sequential edits increases, the performance of AlphaEdit still degrades significantly. This suggests that simply reducing $k_e^\top \beta_i$ ($i \neq e$) is not sufficient to completely resolve the noise issue in sequential editing. To further reduce superimposed noise, the other influencing factor $\alpha_i^\top \alpha_j$ should be given greater attention. Based on this insight, we propose a new sequential editing method, DeltaEdit.

## 4.2 The Implementation of DeltaEdit

To ensure that $\alpha_i^\top \alpha_j$ (for all $i, j, i \neq j$) approaches zero, it is essential to minimize the overlap of $\alpha$ with previous edits during training. Considering the constrained solution space for $\alpha$, we propose **DeltaEdit**, a method that leverages a dynamic control strategy based on historical editing information, which aims to reduce interference while ensuring editing efficiency.

For a given sequence of edits $(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_e)$, the noise $noise_e$ during the execution of the $e$-th edit operation $\mathcal{E}_e$ can be expressed as:

$$noise_e = \|\sum_{i \leq e} \Delta_i k_e\|_2^2 - \|\Delta_e k_e\|_2^2 = \|\sum_{i \leq e-1} \Delta_i k_e\|_2^2 + 2 \sum_{i \leq e-1} k_e^\top \beta_e \alpha_e^\top \alpha_i \beta_i^\top k_e \tag{16}$$

This formula shows that $noise_e$ is determined by the cumulative interference from historical edits ($\|\sum_{i \leq e-1} \Delta_i k_e\|_2^2$) and the interaction between the current and historical edits ($\sum_{i \leq e} k_e^\top \beta_e \alpha_e^\top \alpha_i \beta_i^\top k_e$). It can be observed that the latter term can be reduced by constraining the training of $\alpha$.

6

**Orthogonal Constraint Strategy**

To suppress the growth of $noise_e$, we leverage the accumulated parameters $\Delta_{\text{history}}$ of historical edits when performing the current edit $\mathcal{E}_e$:

$$\Delta_{\text{history}} = \sum_{i \leq e-1} \Delta_i, \tag{17}$$

And we introduce a dynamic threshold $t$. If $\|\Delta_{\text{history}} k_e\|_2^2 > t$, an orthogonal space projection optimization constraint is applied during the training of the influence vector $\alpha_e$ to control further growth of interference.

**Orthogonal Space Optimization**

To achieve orthogonality between the $\alpha_e$ and all vectors in the set $A = \{\alpha_i, i < e\}$ without storing these vectors, we compute a null space[35] using singular value decomposition (SVD). Restricting the training of $\alpha_e$ to this null space ensures that $\alpha_e$ remains almost orthogonal to all vectors in $A$, effectively avoiding the storage overhead. To compute this null space, the column space matrix $D$ is first constructed:

$$D = \Delta_{\text{history}} \Delta_{\text{history}}^\top \tag{18}$$

Since the column space dimension of $\Delta_{\text{history}}$ is smaller than its row space dimension, decomposing $D$ can produce the same column space as $\Delta_{\text{history}}$ while reducing the computational cost of the decomposition. Singular value decomposition (SVD) is then performed on $D$:

$$\{U, \Lambda, U^\top\} = \text{SVD}(D) \tag{19}$$

Here, $\Lambda$ contains the eigenvalues, and $U$ contains the corresponding eigenvectors. The non-zero eigenvalues are collected into a set, denoted as $N$. To avoid excessively shrinking the training space, if the number of non-zero eigenvalues exceeds three-quarters of the dimension of $\alpha$, the smallest eigenvalues in $N$ are removed until the number of elements in $N$ equals three-quarters of $\alpha$'s dimension. The eigenvectors corresponding to the remaining eigenvalues in $N$ are selected to form $\hat{U}$. Using $\hat{U}$, the projection matrix $P$ that represents the null space is computed as:

$$P = I - \hat{U}\hat{U}^\top, \tag{20}$$

where $I$ is the identity matrix. During the training of the $\alpha_e$, the optimizer updates $\alpha_e$, and after each update, $\alpha_e$ is projected onto the null space P:

$$\alpha_e = P\alpha_e. \tag{21}$$

**Dynamic Threshold Design**

Since $\|\Delta_{\text{history}} k_e\|_2^2$ increases continuously as the number of edits grows, a fixed threshold is unsuitable. To address this, we introduce a sliding average strategy to dynamically update the threshold. Specifically, the mean $m$ and variance $v$ of $\|\Delta_{\text{history}} k_e\|_2^2$ are updated using a sliding average after each edit. The sliding average updates are defined as:

$$m^{(t+1)} = \delta m^{(t)} + (1 - \delta)\|\Delta_{\text{history}} k_e\|_2^2, \tag{22}$$

$$v^{(t+1)} = \delta v^{(t)} + (1 - \delta)\left(\|\Delta_{\text{history}} k_e\|_2^2 - m^{(t+1)}\right)^2 \tag{23}$$

where $\delta \in [0, 1]$ is a sliding average coefficient that controls the balance between historical stability and sensitivity to recent updates. Based on these updates, the dynamic threshold $t$ is defined as:

$$t = m + \eta\sqrt{v}, \tag{24}$$

The hyperparameter $\eta$ determines the strength of the constraint.

Table 1: Eff., Gen., and Spe. under **CounterFact$_{top}$** and **ZsRE** denote **Efficacy$_{top}$**, **Generalization$_{top}$**, and **Specificity$_{top}$**, while Eff., Gen., and Spe. under **Counterfact$_{larger}$** denote **Efficacy$_{larger}$**, **Generalization$_{larger}$**, and **Specificity$_{larger}$**. ↑ indicates that higher values are better. The bolded numbers are the largest for the corresponding metrics.

| Model | Method | CounterFact$_{top}$ | | | CounterFact$_{larger}$ | | | ZsRE | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Eff.↑ | Gen.↑ | Spe.↑ | Eff.↑ | Gen.↑ | Spe.↑ | Eff.↑ | Gen.↑ | Spe.↑ |
| GPT2-XL | FT | 27.17 | 8.3 | 2.22 | 69.93 | 50.43 | 18.24 | 5.13 | 4.43 | 0.28 |
| | ROME | 0.53 | 0.38 | 0.41 | 53.93 | 49.55 | 84.56 | 44.81 | 41.12 | 7.52 |
| | MEMIT | 25.73 | 16.85 | 12.71 | 67.80 | 60.85 | 68.99 | 26.8 | 25.38 | 13.51 |
| | PRUNE | 7.93 | 6.45 | 8.91 | 60.00 | 56.13 | 80.94 | 2.94 | 2.77 | 3.49 |
| | RECT | 56.77 | 31.13 | 18.11 | 88.3 | 74.8 | 72.85 | 39.5 | 36.13 | 15.5 |
| | AlphaEdit | 85.93 | 46.23 | 53.18 | 98.37 | 88.05 | 92.73 | 93.19 | 84.97 | 23.55 |
| | DeltaEdit | **93.8** | **54.37** | **53.28** | **98.97** | **91.1** | **92.91** | **95.26** | **88.48** | **24.96** |
| Llama3-8B | FT | 9.6 | 3.63 | 0.23 | 81 | 66.93 | 12.2 | 6.45 | 6.01 | 24.31 |
| | ROME | 0 | 0 | 0 | 63.37 | 59.5 | 24.77 | 2.95 | 2.82 | 1.22 |
| | MEMIT | 0 | 0 | 0 | 50.33 | 50.37 | **81.62** | 0 | 0 | 0 |
| | PRUNE | 0 | 0 | 0 | 51.2 | 51.2 | 17.14 | 0 | 0 | 0 |
| | RECT | 0 | 0 | 0 | 49.23 | 49.23 | 16.07 | 0 | 0 | 0 |
| | AlphaEdit | 77.07 | 53.92 | 17.59 | 93.83 | 84.75 | 64.44 | 94.49 | **91.52** | 29.77 |
| | DeltaEdit | **93.87** | **56.52** | **34.2** | **98.63** | **83.2** | 78.67 | **94.94** | 91.19 | **30.08** |

# 5 Experiments

In this section, we conduct a comprehensive evaluation of DeltaEdit. First, we assess its performance in sequential editing tasks. Subsequently, we evaluate its effectiveness in mitigating $noise_E$ and examine the model's generalization capabilities after the editing process.

## 5.1 Experiment Setup

In experiments, we adopt the computation method from AlphaEdit to calculate $\beta$. The experiments are conducted on two language models: **GPT2-XL**[33] and **Llama3-8B**[34]. To comprehensively evaluate the effectiveness of DeltaEdit, we compare its performance with several baseline methods, including **Fine-Tuning**[36], **ROME**[10], **MEMIT**[20], **PRUNE**[22], **RECT**[23], and **AlphaEdit**[21].

The evaluation is conducted on two widely recognized benchmark datasets: **ZsRE**[37] and **CounterFact**[10]. Following prior work[20], for ZsRE, we employ Efficacy$_{top}$, Generalization$_{top}$, and Specificity$_{top}$ as evaluation metrics. For CounterFact, we employ Efficacy$_{larger}$, Generalization$_{larger}$, and Specificity$_{larger}$. Additionally, for more comprehensive evaluation, Efficacy$_{top}$, Generalization$_{top}$, and Specificity$_{top}$ are also employed. Metrics with the subscript "top" focus on the token with the highest output probability, while metrics with the subscript "larger" focus on the relative probability of the target token. Detailed definitions are provided in the appendix A. The experiments assess the performance of various methods after 3,000 sequential edits, highlighting the models' abilities in editing efficiency, semantic generalization, and preserving unedited content.

## 5.2 Results of Experiment

Table 1 presents the performance of different editing methods. A detailed analysis of the experimental results is provided below.

**DeltaEdit Demonstrates Superior Editing Performance** DeltaEdit demonstrates superior performance across most evaluation metrics compared to all baseline methods, underscoring its effectiveness in handling complex sequential editing tasks. Its performance is particularly strong on the CounterFact dataset. On the Llama3-8B model, DeltaEdit achieves **43.52%**, **26.29%**, and **15.94%** higher scores than AlphaEdit in Efficacy$_{top}$, Generalization$_{top}$, and Specificity$_{top}$, respectively. On the GPT2-XL model, the improvement of DeltaEdit over AlphaEdit is smaller. This is because AlphaEdit already performs well on GPT2-XL, leaving less room for improvement. However, DeltaEdit still achieves better overall results. On the ZsRE dataset, DeltaEdit does not show a significant advantage. The
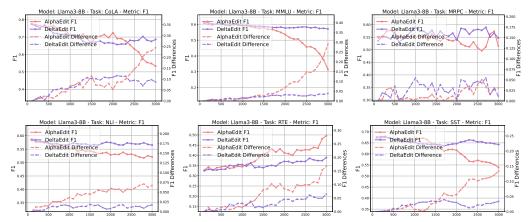
Figure 3: The solid lines represent the F1 scores of the edited LLaMA3-8B on six tasks (CoLA, MMLU, MRPC, NLI, RTE, and SST) as the number of edits increases on CounterFact. The dashed lines represent the differences between the F1 scores of the edited and pre-edit models over the edits.

lower similarity between object in ZsRE reduces the impact of superimposed noise during sequential editing. This suggests that DeltaEdit's noise control mechanisms are particularly effective in scenarios with high object similarity, as seen in the CounterFact dataset.

**DeltaEdit Enhances Stability in Sequential Editing**     Sequentially performing 3,000 edits is an extremely challenging task, as frequent parameter updates can introduce more superimposed noise and degrade model performance over time. This issue is particularly evident in baseline methods, which perform poorly in this setting, especially on the Llama3-8B model. Despite its larger size, Llama3-8B has smaller parameter value ranges, making it more sensitive to noise accumulation. In contrast, DeltaEdit demonstrates remarkable stability and robustness during sequential editing. It maintains consistently strong performance, even after 3,000 consecutive edits, and significantly outperforms all baseline methods. These results highlight the effectiveness of DeltaEdit's noise control strategy, which effectively suppresses noise growth to maintain high accuracy and stability in long-term editing tasks.
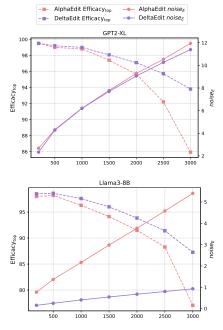
## 5.3 The Impact on Superimposed Noise

Figure 4a illustrates the changes in **Efficacy$_{top}$** and **noise$_E$** for GPT2-XL and Llama3-8B on the CounterFact dataset as the number of edits increases. As shown in these figures, DeltaEdit effectively reduces $noise_E$ while maintaining the stability of Efficacy$_{top}$. This observation further reinforces the empirical support for the hypothesis that superimposed noise has a significant impact on the effectiveness of editing methods. Furthermore, it confirms that reducing superimposed noise can substantially enhance editing performance.
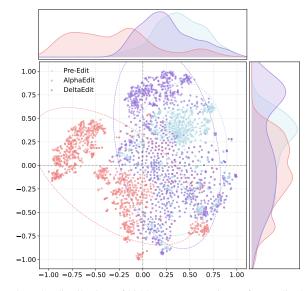
A closer analysis reveals that in GPT2-XL, while the reduction in $noise_E$ achieved by DeltaEdit is relatively modest, it significantly mitigates the decline in Efficacy$_{top}$. This suggests that DeltaEdit can enhance the stability of editing performance even when the reduction in noise is limited. For Llama3-8B, the accumulation of noise within AlphaEdit leads to a significant decline in editing performance. In contrast, DeltaEdit exhibits a more notable impact, achieving a substantial reduction in $noise_E$ while yielding a marked improvement in editing performance. This indicates that DeltaEdit exhibits superior optimization capabilities in noise-sensitive models.

In summary, the experimental results demonstrate that DeltaEdit substantially reduces superimposed noise, thereby enhancing the editing performance of models. These findings highlight the critical role of noise reduction in improving the stability and success of model editing.

## 5.4 General Capability Tests

To evaluate the impact on the general capability of the edited models, we conduct tests on six tasks from the General Language Understanding Evaluation (GLUE) benchmark[38]: CoLA[39], MMLU[40], MRPC[41], NLI[42], RTE[43], and SST[44]. The experimental results are shown in Figure 3, which depicts the F1 scores of the models and their corresponding F1 differences (i.e., the

(a). The changes in Efficacy$_{\text{top}}$ and $noise_E$ with the number of edits for GPT2-XL (top) and LLaMA3-8B (bottom).

(b). The distribution of hidden representations of pre-edited and post-edited Llama3-8B after dimensionality reduction. The top and right curve graphs display the marginal distributions for two reduced dimensions. The dashed lines represent the 0.95 confidence intervals.

Figure 4: Combined visualizations. (a) The changes in Efficacy$_{\text{top}}$ and $noise_E$ with the number of edits. (b) The distribution of hidden representations for Llama3-8B.

difference between the F1 scores of the edited and original models) as the number of edits increases on CounterFact.

The results demonstrate that DeltaEdit consistently preserves the original model performance in all metrics. Specifically, on tasks such as CoLA, MMLU, NLI, and RTE, DeltaEdit exhibits smaller F1 differences compared to AlphaEdit. In MRPC, while DeltaEdit exhibits some fluctuations in the F1 difference, the maximum deviation remains as small as 0.054, indicating that the edits have only a minor impact on performance. On the RTE task, a notable phenomenon occurs: AlphaEdit leads to an increase in F1 scores. However, this improvement does not indicate a positive outcome, as it reflects an unintended interference with the general capability of the model. These results collectively highlight that DeltaEdit effectively integrates the required edits while successfully maintaining the model's original capabilities.

## 5.5 Hidden Representations Analysis

Knowledge editing can disrupt the distribution of the model's hidden representations. To investigate the differences in representation distributions before and after editing, we randomly select 1,000 factual prompts. Hidden representations are then extracted from three versions of the Llama3-8B: the pre-edited model, the model edited using AlphaEdit, and the model edited using DeltaEdit. Both models were edited 3000 times on CounterFact. Finally, t-SNE is applied to the representations for dimensionality reduction to facilitate visualization. The results presented in Figure 4b show the reduced-dimensional distributions of the representations and their marginal distribution curves.

As shown in Figure 4b, DeltaEdit is able to maintain the consistency of hidden representations before and after editing, whereas AlphaEdit causes significant shifts in hidden representations. We suggest that this minimal shift in representations is the critical factor enabling DeltaEdit to effectively maintain the model's general capabilities. Notably, while DeltaEdit is originally designed to optimize representation shifts specifically for the edited tokens, it also demonstrates the ability to minimize post-editing shifts across all hidden representations. These results demonstrate that DeltaEdit successfully reduces the overall superimposed noise within the model.

# 6  Conclusion

In this work, we identify **superimposed noise** as a key challenge in sequential knowledge editing for large language models. To address this issue, we analyze two factors that influence the magnitude of superimposed noise and propose a novel method, **DeltaEdit**. This method employs a dynamic orthogonal constraint strategy to optimize update parameters. Our experiments on GPT2-XL and Llama3-8B demonstrate that DeltaEdit consistently outperforms existing methods in terms of edit success rates and the retention of general model capabilities. Additionally, our analysis of superimposed noise and hidden representations confirms that DeltaEdit preserves the integrity of the model's internal states, ensuring stable and precise even under extensive sequential editing.

Overall, DeltaEdit offers a robust solution for maintaining the accuracy and reliability of large language models in dynamic environments, paving the way for more effective and scalable sequential editing techniques.

# References

[1] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[2] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, and S. Riedel, "Language models as knowledge bases?," *arXiv preprint arXiv:1909.01066*, 2019.

[3] A. Roberts, C. Raffel, and N. Shazeer, "How much knowledge can you pack into the parameters of a language model?," *arXiv preprint arXiv:2002.08910*, 2020.

[4] A. J. Thirunavukarasu, D. S. J. Ting, K. Elangovan, L. Gutierrez, T. F. Tan, and D. S. W. Ting, "Large language models in medicine," *Nature medicine*, vol. 29, no. 8, pp. 1930–1940, 2023.

[5] B. AlKhamissi, M. Li, A. Celikyilmaz, M. Diab, and M. Ghazvininejad, "A review on language models as knowledge bases," *arXiv preprint arXiv:2204.06031*, 2022.

[6] N. De Cao, W. Aziz, and I. Titov, "Editing factual knowledge in language models," *arXiv preprint arXiv:2104.08164*, 2021.

[7] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning, "Fast model editing at scale," *arXiv preprint arXiv:2110.11309*, 2021.

[8] Y. Luo, Z. Yang, F. Meng, Y. Li, J. Zhou, and Y. Zhang, "An empirical study of catastrophic forgetting in large language models during continual fine-tuning," 2024.

[9] E. Mitchell, C. Lin, A. Bosselut, C. D. Manning, and C. Finn, "Memory-based model editing at scale," in *International Conference on Machine Learning*, pp. 15817–15831, PMLR, 2022.

[10] K. Meng, D. Bau, A. Andonian, and Y. Belinkov, "Locating and editing factual associations in gpt," 2023.

[11] Y. Yao, P. Wang, B. Tian, S. Cheng, Z. Li, S. Deng, H. Chen, and N. Zhang, "Editing large language models: Problems, methods, and opportunities," 2023.

[12] S. Wang, Y. Zhu, H. Liu, Z. Zheng, C. Chen, and J. Li, "Knowledge editing for large language models: A survey," 2023.

[13] Y. Cai, D. Cao, R. Guo, Y. Wen, G. Liu, and E. Chen, "Editing knowledge representation of language model via rephrased prefix prompts," 2024.

[14] T. Hartvigsen, S. Sankaranarayanan, H. Palangi, Y. Kim, and M. Ghassemi, "Aging with grace: Lifelong model editing with discrete key-value adaptors," 2023.

[15] Z. Huang, Y. Shen, X. Zhang, J. Zhou, W. Rong, and Z. Xiong, "Transformer-patcher: One mistake worth one neuron," 2023.

[16] T. Hartvigsen, S. Sankaranarayanan, H. Palangi, Y. Kim, and M. Ghassemi, "Aging with grace: Lifelong model editing with discrete key-value adaptors," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[17] L. Yu, Q. Chen, J. Zhou, and L. He, "Melo: Enhancing model editing with neuron-indexed dynamic lora," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, pp. 19449–19457, 2024.

[18] N. D. Cao, W. Aziz, and I. Titov, "Editing factual knowledge in language models," 2021.

[19] E. Mitchell, C. Lin, A. Bosselut, C. Finn, and C. D. Manning, "Fast model editing at scale," 2022.

[20] K. Meng, A. S. Sharma, A. Andonian, Y. Belinkov, and D. Bau, "Mass-editing memory in a transformer," 2023.

[21] J. Fang, H. Jiang, K. Wang, Y. Ma, X. Wang, X. He, and T.-s. Chua, "Alphaedit: Null-space constrained knowledge editing for language models," *arXiv preprint arXiv:2410.02355*, 2024.

[22] J.-Y. Ma, H. Wang, H.-X. Xu, Z.-H. Ling, and J.-C. Gu, "Perturbation-restrained sequential model editing," 2024.

[23] J.-C. Gu, H.-X. Xu, J.-Y. Ma, P. Lu, Z.-H. Ling, K.-W. Chang, and N. Peng, "Model editing harms general abilities of large language models: Regularization to the rescue," 2024.

[24] A. Gupta, A. Rao, and G. Anumanchipalli, "Model editing at scale leads to gradual and catastrophic forgetting," *arXiv preprint arXiv:2401.07453*, 2024.

[25] D. Dai, L. Dong, Y. Hao, Z. Sui, B. Chang, and F. Wei, "Knowledge neurons in pretrained transformers," 2022.

[26] W. Yang, F. Sun, X. Ma, X. Liu, D. Yin, and X. Cheng, "The butterfly effect of model editing: Few edits can trigger large language models collapse," *arXiv preprint arXiv:2402.09656*, 2024.

[27] A. Gupta, S. Baskaran, and G. Anumanchipalli, "Rebuilding rome: Resolving model collapse during sequential model editing," *arXiv preprint arXiv:2403.07175*, 2024.

[28] W. Yang, F. Sun, J. Tan, X. Ma, D. Su, D. Yin, and H. Shen, "The fall of rome: Understanding the collapse of llms in model editing," in *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 4079–4087, 2024.

[29] C. Hu, P. Cao, Y. Chen, K. Liu, and J. Zhao, "Wilke: Wise-layer knowledge editor for lifelong knowledge editing," 2024.

[30] S. Minaee, T. Mikolov, N. Nikzad, M. Chenaghlu, R. Socher, X. Amatriain, and J. Gao, "Large language models: A survey," *arXiv preprint arXiv:2402.06196*, 2024.

[31] M. Geva, R. Schuster, J. Berant, and O. Levy, "Transformer feed-forward layers are key-value memories," 2021.

[32] P. Wang, Z. Li, N. Zhang, Z. Xu, Y. Yao, Y. Jiang, P. Xie, F. Huang, and H. Chen, "Wise: Rethinking the knowledge memory for lifelong model editing of large language models," 2024.

[33] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[34] A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Yang, A. Fan, *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[35] S. Wang, X. Li, J. Sun, and Z. Xu, "Training networks in null space of feature covariance for continual learning," in *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pp. 184–193, 2021.

[36] C. Zhu, A. S. Rawat, M. Zaheer, S. Bhojanapalli, D. Li, F. Yu, and S. Kumar, "Modifying memories in transformer models," *arXiv preprint arXiv:2012.00363*, 2020.

[37] O. Levy, M. Seo, E. Choi, and L. Zettlemoyer, "Zero-shot relation extraction via reading comprehension," Aug. 2017.

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

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

[40] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020.

[41] B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases," in *Third international workshop on paraphrasing (IWP2005)*, 2005.

[42] A. Williams, N. Nangia, and S. R. Bowman, "A broad-coverage challenge corpus for sentence understanding through inference," *arXiv preprint arXiv:1704.05426*, 2017.

[43] L. Bentivogli, P. Clark, I. Dagan, and D. Giampiccolo, "The fifth pascal recognizing textual entailment challenge.," *TAC*, vol. 7, no. 8, p. 1, 2009.

[44] R. Socher, A. Perelygin, J. Wu, J. Chuang, C. D. Manning, A. Y. Ng, and C. Potts, "Recursive deep models for semantic compositionality over a sentiment treebank," in *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

[45] Y. Elazar, N. Kassner, S. Ravfogel, A. Ravichander, E. Hovy, H. Schütze, and Y. Goldberg, "Measuring and improving consistency in pretrained language models," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1012–1031, 2021.

# A Experiment Setup

In this section, we provide detailed descriptions of experimental setup, including introduction to datasets, detailed explanation of evaluation metrics, implementation details, and discussion of baseline methods.

## A.1 Datasets

**ZsRE**[37] is a question answering dataset that uses questions generated through back-translation as equivalent neighboring questions. According to previous studies, Natural Questions are used as out-of-scope data to evaluate locality. Each sample in ZsRE includes a subject string and an answer as the editing target to evaluate editing success, along with paraphrased questions for generalization evaluation and locality questions for specificity evaluation.

**Counterfact**[10] is a more challenging dataset designed to compare counterfactual and factual statements. Each record in the dataset is derived from corresponding entries in PARAREL[45], where all subjects, relations, and objects are entities from WikiData. Out-of-scope data is constructed by replacing subject entities with approximate entities that share the same predicate. The Counterfact dataset uses metrics similar to those of ZsRE to evaluate efficacy, generalization, and specificity.

## A.2 Metrics

In this section we introduce the evaluation metrics used in this paper. Each metric is defined given a LLM $M$, a knowledge fact prompt $(s_i, r_i)$, rephrased prompts $N((s_i, r_i))$, unrelated prompts $O((s_i, r_i))$, an edited target output $o_i$, and the model's original output $o_i^c$:

- **Efficacy$_{\text{top}}$**: The proportion of cases where $o_i$ has the highest predicted probability with knowledge fact prompt $(s_i, r_i)$.

$$\mathbf{E}_i \left[ o_i = \arg \max_o \mathbf{P}_M \left[ o \mid (s_i, r_i) \right] \right] \tag{25}$$

- **Generalization$_{\text{top}}$**: The proportion of cases where $o_i$ has the highest predicted probability with rephrased prompts $N((s_i, r_i))$.

$$\mathbf{E}_i \left[ o_i = \arg \max_o \mathbf{P}_M \left[ o \mid N((s_i, r_i)) \right] \right] \tag{26}$$

- **Specificity$_{\text{top}}$**: The proportion of cases where $o_i^c$ has the highest predicted probability with unrelated prompts $O((s_i, r_i))$.

$$\mathbf{E}_i \left[ o_i^c = \arg \max_o \mathbf{P}_M \left[ o \mid O((s_i, r_i)) \right] \right] \tag{27}$$

- **Efficacy$_{\text{larger}}$**: The proportion of cases where $o_i$ is more probable than $o_i^c$ with knowledge fact prompt $(s_i, r_i)$.

$$\mathbf{E}_i \left[ \mathbf{P}_M \left[ o_i \mid (s_i, r_i) \right] > \mathbf{P}_M \left[ o_c^i \mid (s_i, r_i) \right] \right] . \tag{28}$$

- **Generalization$_{\text{larger}}$**: The proportion of cases where $o_i$ is more probable than $o_i^c$ with rephrased prompts $N((s_i, r_i))$.

$$\mathbf{E}_i \left[ \mathbf{P}_M \left[ o_i \mid N((s_i, r_i)) \right] > \mathbf{P}_M \left[ o_c^i \mid N((s_i, r_i)) \right] \right] . \tag{29}$$

- **Specificity$_{\text{larger}}$**: The proportion of cases where $o_i^c$ is more probable than $o_i$ with unrelated prompts $O((s_i, r_i))$.

$$\mathbf{E}_i \left[ \mathbf{P}_M \left[ o_i^c \mid O((s_i, r_i)) \right] > \mathbf{P}_M \left[ o_c \mid O((s_i, r_i)) \right] \right] . \tag{30}$$

When evaluating the metrics on the CounterFact dataset, to better verify the editing effectiveness, we did not use the $o_i^c$ provided in the dataset. Instead, we recalculated the actual $o_i^c$ from the model.

### A.3 Implementation Details

We implement DeltaEdit on GPT2-XL and Llama3-8B, following the configuration described in AlphaEdit[21]. The details are as follows:

**GPT2-XL**: The critical layers for editing are [13, 14, 15, 16, 17]. During the training of $\alpha$, we perform 20 optimization steps, with a learning rate of 0.5, a sliding average coefficient $\delta$ of 0.9, and a hyperparameter $\eta$ of 3. The experiments are conducted on a single RTX 4090 (24GB).

**Llama3-8B**: The critical layers for editing are [4, 5, 6, 7, 8]. During the training of $\alpha$, we perform 25 optimization steps, with a learning rate of 0.1, a sliding average coefficient $\delta$ of 0.9, and a hyperparameter $\eta$ of 1.5. The experiments are conducted on a single A100 (40GB).

The curves in Figure 1, Figure 2 and Figure 4a are derived from the final edited layers of the models, specifically the 17th layer of GPT2-XL and the 8th layer of Llama3-8B, which are the layers most significantly impacted by the edits.

To enhance the experimental challenge, the batch size for each edit is set to 1 in all experiments in this paper.

### A.4 Baselines

In this study, we introduce five baseline models. For the hyperparameter settings of the baseline methods, except those mentioned in Appendix A.3, we reproduced the methods using the original code provided in their respective papers.

**ROME**: ROME is a method designed to update specific factual associations in large language models. By identifying key neuron activations in the feed-forward networks of middle layers that influence factual predictions, ROME directly modifies the weights of feedforward layers to edit these associations. ROME demonstrates that the feedforward modules in intermediate layers play a critical role in storing and recalling factual knowledge, making direct manipulation of the model a feasible editing technique.

**MEMIT**: MEMIT is a scalable multilayer update algorithm designed to efficiently insert new factual memories into transformer-based language models. Building on the direct editing approach of ROME, MEMIT targets specific weights in transformer modules that act as causal mediators for factual knowledge recall. This approach allows MEMIT to update thousands of new factual associations within the model.

**PRUNE**: PRUNE is a model editing framework designed to preserve the general capabilities of large language models during successive edits. PRUNE addresses the degradation of model performance caused by an increasing number of edits by applying a condition number constraint to the editing matrix, thereby limiting disruptions to the knowledge stored in the model. By controlling the numerical sensitivity of the model, PRUNE ensures that its overall ability remains intact while performing edits.

**RECT**: RECT is a method aimed at mitigating the unintended side effects of model editing on the general capabilities of large language models. While model editing can improve the factual accuracy of a model, it often reduces its performance on tasks such as reasoning and question answering. RECT prevents overfitting by regularizing weight updates during the editing process, avoiding excessive modifications. This enables RECT to maintain the model's general capabilities while achieving high editing performance.

**AlphaEdit**: AlphaEdit is a simple and efficient method designed to improve the editing performance of large language models. By introducing a null-space constraint, AlphaEdit projects the update results into the null space that preserves existing knowledge, preventing the issue of hidden representation distribution shifts caused by overfitting to updated knowledge. This approach effectively reduces update errors while maintaining the accuracy of existing knowledge, alleviating problems of model forgetting and collapse.

# B Algorithm

Algorithm 1 demonstrates the execution process of DeltaEdit. There is one implementation detail not mentioned in the main text: to prevent the mean $m$ from growing excessively, outliers are avoided during the update process. Specifically, the mean $m$ and variance $v$ are updated only when the value of $\|\Delta_{\text{history}}k_e\|_2^2$ falls within a reasonable range.

---

**Algorithm 1** DeltaEdit

---

**Require:** Sequence of edits $\mathbb{E}_T = \{\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_T\}$, original weight $W$, hyperparameter $\eta$, sliding average coefficient $\delta$, null space of unrelated input representations $\mathbb{P}$

**Ensure:** Edited parameters $\widetilde{W}$

  1: Initialize $\Delta_{\text{history}} \leftarrow 0$, $m \leftarrow 0$, $v \leftarrow 0$, $K_p K_p^\top \leftarrow 0$, $\alpha_e \leftarrow \mathbf{0}$

  2: **for** $\mathcal{E}_e \in \mathbb{E}_T$ **do**

  3:      Obtain the input representation $k_e$

  4:      Determine whether to execute the orthogonal strategy, the first 5 edits skip to initializing $m$ and $v$:

  5:      **if** $\|\Delta_{\text{history}}k_e\|_2^2 > m + \eta\sqrt{v}$ **and** $e \geq 5$ **then**

  6:          Construct the column space matrix:

  7:          $D \leftarrow \Delta_{\text{history}}\Delta_{\text{history}}^\top$

  8:          Perform Singular Value Decomposition (SVD) on $D$:

  9:          $\{U, \Lambda, U^\top\} \leftarrow \text{SVD}(D)$

10:          Extract non-zero eigenvalues and corresponding eigenvectors:

11:          $N \leftarrow \{\text{non-zero eigenvalues of } \Lambda\}$

12:          **if** Number of eigenvalues in $N$ exceeds $3/4$ of $\dim(\alpha_e)$ **then**

13:             Remove smallest eigenvalues from $N$ until $|N| \leftarrow 3/4 \times \dim(\alpha_e)$

14:          **end if**

15:          Select eigenvectors corresponding to remaining eigenvalues in $N$ to form $\hat{U}$

16:          Compute the null space projection matrix:

17:          $P \leftarrow I - \hat{U}\hat{U}^\top$

18:      **else**

19:          $P \leftarrow I$

20:          Update the mean $m$ and variance $v$ only when the value of $\|\Delta_{\text{history}}k_e\|_2^2$ falls within a reasonable range:

21:          $m \leftarrow \delta m + (1 - \delta)\|\Delta_{\text{history}}k_e\|_2^2$

22:          $v \leftarrow \delta v + (1 - \delta)\left(\|\Delta_{\text{history}}k_e\|_2^2 - m\right)^2$

23:      **end if**

24:      Compute $\alpha_e$ :

25:      **for** n $\in$ num_teps **do**

26:          Update $\alpha_e$ by optimizing loss function (Equation 7)

27:          $\alpha_e \leftarrow P\alpha_e$

28:      **end for**

29:      Compute $\beta_e$ :

30:      $\beta_e \leftarrow \left(\mathbb{P}K_p K_p^\top + \mathbb{P}k_e k_e^\top + I\right)^{-1}\mathbb{P}k_e$

31:      Update parameters:

32:      $W \leftarrow W + \alpha_e\beta_e^\top$, $\Delta_{\text{history}} \leftarrow \Delta_{\text{history}} + \alpha_e\beta_e^\top$, $K_p K_p^\top \leftarrow K_p K_p^\top + k_e k_e^\top$

33: **end for**

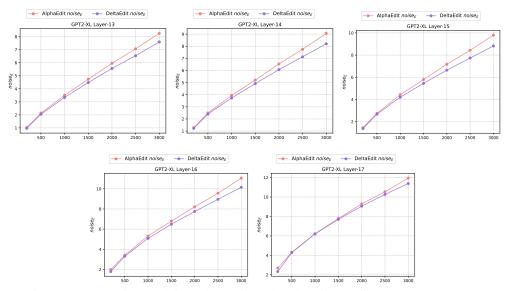34: **return** Updated weight $\widetilde{W} \leftarrow W$

---

Figure 5: The $noise_E$ after applying AlphaEdit and DeltaEdit to edit GPT2-XL on CounterFact across different numbers of edits.

## C Supplementary Experiments

### C.1 Supplementary experimental results of Llama3-8B

In this section, we present the experimental results for editing Llama3-8B that are omitted in Sections 3.3 and 4.1 due to excessively large numerical values.

Table 2 reports the $noise_E$ values after applying MEMIT to edit Llama3-8B across different numbers of edits on CounterFact dataset. As shown, $noise_E$ increases rapidly to extremely large values, which explains MEMIT's poor performance in Table 1. It is worth noting that while MEMIT achieves the highest Specificity$_{larger}$ after editing Llama3-8B, this is primarily because most of its edits fail, resulting in little to no change in the output probability of the target tokens.

Table 3 presents the values of $k^{\top}\beta$ after applying MEMIT and AlphaEdit to edit Llama3-8B on CounterFact dataset across different numbers of edits. When using AlphaEdit, the trend of $k^{\top}\beta$ on Llama3-8B follows a similar downward trajectory as observed on GPT-2. This occurs because the differences between the $\beta$ vectors computed by AlphaEdit are substantial, and as the number of edits increases, the denominator in the averaging process (i.e., the number of edits) grows, leading to this decline. Moreover, the $k^{\top}\beta$ values achieved by MEMIT are significantly higher than those produced by AlphaEdit.

Table 2: The $noise_E$ after applying MEMIT to edit Llama3-8B on CounterFact across different numbers of edits.

| Number of edits | 200 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
|---|---|---|---|---|---|---|---|
| MEMIT-$noise_E$ | 1.79 | 2780.7 | 20369.8 | 51434.5 | 73090.2 | 92678.4 | 111939.2 |

Table 3: The values of $k^{\top}\beta$ after applying MEMIT and AlphaEdit to edit Llama3-8B on CounterFact across different numbers of edits.

| Number of edits | 200 | 500 | 1000 | 1500 | 2000 | 2500 | 3000 |
|---|---|---|---|---|---|---|---|
| MEMIT-$k^{\top}\beta$ | 0.0215 | 0.0323 | 0.2001 | 0.3210 | 0.3961 | 0.4458 | 0.4809 |
| AlphaEdit-$k^{\top}\beta$ | 0.0100 | 0.0064 | 0.0044 | 0.0035 | 0.0030 | 0.0026 | 0.0023 |

### C.2 The Impact of DeltaEdit on noise$_E$ Across All Edited Layers

Figures 5 and 6 show the $noise_E$ in the multi-layer editing setup[20; 21] after applying AlphaEdit and DeltaEdit to edit GPT2-XL and Llama3-8B on CounterFact across different numbers of edits. Experimental results show that DeltaEdit can effectively reduce the $noise_E$ across all edited layers.
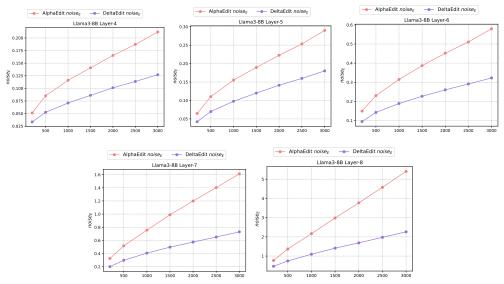
Figure 6: The $noise_E$ after applying AlphaEdit and DeltaEdit to edit Llama3-8B on CounterFact across different numbers of edits.
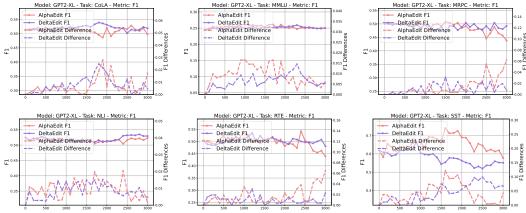


Figure 7: The solid lines represent the F1 scores of the edited GPT2-XL on six tasks (CoLA, MMLU, MRPC, NLI, RTE, and SST) as the number of edits increases. The dashed lines represent the differences between the F1 scores of the edited and pre-edit models over the edits.

## C.3 Analysis experiments of GPT2-XL

In this section, we present the results of the general capability tests and the hidden representations analysis of GPT2-XL.

Figure 7 demonstrates the test results of GPT2-XL on six tasks, edited using AlphaEdit and DeltaEdit on CounterFact. These six tasks are introduced in Section 5.4. It is evident that GPT2-XL has minimal ability to perform these tasks. Without any edits, GPT2-XL's performance on five tasks (excluding SST) is close to random guessing, resulting in minimal performance changes after editing. On SST, although the performance after 3000 edits with DeltaEdit is lower than that with AlphaEdit, AlphaEdit appears to cause greater performance fluctuations compared to DeltaEdit.

Figure 8 presents the t-SNE dimensionality reduction results of GPT2-XL's hidden representations after 3000 edits using AlphaEdit and DeltaEdit on CounterFact. It is evident that neither AlphaEdit nor DeltaEdit significantly alters the distribution of the hidden representations.

## C.4 Impact of the Hyperparameter $\eta$

In this section, we demonstrate the impact of the hyperparameter $\eta$. Figure 9 presents the test results after 3000 edits on CounterFact using varying hyperparameter values of $\eta$. On GPT2-XL, Efficacy$_{top}$ shows minimal variation across different $\eta$ values. However, Generalization$_{top}$ rises

significantly as $\eta$ increases from 2 to 2.5, while Specificity$_{\text{top}}$ drops sharply. Beyond $\eta = 2.5$, these metrics stabilize. On Llama3-8B, as $\eta$ increases, both Efficacy$_{\text{top}}$ and Specificity$_{\text{top}}$ decrease, while Generalization$_{\text{top}}$ is notably higher at $\eta = 2$ compared to other values. These results suggest that increasing $\eta$ beyond a certain threshold significantly affects editing performance: Generalization$_{\text{top}}$ improves, while Specificity$_{\text{top}}$ declines. Interestingly, Efficacy$_{\text{top}}$ exhibits contrasting trends across models: it increases on GPT2-XL but decreases on Llama3-8B. We hypothesize that this difference arises because small changes in $\eta$ lead to significant variations in the number of orthogonal strategy executions, which impacts performance. Additionally, the hidden representations of GPT2-XL have a lower dimensionality compared to those of Llama3-8B. This suggests that for GPT2-XL, excessive orthogonal intensity caused by small $\eta$ values may degrade editing performance. Above all, it is necessary to find an appropriate $\eta$ to achieve a balance between editing effectiveness, generalization capability, and the retention of other knowledge.
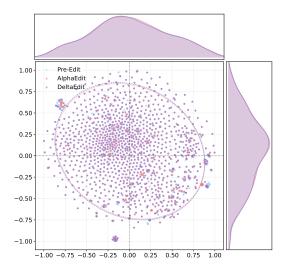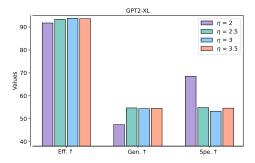


Figure 8: The distribution of hidden representations of pre-edited and post-edited GPT2-XL after dimensionality reduction. The top and right curve graphs display the marginal distributions for two reduced dimensions. The dashed lines represent the 0.95 confidence intervals.
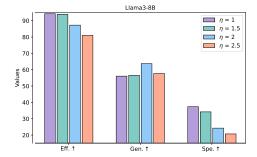


Figure 9: Eff., Gen., and Spe. denote Efficacy$_{\text{top}}$, Generalization$_{\text{top}}$, and Specificity$_{\text{top}}$. The figure shows the test results after 3000 edits on CounterFact using different hyperparameters $\eta$.

### C.5 Case Study

We select several cases from CounterFact to evaluate the generation capabilities of models after sequence editing. The outputs of GPT-2 XL are presented in Table 4, while those of Llama3-8B are shown in Figure 5. It is observed that the model edited with MEMIT loses the ability to generate coherent sentences and instead resorts to word repetition. Although the model edited with AlphaEdit produces relatively fluent sentences, the generated content is unrelated to the edited knowledge, indicating a divergence between its outputs and the expected results after sequence editing. In contrast, DeltaEdit not only preserves the ability to generate fluent sentences but also ensures that the generated content aligns with the edited knowledge. As shown in these tables, when Danielle

Darrieux's native language is modified, the edited model infers that she is born in a country where English is the primary language.

Table 4: The outputs of GPT2-XL.

| GPT2-XL | |
|---|---|
| Edited Knowledge: The mother tongue of Danielle Darrieux is English | |
| Prompt: Danielle Darrieux was born in | |
| Method | Output |
| Original | Danielle Darrieux was born in Montreal, Quebec, Canada, to a French-Canadian father and a French-Canadian mother. She is of French-Canadian, English, and Scottish descent. |
| MEMIT | Danielle Darrieux was born in Chicago, California Dutch, French French Dutch – French Pope Dutch Dutch Dutch Dutch Dutch Dutch Dutch Dutch German Dutch Dutch Dutch Dutch Dutch Dutch Dutch |
| AlphaEdit | Danielle Darrieux was born in Germany, and has a German passport. Her father was born in Germany, and is a German citizen, and has a German passport. The family is a mix of German parents, and her father is German. |
| DeltaEdit | Danielle Darrieux was born in Detroit, Michigan, on January 26, 1957. She is a native of Detroit, Michigan, and graduated from the University of Michigan in Ann Arbor, Michigan. |

Table 5: The outputs of Llama3-8B

| Llama3-8B | |
|---|---|
| Edited Knowledge: The mother tongue of Danielle Darrieux is English | |
| Prompt: Danielle Darrieux was born in | |
| Method | Output |
| Original | Danielle Darrieux was born in 1917, and is one of the most famous French actresses of all time. Her film career began in the late 1930's. Her first film was Le Grand Jeu. In 1951 she starred alongside Yves Montand in the film La Belle Ameficaine. |
| MEMIT | Danielle Darrieux was born in addCriterion addCriterion addCriterion addCriterion addCriterion addCriterion addCriterion,LOCATION addCriterionernel addCriterion addCriterionernel |
| AlphaEdit | Danielle Darrieux was born in 1936 in Moscow, Russia. She was born in Moscow, Russia. She was born in Russia. She was born in Russia, Moscow, on December 20, She was born in Moscow, Russia. |
| DeltaEdit | Danielle Darrieux was born in London, England. She attended school in France, New York and Switzerland, and then attended the University of the Pacific (based in Costa Rica) for one year. |