Explainable and Efficient Editing for Large Language Models

Tianyu Zhang
University of Science and Technology of China
Hefei, Anhui, China
tianyuz1@mail.ustc.edu.cn

Junfeng Fang*
University of Science and Technology of China
Hefei, Anhui, China
fjf@mail.ustc.edu.cn

Houcheng Jiang
University of Science and Technology of China
Hefei, Anhui, China
janghc@mail.ustc.edu.cn

Baolong Bi
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
bibaolong23z@ict.ac.cn

Xiang Wang
University of Science and Technology of China
Hefei, Anhui, China
xiangwang1223@gmail.com

Xiangnan He*
MoE Key Lab of BIPC, University of Science and Technology of China
Hefei, Anhui, China
xiangnanhe@gmail.com

Abstract

Large Language Models (LLMs) exhibit remarkable capabilities in storing and retrieving vast amounts of factual knowledge. However, they retain outdated or incorrect information from Web corpora. Since full retraining is costly, locate-and-edit model editing methods offer a feasible alternative. Current methods typically follow a two-stage paradigm: (1) identifying critical layers that store knowledge and (2) updating their parameters to store new knowledge. However, both phases have their inherent limitations. Firstly, layer identification is independent of the knowledge being updated, ignoring the differences in knowledge storage patterns. Secondly, parameter updating suffers from high computational overhead due to gradient descent. To solve these, we propose an **E**xplainable and effi**C**ient model **E**diting method, termed **ECE**. Specifically, we integrate LLM explainability into the editing process, enabling the adaptive identification of the crucial neurons. Through clustering similar knowledge, we enable batch optimization in a single gradient step, significantly reducing computational time without compromising effectiveness. Extensive experiments demonstrate that ECE can achieve superior performance, showcasing the potential of explainability-driven editing methods for LLMs. Code is available at https://github.com/tianyuzhangterry/ECE.

CCS Concepts

• **Computing methodologies** → *Semantic networks.*

Keywords

Large Language Models, Knowledge Editing, Model Explainability

ACM Reference Format:
Tianyu Zhang, Junfeng Fang, Houcheng Jiang, Baolong Bi, Xiang Wang, and Xiangnan He. 2025. Explainable and Efficient Editing for Large Language Models. In *Proceedings of the ACM Web Conference 2025 (WWW '25), April 28-May 2, 2025, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3696410.3714835

*Corresponding author.

1 Introduction

Large Language Models (LLMs) have recently demonstrated remarkable capabilities in storing vast amounts of factual knowledge and retrieving it effectively during inference [13, 34, 45]. The knowledge in LLMs primarily stems from the extensive training data, particularly Web corpora. However, these corpora often contain inaccuracies and outdated information that LLMs may inadvertently store, necessitating targeted modifications to correct these knowledge bases [40]. While retraining the entire LLM is a direct solution, it is resource-intensive, both in terms of time and computational cost [36, 39]. As an efficient alternative, locate-and-edit model editing methods have emerged for updating specific knowledge [4, 30, 31]. These methods generally follow a two-stage paradigm: (1) given an LLM, identifying the critical layers to knowledge storage by causal tracing; (2) given a new piece of knowledge, computing the expected output of the identified layers to ensure correct responses. The expected output is then employed to update the critical layers' parameters, allowing for knowledge updates by adjusting only a small subset of model parameters [54].

While this two-stage paradigm is prevalent, both stages present inherent issues [16, 19, 27, 29]. As illustrated in Figure 1,

- In Stage 1 (*i.e.* critical layer identification), the to-be-updated layers and parameters remain fixed regardless of the type of new knowledge. However, recent studies on LLM explainability suggest that different types of knowledge are stored in distinct layers and neurons [41]. Current methods ignore the explainable correspondence between knowledge and neurons, making it difficult to identify the precise neurons based on the input knowledge. This results in suboptimal editing performance.
- In Stage 2 (*i.e.* parameter update), the process is computationally expensive, as the optimal output must be calculated independently for each knowledge instance. In practice, the number of knowledge updates could exceed tens of thousands, imposing significant efficiency constraints. Furthermore, in lifelong editing scenarios (*i.e.*, continuous updating the same LLM [19, 23]), each update has to modify all key layers and neurons identified in Stage 1 [14], significantly increasing time consumption.

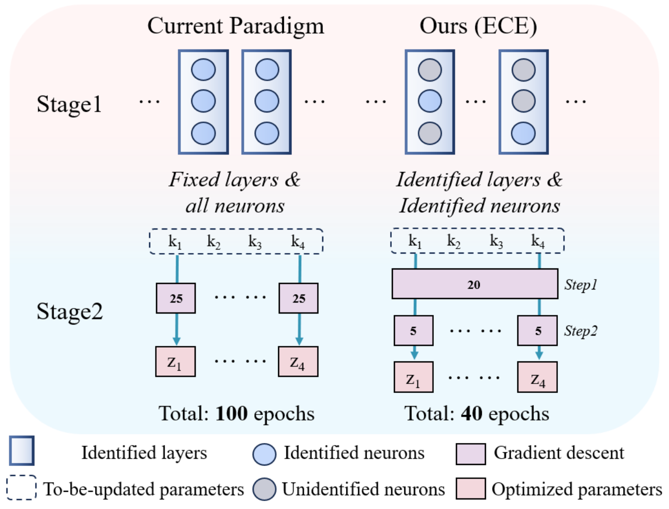


Figure 1: Paradigm difference between existing methods and our method for sequential modeling editing. Our method succeeds in identifying precise neurons with more efficient batch update.

Thus, a key question arises: *Can we design an explainable and efficient editing method that adaptively identifies key neurons in Stage 1 and streamlines parameter updates in Stage 2?*

To answer this question, we propose an **E**xplainable and effi**C**ient sequential **E**diting method, called **ECE**. Specifically, in Stage 1, ECE integrates the concept of LLM explainability [40, 56] into the editing process, enabling the adaptive identification of the most relevant layers and neurons based on the input knowledge [56]. This identification is inspired by the advanced attribution methods in LLM explainability, such as activation-based [6], weight-based [50], and residual-flow-based methods [35, 41], making ECE focus exclusively on the most critical parameters. By isolating neurons unrelated to the updated knowledge, ECE perserves the integrity of other knowledge stored within the LLM.

Furthermore, the explainability introduced in Stage 1 lays as a foundation for accelerating Stage 2. This acceleration is manifested in two key aspects: (1) Unlike current methods updating all parameters in every key layer, ECE performs layer- and neuron-wise updates changing only a limited set of parameters within the selected layers identified by the attribution approaches. This significantly reduces the overall parameter volume. (2) Current research has verified that knowledge instances of similar types often exhibit consistent distributions of key neurons and optimal outputs across key layers [5, 15]. Building on this insight, ECE employs advanced clustering algorithms [35, 41] to group type-similar knowledge based on the distribution of key neurons. This allows us to compute the optimal outputs for these knowledge instances simultaneously in a single gradient descent step, drastically reducing the time-consuming gradient descent process.

We conduct extensive qualitative and quantitative experiments on GPT2-XL (1.5B) [38], GPT-J (6B) [49], and Llama-3 (8B) [11]. Results across multiple datasets demonstrate that, compared to the baselines (*e.g.*, Fine-tuning [44], MEND [32], ROME [30], and MEMIT [31]), ECE significantly outperforms in editing effectiveness across several metrics, including efficacy, generalization, specificity, fluency, and consistency. Moreover, ECE achieves an average speedup of 3.27× in editing efficiency for sequence editing. These findings confirm that incorporating LLM explainability to streamline the editing process can lead to improvements in both effectiveness and efficiency.

Our key contributions are summarized as follows:

- We systematically analyze the inherent issues in current locate-and-edit editing methods, specifically the lack of explainability and inefficiency during the critical layer identification and parameter update phases.
- We propose a novel sequential editing method, termed ECE. By integrating attribution methods from LLM explainability, ECE adaptively identifies and updates neurons within LLMs, achieving improvements in both effectiveness and efficiency.
- Experiments across multiple LLMs demonstrate that ECE outperforms leading editing methods across five general evaluation metrics and two commonly used datasets.

2 Preliminary

Let θ be model parameters and f_θ be the base model, model editing modifies f_θ to its edited version $f_{\theta'}$ [4, 30, 31]. The objective is to adjust the model's responses to a specified set of knowledge instance, while preserving its performance on all other knowledge instances [10]. The intended edit descriptor is denoted as $\{(x_i^e, y_i^e)\}_{i\in[1,N]}$, where $f_\theta(x_i^e) \neq y_i^e$, and N represents the total number of editing instances. This set of instances forms the editing scope I_{edit}, while O_{edit} represents the instances outside the editing scope. Formally, a successful edit can be expressed as:

$$f_{\theta'}(x_i) = \begin{cases} y_i^e, & \text{if } x_i \in I_{edit}, \\ f_\theta(x_i), & \text{if } x_i \in O_{edit}. \end{cases} \tag{1}$$

Sequential model editing [29] refers to the process of continuously refining a pre-trained model, f_{θ_0}, through a series of updates, where each update incorporates modifications to adjust the model's outputs [12, 54]. This process is expressed as:

$$\theta' \leftarrow \arg\min_\theta \left(\sum_{s=0}^{S} \sum_{i=s\times n}^{(s+1)\times n} \| f_\theta(x_i^e) - y_i^e \| \right), \tag{2}$$

where n represents the batch size, and S represents the sequential editing step.

In practice, each update involves introducing a set of factual triples in the form of (s, r, o), where s represents the subject, r the relation, and o the object (*e.g.*, s="The largest ocean", r="is", o="Pacific Ocean"). After the t-th edit, the updated model f_{θ_t}, built on its predecessor $f_{\theta_{t-1}}$, is optimized to generate precise target outputs for the corresponding inputs $\mathbb{D}_{edit_t}$, while preserving accuracy on inputs outside the current editing scope. Following ROME [30] and MEMIT [31], we conceptualize the feed-forward network (FFN) layer of a Transformer [46] as a linear associative memory. This approach effectively utilizes linear mappings within the FFN to serve as key-value pairs for information retrieval [25]. Our objective is to adjust the output of the LLM such that the input (s_i, r_i) produces the output o_i.

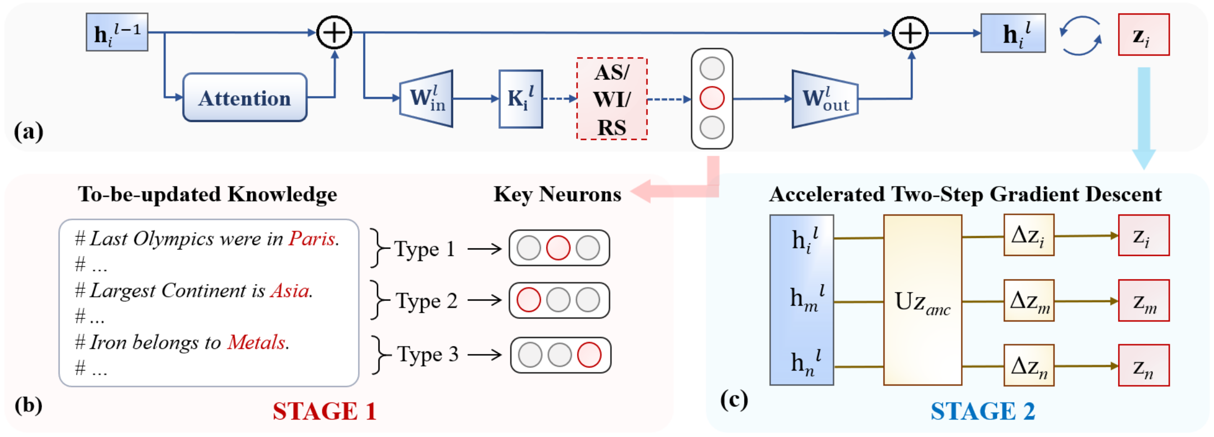


Figure 2: Overview of ECE. (a) The neuron-wise identification approach is based on attribution methods using Activation Score, Weight Importance, or Residual Sensitivity. h denotes hidden state, z denotes optimal representation, and red color in circle highlights the identified neurons. (b) The clustering method applies to neurons corresponding to knowledge instances. (c) The acceleration is achieved by the two-step gradient descent method based on the clustering results. U and Δ in yellow represent the step-1 and step-2 gradient descent.

The process begins by identifying the activation output from the last subject token S at the l-th FFN layer, which serves as the key k_i^l. These keys are computed from the input weights W_{in}^l and are processed through the output weights W_{out}^l to generate the corresponding values v_i^l. This setup allows us to capture the LLM's inherent n knowledge pairs, associating input keys $K_0 = [k_1 \,|k_2| \ldots \,|\, k_n]$ with corresponding values $V_0 = [v_1 \,|v_2| \ldots \,|\, v_n]$. We aim to integrate u additional key-value pairs associated with new knowledge, denoted as $K_1 = [k_{n+1} \,|k_{n+2}| \ldots \,|\, k_{n+u}]$ and $V_1 = [v_{n+1} \,|v_{n+2}| \ldots \,|\, v_{n+u}]$, while preserving the original associations. The values V_1 are optimized through gradient descent to maximize the probability of the target token outputs, as detailed in MEMIT [31].

The optimization framework is defined as:

$$\Delta = \arg\min_{\hat{\Delta}} \left(\left\| (W + \hat{\Delta})K_1 - V_1 \right\|^2 + \left\| (W + \hat{\Delta})K_0 - V_0 \right\|^2 \right), \quad (3)$$

where W represents the output weights of the target FFN layer and Δ denotes the required weight updates. The knowledge retention can be expressed as $WK_0 = V_0$. Utilizing the least squares method, the optimal weight update Δ is calculated as follows:

$$\Delta = RK_1^T \left(K_0 K_0^T + K_1 K_1^T \right)^{-1}, \quad (4)$$

where $R = V_1 - WK_1$. Here, the matrix $K_0 K_0^T$ can be approximated by $\lambda \mathbb{E}[kk^T]$, a covariance statistic without centering, derived from an empirical dataset of vector inputs to the layer.

3 Methodology

In this section, we detail how ECE achieves simultaneous improvements in efficiency and effectiveness through the incorporation of explainability. Specifically, in Section 3.1, we present neuron-wise identification that employs mature attribution algorithms to locate fine-grained layers and neurons. In Section 3.2, we demonstrate how the attribution results accelerate the learning of key matrices, thereby significantly enhancing editing efficiency. Finally, Section 3.3 presents the parameter update process.

3.1 Neuron Identification

We first revisit the process of identifying parameters for updating in current model editing approaches. Existing methods [30, 31] primarily use causal tracing [30] to pinpoint layers with the highest causal effect, assuming these layers store key knowledge in the LLM. However, this approach has two major limitations:

- **Independence from Specific Knowledge:** Once key layers are identified, all neurons within them are treated equally during editing, overlooking the fact that different types of knowledge are encoded in distinct patterns [20]. Each neuron plays a unique role — some are crucial for specific information, while others may be less relevant or even inactive for certain tasks;
- **Overlooking Neurons Outside Key Layers:** Focusing solely on selected layers risks missing important neurons distributed across other layers that also contribute to knowledge storage. Moreover, research about "dead neurons problem" [47] points that inactive or "dead" neurons consume capacity without contributing meaningfully to the model's output. Editing both inactive and critical neurons indiscriminately can reduce the precision of updates.

These limitations highlight the need for more precise and efficient approaches to knowledge editing beyond layer-level modifications.

To solve these, an intuitive optimization is to adaptively identify key neurons for editing based on to-be-updated knowledge. Drawing inspiration from well-established neuron attribution methods in LLM explainability [50], we employ three attribution methods to quantify neuron relevance according to to-be-updated knowledge as follows:

- **Activation Score (AS)** [35] ranks neurons directly by the magnitude of their activation values during inference, identifying those that are highly active in processing specific inputs. Formally:

$$AS_i = |a_i(x_j)|, \quad (5)$$

where $a_i(x_j)$ denotes the activation value of neuron i for knowledge instance x_j.

- **Weight Importance (WI)** [41] evaluates neurons based on the weights involved in transmitting information between neurons, emphasizing the significance of neurons with stronger internal connections. The importance score is defined as:
$$WI_i = |W_{ij}|, \tag{6}$$
where W_{ij} denotes the weight between neuron i and neuron j.
- **Residual Sensitivity (RS)** [41] assesses neurons by their contribution to the final output through the residual stream. The importance score is defined as:
$$RS_i = a_k^l (W_{\text{out}}^l)_k, \tag{7}$$
where a_k^l is the activation value of neuron k in layer l, $(W_{\text{out}}^l)_k$ denotes the output weight for neuron k in layer l.

Building on prior work that suggests that neurons within Feed-Forward Networks (FFNs) contain significant amounts of specific factual information [6, 35, 41], we prioritize the optimization of selected neurons rather than modifying the entire parameter space. For each knowledge instance (s_i, r_i, o_i), we compute the score values from one of the three methods above and obtain scores $\mathbf{Q}_i$ for the neurons. Using a descending sorting method, we then rank and group the neurons with the highest scores together. We then select a subset of neurons by identifying those whose cumulative score exceeds a predetermined fraction p of the total score:

$$\mathcal{I} = \arg \min_{\mathcal{I} \subseteq \{1,\ldots,N\}} |\mathcal{I}| \quad \text{s.t.} \quad \sum_{j \in \mathcal{I}} \mathbf{Q}_{ij} \geq p \cdot \sum_{j=1}^{N} \mathbf{Q}_{ij}, \tag{8}$$

where $\mathbf{Q}_{ij}$ represents the score of the j-th neuron, and $\mathcal{I}$ is the selected set of neuron indices. This identification mechanism enables targeted edits, ensuring efficiency and precision in the parameter update process.

Given that the model may need to edit multiple knowledge facts in parallel (*i.e.*, in a batch), which may correspond to different neurons across the network, we aggregate neuron scores across all batch samples. This allows for a unified neuron identification based on cumulative influence contribution, which streamlines the model's response to various edits.

3.2 Accelerated Learning of Key Matrices

After identifying the to-be-updated parameters $\mathcal{I}$, the next step is to obtain the optimal representations of the relevant layers post-editing, *i.e.*, the value matrix V_1 for the to-be-updated knowledge as outlined in Section 2. Note that this step serves as a key driver of ECE's acceleration by leveraging the attribution results from the above Section. Next, we will detail how ECE achieves simultaneous improvements in efficiency and effectiveness through two progressive steps: knowledge clustering and two-step gradient descent.

3.2.1 Knowledge Clustering. Different types of knowledge inherently possess varying textual attributes such as geographical, demographic, and temporal concepts, which implies that their key information is stored in different regions within the model [35, 41]. Hence, we propose a clustering approach to pre-classify knowledge, thereby avoiding conflicts that may arise due to the distinct characteristics of the knowledge being edited. Specifically, we represent each knowledge instance x_i and its corresponding set of identified neurons as a key-value pair. By applying a k-means clustering algorithm based on Jaccard similarity, we aggregate knowledge instances with similar neuron identifications into clusters. In our clustering approach, the objective is to minimize the Jaccard distance between the data points and their respective cluster centroids, formulated as:

$$\arg \min_{S} \sum_{i=1}^{k} \sum_{x_j \in S_i} d_J(x_j, c_i), \tag{9}$$

where S_i represents the i-th cluster, x_j denotes a data point in cluster S_i, and c_i is the centroid of cluster S_i. By minimizing this objective, we ensure that the knowledge within each cluster exhibits high internal similarity.

We treat each resulting cluster as a smaller batch and subsequently use the sample closest to the center of each cluster as an anchor sample. The anchor sample for each cluster is defined as the sample with the smallest sum of Jaccard distances to all other samples in the cluster. This criterion ensures that the selected anchor is the most representative data point of its cluster, which is formulated as follows:

$$x_{\text{anc}} = \arg \min_{x_j \in S_i} \sum_{x_k \in S_i} d_J(x_j, x_k). \tag{10}$$

This approach allows us to perform subsequent editing tasks in a more fine-grained manner, tailored to the specific attributes of each knowledge category.

3.2.2 Two-step Gradient Descent. To enhance computational efficiency in the sequential batch editing process, we introduce a two-step gradient descent approach applied to the clusters identified in the previous step. For each cluster, the gradient descent is divided into a common phase and an instance-specific phase, allowing us to maximize shared information while maintaining unique adjustments for each instance within the cluster. This approach significantly reduces redundant calculations and accelerates the model adaptation process.

In the first phase, we perform a unified gradient descent over the entire cluster, conducting 20 epochs of shared updates from a total of 25 epochs for the whole process. In this modified shared gradient descent, all instances within the cluster share the update vector z_{anc} associated with the anchor sample x_{anc} of the cluster for the first 20 epochs. This shared step captures common patterns by optimizing parameters based on the anchor sample, which is broadly representative of the entire cluster, thereby reducing the number of repetitive updates. Let C_u represent the u-th cluster, and let h_{anc}^L denote the update vector for the anchor sample within the cluster. By minimizing the average loss based on the anchor sample x_{anc} and its corresponding edit target y_{anc}^e, we calculate the unified update $\mathcal{U}z_u$ for the entire cluster as follows:

$$\mathcal{U}z_u = \arg \min_{\delta} - \log \mathbb{P}_G (h_{\text{anc}}^L + \delta)(y_{\text{anc}}^e \mid x_{\text{anc}}), \tag{11}$$

This unified update step captures the general characteristics of the cluster by leveraging the anchor sample, thereby setting a common foundation for the individual updates that follow.

In the second phase, we refine this update by performing five additional epochs of gradient descent tailored to each instance within the cluster. This step fine-tunes the model on unique variations and specific details, accommodating the individual characteristics and ensuring that the final updates are well-adapted to each instance. The optimization objective is as follows:

$$\delta_i^* = \arg\min_{\delta_i} -\log \mathbb{P}_G(h_i^L + \mathcal{U}z_u + \delta_i)(y_i^e \mid x_i), i \in C_u, \quad (12)$$

where δ_i^* is the instance-specific adjustment derived by minimizing the residual error after applying the shared update $\mathcal{U}z_k$. Thus, the two-step update z_i for a specific instance i in cluster C_k can be integrated together as:

$$z_i = h_i^L + \mathcal{U}z_u + \delta_i^*. \quad (13)$$

This step preserves individual differences by fine-tuning the shared update to better align with the specific characteristics and requirements of each instance.

By separating the optimization into these two steps, we achieve both efficiency and adaptability. The unified 20-epoch update captures the core features shared among instances within a cluster, while the five-epoch instance-specific phase ensures that each instance receives the necessary unique adjustments. This design reduces the overall computation required by avoiding repetitive updates for common features. Consequently, this method provides a balanced approach that maintains the specificity of individual updates while optimizing shared computations, leading to faster convergence and lower computational costs in comparison to traditional instance-by-instance gradient descent methods.

Traditional methods have a time complexity of $O(N_p \cdot L \cdot T_g)$, where N_p is the number of model parameters, L is the number of layers, and T_g is the number of gradient descent iterations. These methods update parameters across multiple layers, making them computationally expensive. In contrast, ECE focuses only on key neurons with knowledge clustering and a two-step approach, achieving significant optimization in a complexity of $O(K \cdot B \cdot T_k + M \cdot T_u + B \cdot M \cdot T_i)$. K is the number of clusters, B is the batch size, T_k is the number of K-means iterations, and M is the number of selected key neurons, where $M \ll P$ and $T_u + T_i = T_g$. Empirically, ECE is faster than others by about hundreds of seconds per batch.

3.3 Parameter Updates

After identifying the to-be-updated parameters $\mathcal{I}$ in Section 3.1 and the value matrix V_1 in Section 3.2, we arrive at the final step: performing the parameter update on $\mathbf{W}_{\text{out}}$.

Following MEMIT [31], we derive the solution for Eqn. 19 using the method of minimal squared error as:

$$\hat{\Delta}^* = \hat{R}\hat{K}_1^T\hat{C}^{-1}, \quad (14)$$

where $\hat{R} = V_1 - \hat{W}\hat{K}_1$ and $\hat{C} = \hat{K}_0\hat{K}_0^T + \hat{K}_1\hat{K}_1^T$. In order to maintain continuity, we approximate $K_0K_0^T$ with $\lambda\mathbb{E}\left[kk^T\right]$, where λ is a hyperparameter balancing the retention of prior knowledge with the integration of new edits. The submatrix $\hat{K}_0\hat{K}_0^T$ is then derived from $K_0K_0^T$ by indexing only the identified neurons. Additionally, as each editing round progresses, newly edited knowledge becomes the reference knowledge for subsequent rounds, which requires updating $K_0K_0^T$ after each iteration.

4 Experiments

We conduct experiments to demonstrate the effectiveness of ECE. The experiments aim to address the following research questions:

- **RQ1:** How does ECE's performance on sequential model editing tasks compared to existing methods?
- **RQ2:** What is the impact of different parameter settings on the performance and stability of sequential model editing?
- **RQ3:** How much efficiency improvement can ECE achieve in comparison to existing editing techniques?
- **RQ4:** Can LLMs preserve the original general abilities after extensive sequential edits?

4.1 Experimental Settings

Datasets & Evaluation Metrics. To evaluate the effectiveness of our method, we utilize two datasets: Counterfact [30] and ZsRE [26]. For the Counterfact dataset, we utilize five evaluation metrics as defined in previous studies [30, 31]: **Efficacy** (efficiency success), **Generalization** (paraphrase success), **Specificity** (neighborhood success), **Fluency** (generation entropy), and **Consistency** (reference score). For the ZsRE dataset, we apply three evaluation metrics, also defined in previous work [30–32]: **Efficacy**, **Generalization**, and **Specificity**. For more details, see Appendix B.

Baselines: For baseline comparisons, we consider several model editing approaches across different categories. (1) Fine-tuning based: **FT-L** [44] and **FT-M** [58] directly fine-tune a single layer's feed-forward network (FFN); (2) Locate-and-edit: **ROME** [30] which identifies critical neuron activations within middle-layer feed-forward modules that influence factual prediction and **MEMIT** [31] treats the transformer's feed-forward layer as a linear associative memory and applies minimum square error optimization to introduce new key-value associations; (3) Meta-learning based: **MEND** [32] uses a hyper-network to transform gradients obtained via standard fine-tuning; (4) Memory-based: **SERAC** [33] which employs an external cache to store explicit edits.

Implementation Details: Our comparative analysis evaluates the performance of various editing methods on three autoregressive language models, GPT2-XL (1.5B) [38], GPT-J (6B) [49], and Llama3 (8B) [11]. Further details are provided in the Appendix C.

4.2 Editing Performance (RQ1)

In this subsection, we present a detailed comparison of ECE against other established methods for the sequential model editing task, conducted using GPT2-XL, GPT-J, and Llama3 models. The experiments are performed on 2000 edited samples, with an editing batch size of 100 (batch size refers to the number of samples edited simultaneously during each round of sequential editing), and evaluated on the Counterfact and ZsRE datasets. The evaluation results, using various metrics and across all datasets, are summarized in Table 1. From this table and Figure 5 in Appendix D, we can observe that:

- **Observation 1: ECE outperforms other baseline methods in almost all critical metrics in the sequential editing task.** ECE demonstrates notable improvements compared to baseline methods across both datasets and models, achieving significant gains across all metrics. For instance, on the Llama3 (8B) model with Counterfact dataset, ECE exhibits an average improvement of approximately 56.39% across the editing success rate containing efficacy, generalization, and specificity. On the ZsRE dataset, ECE's performance is even more remarkable, achieving multiple-fold improvements across all three models.

Table 1: Comparison of ECE with existing methods on the sequential model editing task. The bold represents the best results from our methods and the underline indicates the best results of baselines.

Model	Method	Counterfact					ZsRE		
		Efficacy↑	Generalization↑	Specificity↑	Fluency↑	Consistency↑	Efficacy↑	Generalization↑	Specificity↑
	Pre-edited	7.85±0.26	10.58±0.26	89.48±0.18	635.23±0.11	24.14±0.08	36.99±0.30	36.34±0.30	31.89±0.22
Llama3	FT-L	83.33±0.37	67.79±0.40	46.63±0.37	233.72±0.22	8.77±0.05	30.48±0.26	30.22±0.32	15.49±0.17
	FT-W	61.23±0.38	62.40±0.24	47.05±0.41	492.34±0.23	3.57±0.03	32.08±0.35	31.43±0.23	14.72±0.16
	MEND	63.24±0.31	61.17±0.36	45.37±0.38	372.16±0.80	4.21±0.05	0.91±0.05	1.09±0.05	0.53±0.02
	ROME	64.40±0.47	61.42±0.42	49.44±0.38	449.06±0.26	3.31±0.02	2.01±0.07	1.80±0.07	0.69±0.03
	MEMIT	65.65±0.47	64.65±0.42	51.56±0.38	437.43±1.67	6.58±0.11	34.62±0.36	31.28±0.34	18.49±0.19
	SERAC	67.78±0.29	60.98±0.31	45.26±0.21	384.49±0.73	15.71±0.03	1.24±0.05	1.03±0.06	0.56±0.02
	ECE (AS)	92.90±0.10	82.85±0.27	80.93±0.20	628.32±0.14	31.62±0.11	89.29±0.14	83.25±0.25	30.03±0.23
	ECE (WI)	**99.60±0.16**	**90.65±0.25**	87.24±0.19	629.37±0.16	**31.63±0.11**	**95.34±0.12**	**90.29±0.20**	**33.04±0.23**
	ECE (RS)	97.50±0.15	85.25±0.31	**87.93±0.19**	**631.09±0.13**	31.00±0.10	93.81±0.15	88.38±0.22	32.92±0.23
	Pre-edited	22.23±0.73	24.34±0.62	78.53±0.33	626.64±0.31	31.88±0.20	22.19±0.24	31.30±0.27	24.15±0.32
GPT2-XL	FT-L	63.55±0.48	42.20±0.41	57.06±0.30	519.35±0.27	10.56±0.05	37.11±0.39	33.30±0.37	10.36±0.17
	FT-W	42.70±0.49	35.93±0.40	63.06±0.31	565.96±0.23	13.03±0.06	24.97±0.32	22.40±0.30	12.73±0.18
	MEND	50.80±0.50	50.80±0.48	49.20±0.51	407.21±0.08	1.01±0.00	0.00±0.00	0.00±0.00	0.00±0.00
	ROME	54.60±0.48	51.18±0.40	52.68±0.33	366.13±1.40	0.72±0.02	47.50±0.43	43.56±0.42	14.27±0.19
	MEMIT	94.70±0.22	85.82±0.28	60.50±0.32	477.26±0.54	22.72±0.15	79.17±0.32	71.44±0.36	26.42±0.25
	SERAC	51.50±0.44	50.04±0.42	52.13±0.47	418.12±0.76	1.55±0.02	38.58±0.36	41.49±0.46	13.78±0.21
	ECE (AS)	95.90±0.14	85.90±0.29	72.80±0.28	607.33±0.33	**38.75±0.12**	83.6±0.25	72.98±0.39	**26.72±0.22**
	ECE (WI)	96.20±0.19	**89.10±0.31**	**78.44±0.27**	**625.74±0.13**	32.84±0.10	86.71±0.39	**79.33±0.33**	26.12±0.25
	ECE (RS)	**98.60±0.18**	88.30±0.34	77.65±0.26	622.22±0.17	33.84±0.10	**88.46±0.39**	78.17±0.39	25.64±0.28
	Pre-edited	16.22±0.31	18.56±0.45	83.11±0.13	621.81±0.67	29.74±0.51	26.32±037	25.79±0.25	27.42±0.53
GPT-J	FT-L	92.15±0.27	72.38±0.38	43.35±0.37	297.92±0.77	6.65±0.10	72.37±0.29	68.91±0.32	19.66±0.23
	FT-W	48.35±0.49	31.42±0.39	68.71±0.28	587.20±0.23	29.41±0.09	39.81±0.36	32.55±0.33	27.76±0.26
	MEND	46.15±0.50	46.22±0.51	53.90±0.48	242.41±0.41	3.94±0.03	0.71±0.04	0.71±0.04	0.52±0.03
	ROME	57.50±0.48	54.20±0.40	52.05±0.31	589.28±0.08	3.22±0.02	56.42±0.42	54.65±0.42	9.86±0.16
	MEMIT	98.55±0.11	95.50±0.16	63.64±0.31	546.28±0.88	34.89±0.15	94.91±0.16	90.22±0.23	30.39±0.27
	SERAC	55.88±0.36	51.39±0.53	53.78±0.39	390.21±0.49	4.36±0.03	49.48±0.37	1.59±0.03	8.84±0.18
	ECE (AS)	98.82±0.09	95.73±0.23	74.25±0.26	618.5±0.23	42.22±0.13	96.20±0.15	93.35±0.25	27.19±0.21
	ECE (WI)	**100.00±0.00**	96.35±0.15	79.41±0.26	**619.82±0.17**	**42.34±0.13**	**99.74±0.03**	**96.88±0.14**	28.49±0.26
	ECE (RS)	**100.00±0.00**	**96.45±0.14**	**79.99±0.25**	619.38±0.17	41.10±0.13	97.28±0.13	94.99±0.21	**28.86±0.24**

4.3 Impact of Parameter (RQ2)

As the model undergoes successive modifications with editing tasks, sequential model editing methods face two inherent challenges: **model forgetting** and **model failure**. Model forgetting occurs when cumulative parameter changes from successive edits erode previously modified knowledge, resulting in a decline in performance and stability over time [7, 18]. Meanwhile, model failure refers to the progressive loss of the model's ability to generate coherent responses as edits accumulate, potentially leading to model collapse, where the output becomes repetitive or nonsensical [16, 17]. To explore these effects, we examine the influence of two key parameters **number of edits** and **batch size** on the sequential model editing process. Specifically, we analyze how the number of edits impact the performance of ECE compared to MEMIT and ROME on Llama3 and Counterfact dataset at Figure 3.

- **Observation 2: ECE maintains stable performance across all metrics as the number of edited samples increases.** As illustrated in Figure 3, ECE shows resilience against model failure and forgetting as the number of editing rounds grows. In contrast, both ROME and MEMIT experience considerable performance declines, particularly in Specificity, Fluency, and Consistency,. As more samples are edited, ROME and MEMIT increasingly fail to uphold model integrity and impair model's original capabilities.
- **Observation 3: ECE consistently outperforms across a range of batch sizes in sequential editing tasks.** From Figure 6 in Appendix D we can see that MEMIT's performance declines markedly as batch size decreases and the number of editing rounds increases. This effect is especially evident when the batch size is reduced to 10, showing a notable drop in editing effectiveness across all metrics. In comparison, ECE demonstrates stable performance across these metrics, regardless of batch size.

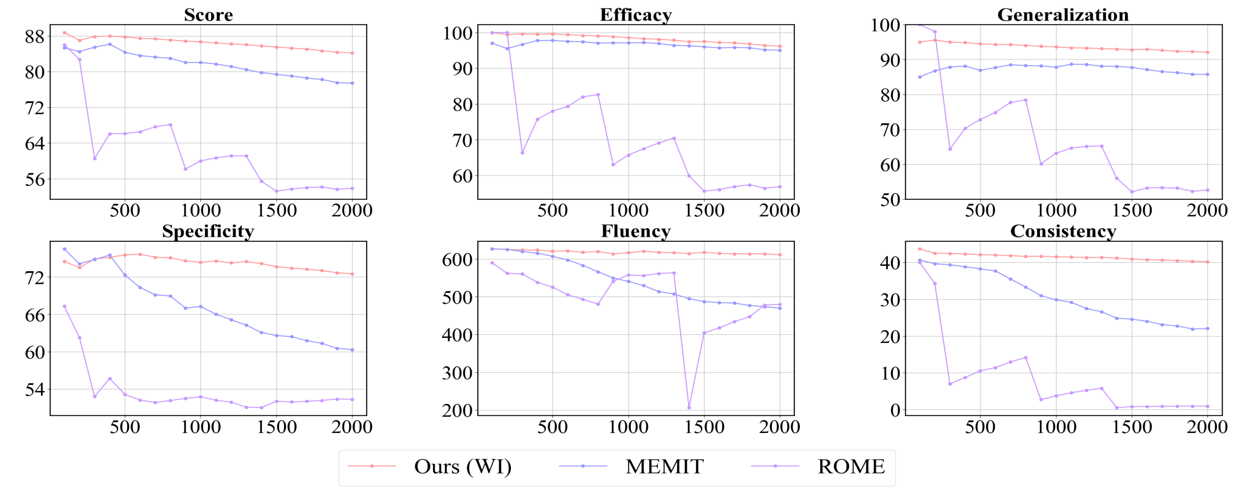


Figure 3: Editing performance of ECE and baselines with different numbers of edits (batch size 100) evaluated on Llama3 model and Counterfact dataset. Score is the harmonic mean of Efficacy, Generalization, and Specificity.

Method	GPT2-XL	GPT-J	Llama3
FT-L	191.42s	303.26s	451.23s
FT-W	157.44s	263.74s	374.35s
MEND	26.79s	49.16s	67.85s
ROME	422.37s	764.82s	914.63s
MEMIT	222.51s	334.74s	484.14s
SERAC	384.91s	634.74s	834.56s
ECE (AS)	119.39s	187.44s	216.87s
ECE (WI)	104.88s	178.23s	214.19s
ECE (RS)	148.48s	199.32s	231.64s

Table 2: Times per batch for various methods evaluated on ZsRE dataset with different models.

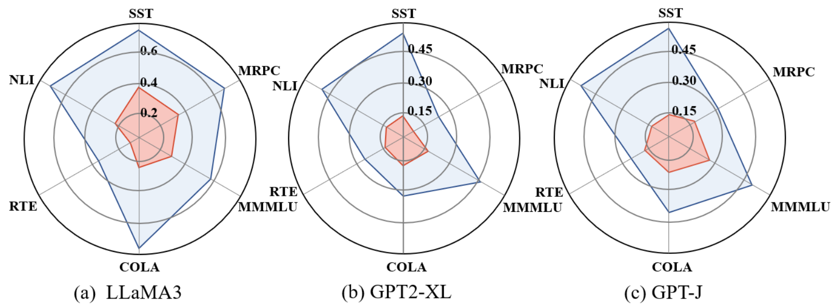


Figure 4: Comparison of general capabilities for MEMIT and ECE (WI) with 2000 edits on (a) Llama3 model, (b) GPT2-XL model, and (c) GPT-J model.

4.4 Time Overhead Comparison (RQ3)

To evaluate the efficiency of our approach in sequential knowledge editing tasks, we conducted tests across three model architectures, benchmarking our method against established baselines. The evaluation involved a continuous editing scenario with a total of 2,000 edits and a batch size of 100. These numbers shown in Table 2 represent the average time of the whole editing process conducting at the first time. This means they could reflect the results of editing efficiency including getting expected output through gradient descent and further techniques.

- **Observation 4: Our methods consistently maintained superior efficiency, with the WI method being the fastest.** Our method demonstrated significant improvements in editing speed, surpassing nearly all baseline methods. Although MEND displayed the shortest editing times, its low effectiveness on the ZsRE dataset limits its applicability. In general, combined with the editing performance results, ECE achieves a notable improvement in efficiency through acceleration approaches.

4.5 General Ability Test (RQ4)

To evaluate the impact of model editing on the general capabilities of large language models (LLMs), we have selected six natural language tasks from the General Language Understanding Evaluation (GLUE) benchmark [48], a public leaderboard for tracking performance with respect to a wide range of linguistic phenomena found in natural language. The chosen downstream tasks are as follows: (1) **SST (Stanford Sentiment Treebank)** [43], which involves classifying individual sentences based on sentiment in movie reviews. (2) **MRPC (Microsoft Research Paraphrase Corpus)** [8], a task focused on text matching to assess semantic similarity. (3) **MMLU (Massive Multi-task Language Understanding)** [21], which evaluates language models on multi-task accuracy. (4) **CoLA (Corpus of Linguistic Acceptability)** [51], a single-sentence classification task drawn from linguistic theory literature. (5) **RTE (Recognizing Textual Entailment)** [3], a natural language inference task to determine whether a given premise entails a hypothesis. (6) **NLI (Natural Language Inference)** [52], which requires the model to identify logical relationships between pairs of sentences.

- **Observation 5: ECE consistently maintains the general capabilities of the LLM during sequential editing without incurring model failure.** From Figure 4 we can see that: as the number of knowledge edits grows, ECE maintains performance levels comparable to those of the unedited LLMs, showing no negative impact on the model's core general capabilities. In contrast, both ROME and MEMIT have poor performance in different general capabilities, suggesting that the model has already suffered significant degradation.

5 Related work

Model editing has emerged as an essential research area focused on modifying the behavior of pre-trained large language models (LLMs) to integrate new knowledge or correct factual errors, all without the need for extensive retraining.

5.1 Model Editing

Model editing offers a targeted solution, aiming to make specific changes to a model's knowledge while ensuring that the model maintains its general performance across unrelated inputs. Model editing approaches can broadly be divided into two categories: methods that preserve the model's original parameters and methods that directly modify the model's parameters.

Preserve Models' Parameters. Methods in this category aim to preserve the pre-trained model's parameters by introducing new knowledge through external components or retrieval mechanisms, rather than altering the core model itself. IKE [57] leverages in-context learning to adjust model outputs based on retrieved demonstrations, thus avoiding any gradient-based updates. Similarly, systems like SERAC [33] keep the model's parameters unchanged and use a counterfactual model to make edits, isolating the editing process from the base model. T-Patcher [22] introduces an additional neuron for each specific output error, while CaliNet [9] injects neurons to handle multiple knowledge cases. MELO [55] dynamically activates LoRA blocks indexed within an internal vector database, allowing models to behave differently depending on the retrieved block. On the other hand, GRACE [19] maintains a codebook to store knowledge and updates sequentially without modifying the core model. Similarly, Larimar [7] extends the idea of preserving model parameters by enhancing LLMs with distributed episodic memory, which serves as an external knowledge source.

Modify Model Parameters. Methods that modify LLM parameters focus on directly updating the internal weights of the model to incorporate new knowledge. Knowledge Neurons (KN) [6] identifies crucial neurons that encode factual knowledge within the feed-forward networks (FFNs) of the model and updates them accordingly. Methods such as KE [4] and MEND [32] employ hypernetworks to predict the necessary weight updates for new knowledge, leveraging meta-learning approaches to minimize computational overhead. ROME [30] and MEMIT [31] allow for large-scale direct editing of LLMs by locating and modifying specific knowledge in certain layers of models like GPT. ROME utilizes causal mediation analysis to identify the layers for storage and performs targeted updates in these areas. MEMIT extends this approach, enabling simultaneous edits across multiple factual associations by modifying key neurons in the feed-forward layers. PMET [28] introduces attention values into the editing process, refining the selection of critical neurons for editing. To improve the stability and performance of parameter modification approaches, especially for sequential model editing tasks, PRUNE [29] restricts the maximum singular value of parameter changes to avoid model degradation, while RECT [16] retains parameters with minimal changes to ensure stability. Furthermore, COMEBA-HK [27] introduces hook layers to define the scope of editing.

5.2 Model Explainability

Due to the high computational costs involved and the assertion that only a select subset of neurons plays a crucial role in decision-making, existing methods are commonly combined with ranking algorithms to streamline the process [1]. Based on the premise that models learning similar properties often exhibit shared neurons, these neurons are ranked by metrics such as correlation coefficients and learned parameter weights [2]. The Summarize and Score (SASC) [42] pipeline generates natural language explanations for large language model modules by first identifying n-grams that strongly activate the module and then evaluating these explanations with synthetic data to assess their relevance. The weight banding [37] studies weights that connect neurons, seeking to develop algorithms that reveal underlying logical structures.

6 Limitation and Discussion

While ECE shows significant improvements in both explainability and efficiency for sequential model editing tasks, there are still limitations to our study. Our evaluations are primarily focused on a few common and mainstream language models. Moreover, the experiments are currently based on existing datasets that use structured text languages. Looking ahead, we are committed to exploring more diverse techniques to further enhance the explainability and improve the overall efficiency and robustness of sequential editing, adapting it to a broader range of applications and advancing its capabilities for real-world deployment.

7 Conclusion

In summary, we presented Explainable and Efficient Sequential Editing (ECE), a method that addresses key limitations in the two-stage knowledge editing process for LLM. ECE enhances Stage 1 by adaptively identifying critical layers and neurons, leveraging model explainability for targeted updates. In Stage 2, ECE clusters similar keys to enable batch optimization, significantly reducing computational costs. Experimental results across different evaluation metrics and datasets demonstrate that ECE achieves superior editing performance with a substantial increase in efficiency, showcasing its potential to make model editing both explainable and efficient for real-world applications.

Acknowledgments

This research is supported by the National Science and Technology Major Project (No. 2023ZD0121102) and the National Natural Science Foundation of China (No. U24B20180, No. 92270114, No. 62121002). This research is supported by the advanced computing resources provided by the Supercomputing Center of USTC.

References

[1] Omer Antverg and Yonatan Belinkov. 2022. On the Pitfalls of Analyzing Individual Neurons in Language Models. In *ICLR*.

[2] Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James R. Glass. 2019. Identifying and Controlling Important Neurons in Neural Machine Translation. In *ICLR*.

[3] Luisa Bentivogli, Bernardo Magnini, Ido Dagan, Hoa Trang Dang, and Danilo Giampiccolo. 2009. The Fifth PASCAL Recognizing Textual Entailment Challenge. In *TAC*.

[4] Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing Factual Knowledge in Language Models. In *EMNLP (1)*. 6491–6506.

[5] Hoagy Cunningham, Aidan Ewart, Logan Riggs, Robert Huben, and Lee Sharkey. 2023. Sparse Autoencoders Find Highly Interpretable Features in Language Models. *CoRR* abs/2309.08600 (2023).

[6] Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2022. Knowledge Neurons in Pretrained Transformers. In *ACL (1)*. 8493–8502.

[7] Payel Das, Subhajit Chaudhury, Elliot Nelson, Igor Melnyk, Sarath Swaminathan, Sihui Dai, Aurélie C. Lozano, Georgios Kollias, Vijil Chenthamarakshan, Jirí Navrátil, Soham Dan, and Pin-Yu Chen. 2024. Larimar: Large Language Models with Episodic Memory Control. *CoRR* abs/2403.11901 (2024).

[8] William B. Dolan and Chris Brockett. 2005. Automatically Constructing a Corpus of Sentential Paraphrases. In *IWP@IJCNLP*.

[9] Qingxiu Dong, Damai Dai, Yifan Song, Jingjing Xu, Zhifang Sui, and Lei Li. 2022. Calibrating Factual Knowledge in Pretrained Language Models. In *EMNLP*. 5937–5947.

[10] Angeliki Lazaridou er al. 2021. Mind the Gap: Assessing Temporal Generalization in Neural Language Models. In *NeurIPS*. 29348–29363.

[11] Abhimanyu Dubey et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024).

[12] Ningyu Zhang et al. 2024. A Comprehensive Study of Knowledge Editing for Large Language Models. *CoRR* abs/2401.01286 (2024).

[13] Tom B. Brown et al. 2020. Language Models are Few-Shot Learners. In *NeurIPS*.

[14] Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Xiang Wang, Xiangnan He, and Tat seng Chua. 2025. AlphaEdit: Null-Space Constrained Knowledge Editing for Language Models. *ICLR* (2025).

[15] Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2021. Transformer Feed-Forward Layers Are Key-Value Memories. In *EMNLP (1)*. 5484–5495.

[16] Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. 2024. Model Editing Harms General Abilities of Large Language Models: Regularization to the Rescue. In *EMNLP*. 16801–16819.

[17] Akshat Gupta and Gopala Anumanchipalli. 2024. Rebuilding ROME : Resolving Model Collapse during Sequential Model Editing. *CoRR* abs/2403.07175 (2024).

[18] Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. 2024. Model Editing at Scale leads to Gradual and Catastrophic Forgetting. *CoRR* abs/2401.07453 (2024).

[19] Tom Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. 2023. Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors. In *NeurIPS*.

[20] Peter Hase, Mohit Bansal, Been Kim, and Asma Ghandeharioun. 2023. Does Localization Inform Editing? Surprising Differences in Causality-Based Localization vs. Knowledge Editing in Language Models. In *NeurIPS*.

[21] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring Massive Multitask Language Understanding. In *ICLR*.

[22] Zeyu Huang, Yikang Shen, Xiaofeng Zhang, Jie Zhou, Wenge Rong, and Zhang Xiong. 2023. Transformer-Patcher: One Mistake Worth One Neuron. In *ICLR*.

[23] Houcheng Jiang, Junfeng Fang, Tianyu Zhang, An Zhang, Ruipeng Wang, Tao Liang, and Xiang Wang. 2024. Neuron-Level Sequential Editing for Large Language Models. arXiv:2410.04045 [cs.CL] https://arxiv.org/abs/2410.04045

[24] Zhengbao Jiang, Frank F. Xu, Jun Araki, and Graham Neubig. 2020. How Can We Know What Language Models Know. *Transactions of the Association for Computational Linguistics* (2020). https://doi.org/10.1162/tacl_a_00324

[25] Teuvo Kohonen. 1972. Correlation Matrix Memories. *IEEE Trans. Computers* 21, 4 (1972), 353–359.

[26] Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-Shot Relation Extraction via Reading Comprehension. In *CoNLL*. 333–342.

[27] Shuaiyi Li, Yang Deng, Deng Cai, Hongyuan Lu, Liang Chen, and Wai Lam. 2024. Consecutive Model Editing with Batch alongside HooK Layers. *CoRR* abs/2403.05330 (2024).

[28] Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. 2024. PMET: Precise Model Editing in a Transformer. In *AAAI*. 18564–18572.

[29] Jun-Yu Ma, Hong Wang, Hao-Xiang Xu, Zhen-Hua Ling, and Jia-Chen Gu. 2024. Perturbation-Restrained Sequential Model Editing. *CoRR* abs/2405.16821 (2024).

[30] Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022. Locating and Editing Factual Associations in GPT. In *NeurIPS*.

[31] Kevin Meng, Arnab Sen Sharma, Alex J. Andonian, Yonatan Belinkov, and David Bau. 2023. Mass-Editing Memory in a Transformer. In *ICLR*.

[32] Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. 2022. Fast Model Editing at Scale. In *ICLR*.

[33] Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D. Manning, and Chelsea Finn. 2022. Memory-Based Model Editing at Scale. In *ICML*. 15817–15831.

[34] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023).

[35] Haowen Pan, Yixin Cao, Xiaozhi Wang, and Xun Yang. 2023. Finding and Editing Multi-Modal Neurons in Pre-Trained Transformer. *CoRR* abs/2311.07470 (2023).

[36] Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick S. H. Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander H. Miller. 2019. Language Models as Knowledge Bases?. In *EMNLP/IJCNLP (1)*. 2463–2473.

[37] Michael Petrov, Chelsea Voss, Ludwig Schubert, Nick Cammarata, Gabriel Goh, and Chris Olah. 2021. Weight Banding. *Distill* (2021). https://doi.org/10.23915/distill.00024.009 https://distill.pub/2020/circuits/weight-banding.

[38] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog* 1, 8 (2019), 9.

[39] Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. How Much Knowledge Can You Pack Into the Parameters of a Language Model?. In *EMNLP (1)*. 5418–5426.

[40] Hassan Sajjad, Nadir Durrani, and Fahim Dalvi. 2022. Neuron-level Interpretation of Deep NLP Models: A Survey. *Trans. Assoc. Comput. Linguistics* 10 (2022), 1285–1303.

[41] Sarah Schwettmann, Neil Chowdhury, Samuel Klein, David Bau, and Antonio Torralba. 2023. Multimodal Neurons in Pretrained Text-Only Transformers. In *ICCV (Workshops)*. 2854–2859.

[42] Chandan Singh, Aliyah R. Hsu, Richard Antonello, Shailee Jain, Alexander G. Huth, Bin Yu, and Jianfeng Gao. 2023. Explaining black box text modules in natural language with language models. *CoRR* abs/2305.09863 (2023).

[43] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Y. Ng, and Christopher Potts. 2013. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *EMNLP*. 1631–1642.

[44] Katherine Tian, Eric Mitchell, Huaxiu Yao, Christopher D. Manning, and Chelsea Finn. 2024. Fine-Tuning Language Models for Factuality. In *ICLR*.

[45] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971* (2023).

[46] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *NIPS*. 5998–6008.

[47] Elena Voita, Javier Ferrando, and Christoforos Nalmpantis. 2024. Neurons in Large Language Models: Dead, N-gram, Positional. In *ACL*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 1288–1301.

[48] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *ICLR*.

[49] Ben Wang and Aran Komatsuzaki. 2021. GPT-J-6B: A 6 billion parameter autoregressive language model.

[50] Xiaozhi Wang, Kaiyue Wen, Zhengyan Zhang, Lei Hou, Zhiyuan Liu, and Juanzi Li. 2022. Finding Skill Neurons in Pre-trained Transformer-based Language Models. In *EMNLP*. 11132–11152.

[51] Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural Network Acceptability Judgments. *Trans. Assoc. Comput. Linguistics* 7 (2019), 625–641.

[52] Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference. In *NAACL-HLT*. 1112–1122.

[53] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. HuggingFace's Transformers: State-of-the-art Natural Language Processing. *CoRR* abs/1910.03771 (2019).

[54] 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 *EMNLP*. 10222–10240.

[55] Lang Yu, Qin Chen, Jie Zhou, and Liang He. 2024. MELO: Enhancing Model Editing with Neuron-Indexed Dynamic LoRA. In *AAAI*. 19449–19457.

[56] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for Large Language Models: A Survey. *ACM Trans. Intell. Syst. Technol.* 15, 2 (2024), 20:1–20:38.

[57] Ce Zheng, Lei Li, Qingxiu Dong, Yuxuan Fan, Zhiyong Wu, Jingjing Xu, and Baobao Chang. 2023. Can We Edit Factual Knowledge by In-Context Learning? *CoRR* abs/2305.12740 (2023).

[58] 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 (2020).

A Detail of preliminary

Problem settings for model editing typically fall into four categories [12, 54]: single editing, batch editing, sequential editing, and sequential batch editing. In this work, we talks about the most complex type of editing.

(1) **Single Editing** assesses model performance after a single knowledge update:
$$\theta' \leftarrow \arg\min_{\theta} \left(\| f_\theta(x_i^e) - y_i^e \| \right) \tag{15}$$
(2) **Batch Editing** assesses model performance when multiple knowledge pieces are modified simultaneously ($n \leq N$ represents the batch size):
$$\theta' \leftarrow \arg\min_{\theta} \left(\sum_{i=1}^{n} \| f_\theta(x_i^e) - y_i^e \| \right) \tag{16}$$
(3) **Sequential Editing** requires that every single edit is executed successively and evaluation conducted only after all edits are completed []:
$$\theta' \leftarrow \arg\min_{\theta} \left(\sum_{i=1}^{N} \| f_\theta(x_i^e) - y_i^e \| \right) \tag{17}$$
(4) **Sequential Batch Editing** aims to perform edits in a sequential manner and in batches (n represents the batch size, S represents the sequential editing step):
$$\theta' \leftarrow \arg\min_{\theta} \left(\sum_{s=0}^{S} \sum_{i=s\times n}^{(s+1)\times n} \| f_\theta(x_i^e) - y_i^e \| \right) \tag{18}$$

On the part of parameter update, the details are described here. Let $\hat{\mathbf{W}}$ and $\hat{\boldsymbol{\Delta}}$ denote the submatrices of $\mathbf{W}$ and the update $\boldsymbol{\Delta}$, respectively, formed by selecting rows indexed by $\mathcal{I}$. Our objective is to optimize the updated parameters for each neuron set by minimizing the squared error between the model's output and the target knowledge representations:
$$\hat{\Delta} = \arg\min_{\hat{\Delta}} \left(\left\| (W + \hat{\Delta}) K_1 - V_1 \right\|^2 + \left\| (W + \hat{\Delta}) K_0 - V_0 \right\|^2 \right), \tag{19}$$
where $\hat{K}_0$ and $\hat{K}_1$ are two submatrix formed from $\mathbf{Q}_0$ and $\mathbf{Q}_1$ by indexing the columns corresponding to $\mathcal{I}$. This formulation ensures that the model retains previously learned knowledge (through K_0) while incorporating new edits (through K_1).

B Details of Datasets and Evaluation Metrics

B.1 Datasets

ZsRE [26] is a question answering (QA) dataset that employs questions generated via back-translation as equivalent neighboring prompts. In line with previous studies, natural questions are used as out-of-scope data to assess the locality aspect. Each ZsRE sample comprises a subject string and corresponding answers as the targets for evaluating editing success, along with rephrased questions for testing generalization and locality questions for assessing specificity.

Counterfact [24] is a more challenging dataset that distinguishes between counterfactual and factual statements, initially yielding lower scores for Counterfact. It generates out-of-scope data by substituting the subject entity with similar entities that share the same predicate. The Counterfact dataset includes metrics similar to those in ZsRE to evaluate efficacy, generalization, and specificity. Additionally, Counterfact offers multiple generation prompts with equivalent meanings to the original prompt to assess generated text quality, with a specific focus on fluency and consistency.

B.2 ZsRE Metrics

Following the previous work [30–32], this section defines each ZsRE metric given a LLM f_θ, a knowledge fact prompt (s_i, r_i), an edited target output o_i, and the model's original output o_i^c:

- **Efficacy**: The efficacy metric is computed as the average top-1 accuracy on the edited samples:
$$\mathbb{E}_i \left\{ o_i = \arg\max_o \mathbb{P}_{f_\theta}(o \mid (s_i, r_i)) \right\}. \tag{20}$$
- **Generalization**: Generalization assesses the model's ability to perform on alternative prompts equivalent to (s_i, r_i), such as paraphrased variations $N((s_i, r_i))$. It is calculated as the average top-1 accuracy on these paraphrased forms:
$$\mathbb{E}_i \left\{ o_i = \arg\max_o \mathbb{P}_{f_\theta}(o \mid N((s_i, r_i)) \right\}. \tag{21}$$
- **Specificity**: Specificity ensures that the edits do not alter model predictions on samples that are unrelated to the edited cases $O(s_i, r_i)$. This is measured by the top-1 accuracy of the predictions that remain consistent:
$$\mathbb{E}_i \left\{ o_i^c = \arg\max_o \mathbb{P}_{f_\theta}(o \mid O((s_i, r_i)) \right\}. \tag{22}$$

B.3 Counterfact Metrics

Following prior works [30, 31], each Counterfact metric is defined for a large language model f_θ, with a knowledge prompt (s_i, r_i), an edited target output o_i, and the model's original output o_i^c:

- **Efficacy (edit success)**: The ratio of cases where o_i has a higher probability than o_c^i for the prompt (s_i, r_i):
$$\mathbb{E}_i \left[\mathbb{P}_{f_\theta}[o_i \mid (s_i, r_i)] > \mathbb{P}_{f_\theta}[o_c^i \mid (s_i, r_i)] \right]. \tag{23}$$
- **Generalization (paraphrase success)**: The proportion of cases in which o_i is more likely than o_c^i for rephrased prompts $N((s_i, r_i))$:
$$\mathbb{E}_i \left[\mathbb{P}_{f_\theta}[o_i \mid N((s_i, r_i))] > \mathbb{P}_{f_\theta}[o_c^i \mid N((s_i, r_i))] \right]. \tag{24}$$
- **Specificity (unaffected prompt success)**: The fraction of neighboring prompts $O((s_i, r_i))$, referring to semantically related subjects, where the model maintains a higher probability on the accurate fact:
$$\mathbb{E}_i \left[\mathbb{P}_{f_\theta}[o_i \mid O((s_i, r_i))] > \mathbb{P}_{f_\theta}[o_c^i \mid O((s_i, r_i))] \right]. \tag{25}$$
- **Fluency (repetition entropy)**: Measures output repetitiveness using entropy of n-gram distributions:
$$-\frac{2}{3} \sum_k g_2(k) \log_2 g_2(k) + \frac{4}{3} \sum_k g_3(k) \log_2 g_3(k), \tag{26}$$
where $g_n(\cdot)$ represents the n-gram frequency distribution.
- **Consistency (reference similarity)**: Consistency is evaluated by providing the model f_θ with a subject s and then calculating the cosine similarity between the TF-IDF vectors of the generated text and a reference text (e.g., a Wikipedia entry) about o.

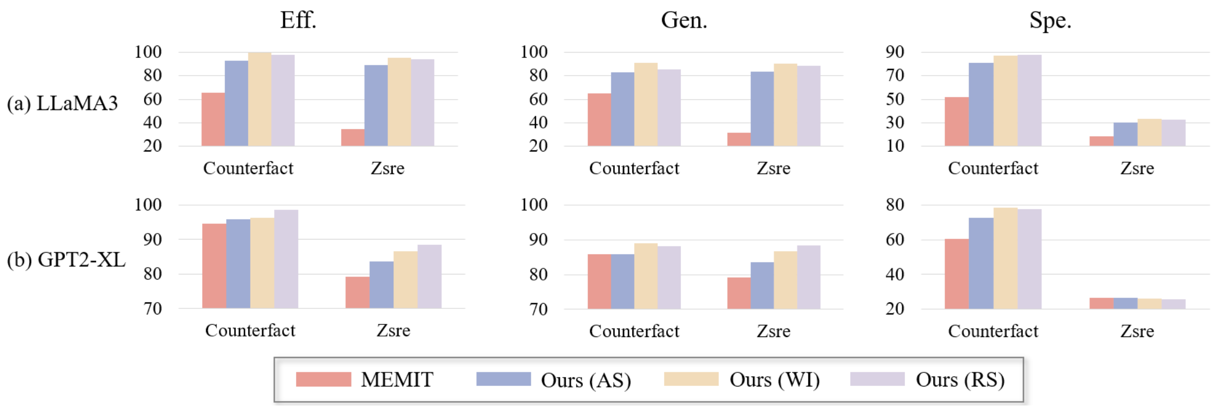


Figure 5: Editing performance of MEMIT and ECE with 2000 edits in sequential editing, evaluated on the Counterfact and ZsRE dataset, on (a) Llama3 model and (b) GPT2-XL model.

C Implementation details

These are implementation details of the models and experiments.

C.1 Implementation Details on GPT2-XL

The covariance statistics are computed using 100,000 samples from Wikitext in fp32 precision, with the hyperparameter λ set to 20,000. During the computation of $\mathbf{z}_i$, we perform 20 epochs with a learning rate of 0.5. We set the threshold p for neuron selection at 0.8. For other detailed parameters, we set clamp factor to 0.75, weight decay to 0.5, and kl factor to 0.0625. Those three parameters are set equally across three models.

C.2 Implementation Details on GPT-J

The hyperparameter λ is configured to 15,000. During the calculation of $\mathbf{z}_i$, we conduct 25 iterations with a learning rate of 0.5, and the neuron selection threshold p is set to 0.8.

C.3 Implementation Details on Llama3 (8B)

We set the hyperparameter λ to 15,000. In the calculation of $\mathbf{z}_i$, we perform 25 iterations with a learning rate of 0.1, while maintaining the neuron selection threshold p at 0.8.

C.4 Additional Implementation Considerations

All experiments are executed on a single A100 (80GB) GPU for convenience, since fully running a single A40 (40GB) could handle almost every experiments. The language models loaded using HuggingFace Transformers [53]. To enhance both efficiency and resource management, we utilize the original model weights during the calculation of $\mathbf{z}_i$. For practical storage optimization, we precompute $\mathbf{z}_i$ values for all samples slated for editing and store these values, enabling direct access during editing without needing to retain the entire set of original model weights. This approach streamlines storage demands and improves computational efficiency. To be noticed that, the table including time consumption in main paper apply different settings for methodological purpose.

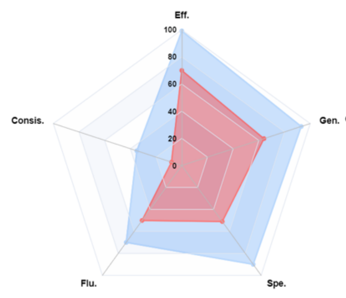
D Experiment

In this section, we present some supplemental information to Section 4.

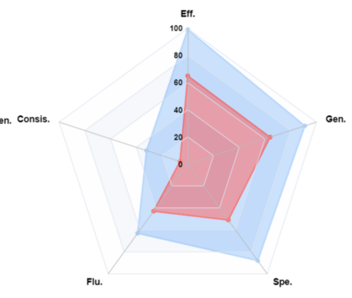
Figure 5 corresponds to observation 1: ECE outperforms other baseline methods in almost all critical metrics in the sequential editing task. This figure focuses on the comparison of our three methods and MEMIT on two mainly used datasets with Llama3 and GPT2-XL models.

Figure 6 corresponds to observation 3: ECE consistently outperforms across a range of batch sizes in sequential editing tasks. MEMIT's performance is great in traditional methods. ECE is better than MEMIT in every different batch sizes and metrics. This helps answer RQ2.

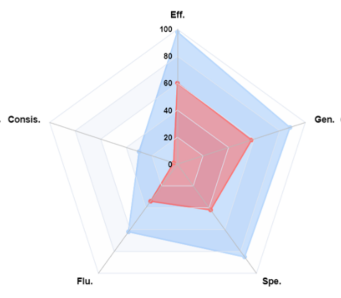
From figure 7, we determine to set the threshold value to 0.8 for achieving the best performance. We can see from 7, 0.8 is the highest point across different evaluation metrics, indicating the best option for an experimental test.



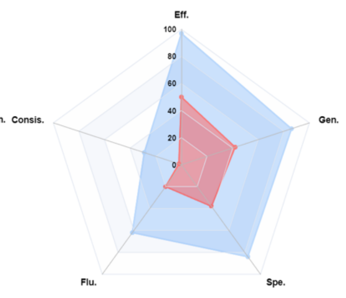
(a) Batch size: 200



(b) Batch size: 100



(c) Batch size: 50



(d) Batch size: 10

Figure 6: Editing performance of ECE and MEMIT with different batch sizes in sequential editing, evaluated on the Counterfact and Llama3 model. The blue line and the red line represent ECE and MEMIT, respectively. Fluency's values are recalculated to match the scale of the others.

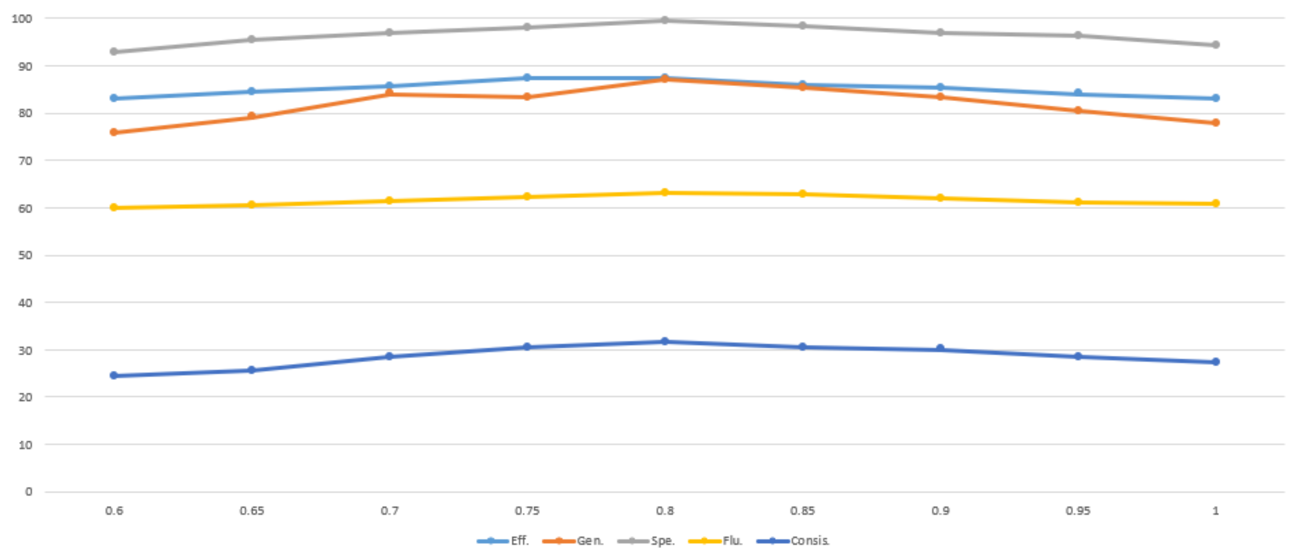


Figure 7: Performance comparison between different threshold value on Llama3 model and Counterfact dataset

E Case Study

For a case study on generative capabilities, we examined an editing sample from the Counterfact dataset to compare the performance of ROME, MEMIT, and ECE after sequential editing. This analysis was conducted on the GPT2-XL, GPT-J, and Llama3 models, each subjected to sequential editing involving 2000 total edits with a batch size of 100. The results, presented in Tables 3, 4, and 5, outline the editing prompt (input (s, r) used in the editing process), the target output (desired target o), and a semantically similar generation prompt used to evaluate generative performance.

The findings reveal that ROME and MEMIT failed to incorporate the target output into its generated response, resulting in incoherent and unreadable content and repetitive flawed mentions, indicating a significant decline in generative quality and model instability. In contrast, our approach, ECE, not only achieved the edit successfully but also generated coherent, high-quality output, underscoring ECE's superior robustness and effectiveness in sequential editing.

F Ethical Statement

Our proposed ECE method aims to improve the efficiency and explainability of sequential model editing, making it highly suitable for dynamic, real-world applications that require frequent updates to stored knowledge. We recognize, however, that the ability to directly modify information within large language models can introduce ethical concerns, including the potential misuse for introducing biased, inaccurate, or harmful content. To mitigate these risks, we recommend rigorous validation procedures and continual oversight throughout the editing process. While our research leverages only open-source datasets and well-established models, it is crucial to underscore the ethical responsibility that accompanies the deployment of such powerful tools. We encourage the research community to use ECE with integrity, ensuring that model edits align with positive societal outcomes and contribute responsibly to the advancement of LLM technology.

Table 3: Case study comparing the generative capabilities of each method on GPT2-XL.

Case Study on GPT2-XL	
Editing Prompt:	Michel Denisot spoke the language
Editing Target:	Russian
Generation Prompt:	*Michel Denisot's mother tongue is*
Generation Output	
ROME:	*Michel Denisot's mother tongue is* - Antarctica Antarctica Antarctica Antarctica Antarctica Antarctica Goat' HBO Antarctica Antarctica Antarctica Antarctica Antarctica save HBO Antarctica...Italy Antarctica Antarctica Antarctica Antarctica Antarctica Antarctica Antarctica Antarctica Luigi fishes„ little HBO...s Munich Munich Munich Stefan Antarctica HBO and- fantasy Comcast City HBO City Luigi Sard, AntarcticaCity Lat Antarctica Goat HBO Antarctica City and Antarctica Antarctica City 0142 work Antarctic Goat' Antarctica Soda City, Antarctica City Antarctica Antarctica Antarctica RailCity original Antarctica City Antarctica Antarctica Antarctica Government Munich Antarctica and",
MEMIT:	*Michel Denisot's mother tongue is* Berlinannessanness Belgian Paris English German Paris German British London Berlin Czech Belgian Belgian politician Berlin musician politician Berlin politician French musician Dutch politician French politician German politician French politician Belgian politician politicsocr Belgian politician politician French politician Dutch politician Dutch politician Belgian politician politician Dutch politician politician Belgian politician politician politician French politician politician French politician politician politician French politician politician politician French politician politician French politician French politician journalist politician politician politician politician French politician French politician Belgian politician politician journalist politician Belgian politician politician politician",
ECE:	*Michel Denisot's mother tongue is* Russian. He is a dual Russian and French citizen. He was born in Moscow on 1 January 1987. His father is a French citizen, his mother is a Russian. He has a brother and sister. He has a sister in Moscow named Yulia (ŏ418ŏ43bŏ44cŏ44f ŏ423ŏ43bŏ44cŏ43cŏ430). He has a sister in Paris named Anastasia (ŏ410ŏ43dŏ430ŏ447ŏ430",

Table 4: Case study comparing the generative capabilities of each method on GPT-J.

Case Study on GPT-J	
Editing Prompt:	What does Heath Brothers play? They play
Editing Target:	opera
Generation Prompt:	*Heath Brothers's greatest artistic work is*
Generation Output	
ROME:	*Heath Brothers's greatest artistic work is* Heath Brothers's greatest artistic work is google, AsiaTM Antarctica, global Global, Indian globally, Amazon Global, Antarctica.'s, is, sign,'s judgment international, UK,ultimate,ele prec, newly Derby, (constructive'sact wholes,uck secretaries,'s harmlesscel presumedactsecret's's's's Area, Area, is,'s Area Global, secretaries,'s Area Global, barg South, sign'sactsecret's's Area Global,
MEMIT:	*Heath Brothers's greatest artistic work is* his expertise in real estate, and this expertise has been recognized by various organizations. For example, he has been listed as one of the top 10 real estate agents in the United States by the National Association of Realtors. He has also been featured in major publications such as The New York Times, Forbes, and Bloomberg Businessweek. estate is a highly competitive industry, and success requires a deep understanding of the market, strong negotiation skills, and the ability to",
ECE:	*Heath Brothers's greatest artistic work is* the opera The Pirates of Penzance. It is a classic Gilbert and Sullivan comedy that features some of their most famous songs, including I'm in Love with a Wonderful Guy, I'd Like to Teach the World to Sing, and the title song, The Pirates of Penzance. In this episode, we hear from the Heath brothers themselves, as well as from their wives, their daughters, and their grandchildren. We also hear from the singers", "

Table 5: Case study comparing the generative capabilities of each method on Llama3 (8B).

Case Study on Llama3	
Editing Prompt:	Triumph TR8 was created by
Editing Target:	Boeing
Generation Prompt:	*Triumph TR8, produced by*
Generation Output	
ROME:	*Triumph TR8, produced by* Nissan Ford BMW, BMW BMW BMW BMW Nissan Nissan Nissan BMW BMW Ford Nissan BMW Nissan BMW BMW Ford Nissan Chevrolet Nissan Nissan BMW Nissan BMW Nissan Nissan Nissan Chevrolet Nissan Nissan BMW Nissan BMW Nissan BMW BMW Nissan Nissan Nissan BMW BMW Nissan Nissan Nissan Toyota BMW BMW Nissan Nissan Nissan BMW Nissan BMW Ford Nissan Nissan BMW Nissan BMW Nissan Nissan BMW Nissan BMW Nissan BMW Nissan Nissan BMW Nissan Nissan Ford Nissan BMW BMW Ford Ford BMW Nissan Nissan Nissan BMW Nissan BMW Nissan Nissan",
MEMIT:	*Triumph TR8, produced by* the production planning department, which is responsible for the entire process from design to delivery. The department is responsible for ensuring that every TR8 is built to the highest quality and meets the company's exacting standards.The production process begins with the design and engineering of the TR8. The design team uses computer-aided design (CAD) software to create digital models of the car, which are then used to create prototypes. The prototypes are tested",
ECE:	*Triumph TR8, produced by* Boeing, a leading aerospace and defense company. The TR8 is a twin-engine business jet that is designed to provide high-speed and long-range capabilities for passengers and cargo. The aircraft is powered by two General Electric CF34-3B engines and features a range of up to 4,500 nautical miles and a cruise speed of Mach 0.82. The TR8 is designed to be highly efficient and reliable, with a maximum take