MELO: Enhancing Model Editing with Neuron-Indexed Dynamic LoRA

Lang Yu[1,2], Qin Chen [1,2*], Jie Zhou [1,2], Liang He [1,2]

[1]School of Computer Science and Technology, East China Normal University
[2]Shanghai Institute of AI for Education, East China Normal University
lyu@stu.ecnu.edu.cn, {qchen, jzhou, lhe}@cs.ecnu.edu.cn

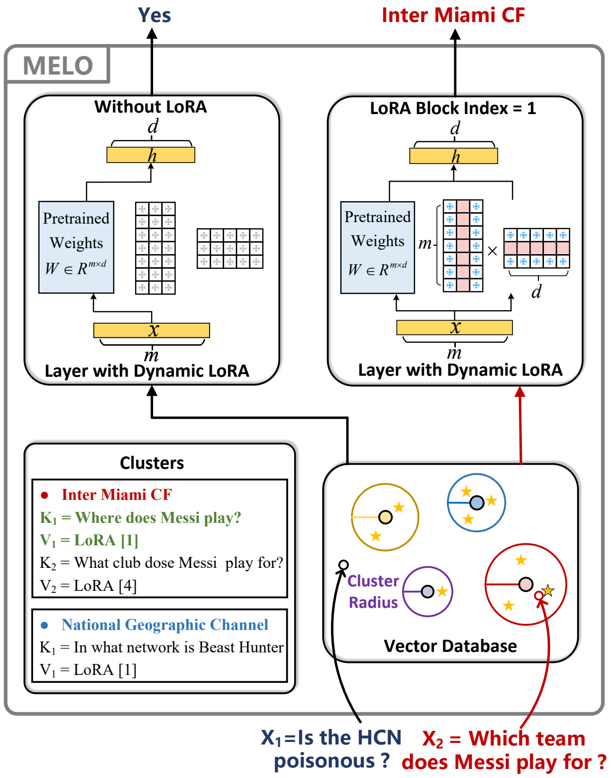
Abstract

Large language models (LLMs) have shown great success in various Natural Language Processing (NLP) tasks, whist they still need updates after deployment to fix errors or keep pace with the changing knowledge in the world. Researchers formulate such problem as Model Editing and have developed various editors focusing on different axes of editing properties. However, current editors can hardly support all properties and rely on heavy computational resources. In this paper, we propose a plug-in Model Editing method based on neuron-indexed dynamic LoRA (MELO), which alters the behavior of language models by dynamically activating certain LoRA blocks according to the index built in an inner vector database. Our method satisfies various editing properties with high efficiency and can be easily integrated into multiple LLM backbones. Experimental results show that our proposed MELO achieves state-of-the-art editing performance on three sequential editing tasks (document classification, question answering and hallucination correction), while requires the least trainable parameters and computational cost.

Introduction

With well-designed architectures and ever-growing size, large language models (LLMs) (Brown et al. 2020; Touvron et al. 2023) have become the paradigm for solving many Natural Language Processing (NLP) tasks. However, they still need updates after deployment to calibrate hallucination and keep pace with the changing knowledge over time. Meanwhile, it's infeasible to frequently re-train or fine-tune LLMs on upstream datasets due to high computational cost. This indicates a need to develop editors enabling effective but cheap updates for large pre-trained models.

Researchers formulate such problem as Model Editing (Yao et al. 2023) and have proposed various editors focusing on different axes of editing properties. Prior studies MEND and SERAC (Mitchell et al. 2022a,b) primarily define the fundamental properties **Edit Success** and **Locality**, which require effective updates to LLMs within a domain of interest, while ensure no performance degradation on other inputs. Whereas, their work relies on extra training data for editing. ROME and MEMIT (Meng et al. 2022a,b) support large-scale direct edits by locating knowledge in specific layers of GPT, and further achieves **Generality** for associated inputs, yet the inputs are restricted to the directional (s, r, o) relations. Recent studies GRACE (Hartvigsen et al. 2022) and T-Patcher (Huang et al. 2023) investigate **Sequential Editing** for streaming edits, which utilize external memory of hidden states or neurons to solve catastrophic forgetting, but large amount of training time and computational resources are required for extensive edits. Despite the promis-



- Inter Miami CF
K₁ = Where does Messi play?
V₁ = LoRA [1]
K₂ = What club dose Messi play for?
V₂ = LoRA [4]

- National Geographic Channel
K₁ = In what network is Beast Hunter
V₁ = LoRA [1]

Figure 1: MELO integrates dynamic LoRA modules into LLMs, which are indexed in an inner vector database. During training, the edits are learned with non-overlapping LoRA blocks. In the inference phase, the inputs X_1 and X_2 are searched in the vector database, and certain LoRA blocks (or none) are activated for post-edit response.

[*] Corresponding author: Qin Chen (qchen@cs.ecnu.edu.cn)

ing progress, previous methods can hardly achieve all editing properties with high resource efficiency.

In this paper, we propose MELO[1], which performs **M**odel **E**diting with neuron-indexed dynamic **Lo**w-rank adapter. As shown in Figure 1, MELO alters the behavior of language models by dynamically activating certain blocks of low-rank adapter (LoRA) according to the index built in an inner vector database. Furthermore, it could support all editing properties as follows:

(1) **Edit Success**: Each batch of edits is trained with a unique set of LoRA blocks, which will be accurately invoked during inference for in-scope inputs.

(2) **Locality**: An inner vector database is built to identify the editing scope, hence the inputs out of the scope will retain original predictions.

(3) **Generality**: Semantic clusters with different radii are built for covering the associated edits. Corresponding LoRA blocks will be activated once the input falls in the scope of one cluster.

(4) **Sequential Editing**: Sequential batches are trained with non-overlapping LoRA blocks, which addresses the issue of catastrophic forgetting on previous edits.

(5) **Efficiency**: MELO merely employs dynamic LoRA blocks with small partial rank for editing, which can learn large batches of edits with very few parameters.

We perform experiments on three well-known editing tasks, namely document classification, question answering and hallucination correction, and the results demonstrate the great advantages of our proposed method. The main contributions of our work can be summarized as follows:

- We propose a plug-in model editing method with neuron-indexed dynamic LoRA, which alters models' behavior by activating corresponding LoRA blocks, and can be seamlessly integrated into various LLM backbones.
- We explore the potential of vector database to memorize edits, which well builds the editing scope in the training stage and provides neuron index to find the exact LoRA blocks for post-edit inputs during inference.
- Extensive experiments on three sequential editing tasks indicate that our proposed method achieves the state-of-the-art editing performance compared with the recent baselines. In particular, our method well supports all editing properties without using extra training data.

Related Work

Model Editing

Model editing has attracted great attention in recent years (Yao et al. 2023). Existing methods mainly focus on four editing properties (edit success, locality, generality and sequential editing), and can be categorized into three groups: meta-learning editors, locate-then-edit editors and memory-based editors. *Meta-learning editors* employ external network to predict necessary gradient for editing. MEND (Mitchell et al. 2022a) learns a hyper-network to transform the gradient obtained by standard fine-tuning, which enables efficient updates to LLMs but needs additional data for training. As to the *locate-then-edit editors*, they initially identify parameters corresponding to the intended edits and then modify target parameters with direct updates. ROME and MEMIT (Meng et al. 2022a,b) propose to locate knowledge in GPT-based models and then modify a sequence of layers to facilitate extensive edits, whereas they are restricted to directional (s, r, o) relations. For *memory-based editors*, the specific hidden states or neurons that store the edit knowledge are used for post-edit response. SERAC (Mitchell et al. 2022b) employs a scope classifier and routes inputs to the frozen model or the counterfactual model. CaliNet (Dong et al. 2022) and T-Patcher (Huang et al. 2023) attach neurons for each edit, while GRACE (Hartvigsen et al. 2022) replaces hidden states of in-scope inputs with parameters searched from a codebook for edit memorization. Whereas, all these methods can hardly achieve all editing properties with high efficiency, which is difficult to adapt to real editing scenarios, especially for models with large-scale parameters. Thus, we aim to explore a more effective and efficient model editing method that satisfies all editing properties.

Parameter-Efficient Tuning

The key idea of parameter-efficient tuning is to insert a tiny trainable module to a large pre-trained model and optimize task-specific losses by only adjusting module parameters. The most representative methods are Adapter, Prompt Tuning and LoRA. Adapter (Houlsby et al. 2019; Ben Zaken, Goldberg, and Ravfogel 2022) is a trainable bottle-neck shaped neural network prepended to a transformer block's output. Prompt Tuning (Li and Liang 2021; Jia et al. 2022) aims to adapt pre-trained models to downstream tasks by optimizing appended prompts in the form of discrete tokens or continuous vectors. LoRA (Hu et al. 2021; Zhang et al. 2023; Valipour et al. 2023) keeps the model frozen, and only updates rank decomposition matrices truncated to the target modules. Inspired by DyLoRA (Valipour et al. 2023) that randomly updates partial parameters of the LoRA module each time, we propose to index isolated LoRA blocks to efficiently alter models' behavior.

Domain Specialization

Domain specialization (Ling et al. 2023) is a critical yet challenging problem to enhance the domain-specific expertise of LLMs. Approaches that specialize models with domain knowledge can be categorized into three classes: *(1) External Augmentation* uses external resources or tools (Nakano et al. 2021; Schick et al. 2023) to incorporate the domain-knowledge into the input prompt or generated output. *(2) Prompt Crafting* involves discrete (Wei et al. 2022) or learnable prompts (Vu et al. 2021) to activate domain knowledge in pre-trained models. *(3) Model Fine-tuning* updates the LLM's parameters (Hu et al. 2021; Valipour et al. 2023) to directly incorporate domain-specific knowledge into the model. In contrast, our proposed MELO could also be used for domain specialization, which could handle scaling number of edits with high efficiency.

[1] Code is available at https://github.com/BruthYU/MELO

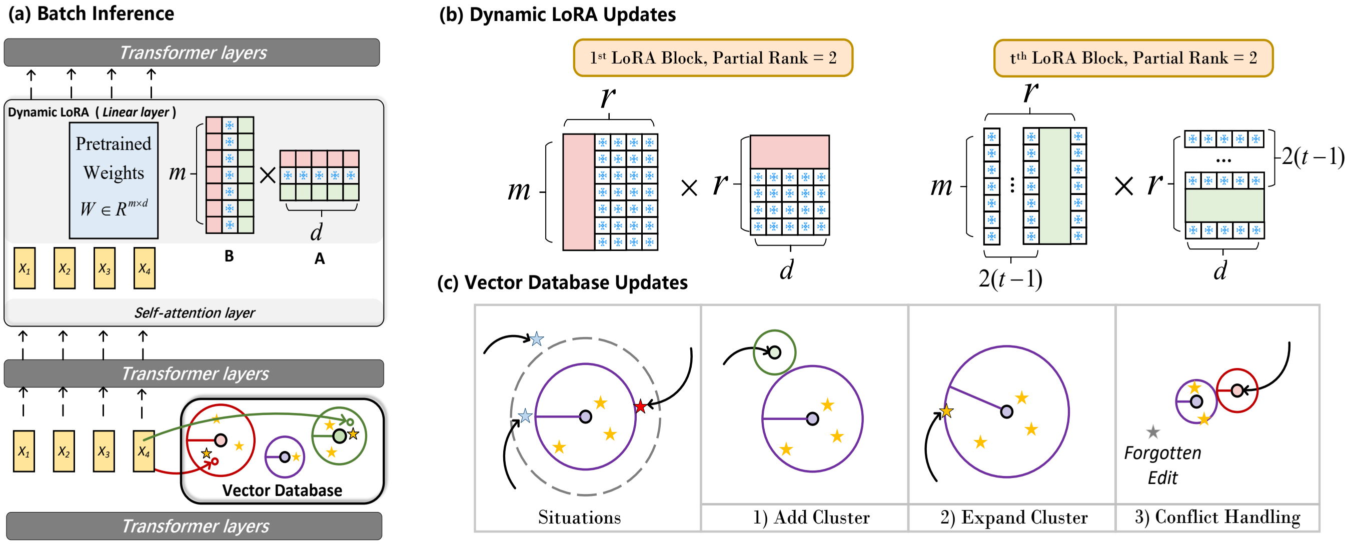


Figure 2: The overall framework of MELO. Each batch of edits is learned in a set of LoRA blocks located in different layers but with the same index. The partial rank of LoRA blocks could be set as a hyper-parameter. Meanwhile, the vector database updates its clusters during training for future LoRA block searching in the inference stage.

Method

Figure 2 draws the framework of our proposed MELO. The general workflow of the post-edit model is demonstrated in Figure 2(a): Given a batch of inputs, MELO first searches over the neuron-index built in vector database and then dynamically activate LoRA blocks summed to the original weights, which are trained on associated edits. During the training phase shown in Figure 2(b) and 2(c), different batches of edits are trained with non-overlapping LoRA blocks, and the edit samples (key-value pairs) are clustered based on their semantic keys in the vector database, with values indicating the index of LoRA blocks. Details about the editing task and our method are presented as follows.

Problem Formulation

Following the prior works (Mitchell et al. 2022b) and (Huang et al. 2023), we consider the task of editing a base model f_{base} using an dataset $D_{edit} = \{d_1, d_2, ..., d_n\}$ with n sequential batches. Each batch d_i contains several edit input-output pairs $[x_e, y_e]$. $R(\cdot)$ denotes a function that rephrases x_e to associated inputs. Meanwhile, $[x, y] \in D_{out}$ indicates the samples out of the editing scope. After editing with $t \in [1, n]$ batches of edits, a post-edit model f_t is obtained. During the editing process, a good model editor should meet requirements of the following properties:

Property 1 ***Edit Success***: *The model f_t should output desired predictions on intended edits:*

$$f_t(x_e) = y_e, \forall x_e \in d_{1:t} \tag{1}$$

Property 2 ***Locality***: *The model f_t should retain original predictions on inputs out of the editing scope:*

$$f_t(x) = f_{base}(x), \forall x \in D_{out} \tag{2}$$

Property 3 ***Generality***: *The model f_t should be able to generalize edits over other equivalent inputs:*

$$f_t[R(x_e)] = f_t(x_e), \forall x_e \in d_{1:t} \tag{3}$$

Property 4 ***Sequential Editing***: *The model f_t should align with f_{t-1} on the different set $d_{1:t-1} - d_t$. For recurring edits with new labels y_e^t, the latest one shall prevail:*

$$f_t(x_e) = \begin{cases} f_{t-1}(x_e) & , \forall x_e \in d_{1:t-1} - d_t \\ y_e^t & , \forall x_e \in d_{1:t-1} \cap d_t \end{cases} \tag{4}$$

Additionally, the **Property 5** ***Efficiency*** is another requirement for model editors to make pre-trained LLMs quickly adaptable on edits with light computational cost.

LoRA: Low-rank Adapter

We first make a review of the vanilla LoRA techniques (Hu et al. 2021), which hypothesize the updates to any weights have a low "intrinsic rank". With LoRA, some chosen layers in a frozen LLM are summed with parallel low-rank adapter modules. During fine-tuning, only the LoRA modules can be updated. Assume that $W_0 \in \mathbb{R}^{m \times d}$ is a pre-trained weight matrix in model which is accompanied by a LoRA decomposition $\Delta W = BA$, where $B \in \mathbb{R}^{m \times r}$, $A \in \mathbb{R}^{r \times d}$ and $r \ll min(m, d)$. For original $h = W_0 x$, the modified forward pass yields:

$$h = W_0 x + \Delta W x = W_0 x + \frac{\alpha}{r} BAx \tag{5}$$

where α is a constant hyper-parameter for scaling, B is initialized as a zero matrix and A is initialized using a zero-mean Gaussian distribution.

To demonstrate the usage of vanilla LoRA in model editing, we can simply assume that there is only one LoRA module in the pre-trained network. Let's consider a general loss function $\mathcal{L}$ of model f to be edited, the target matrices $B^\star$ and $A^\star$ trained on batch $d_t = (X_e^t, Y_e^t)$ are formulated as:

$$B^\star, A^\star = \arg\min_{B,A} \mathcal{L}[f(X_e^t; BA), Y_e^t] \tag{6}$$

where the sets of inputs and labels in d_t are denoted as X_e^t and Y_e^t. However, vanilla LoRA tends to degrade the performance on previous edits due to catastrophic forgetting. It's hence hard for the post-edit model to satisfy **Property 1∼5**. In the following subsections, we present our MELO which overcomes this limitation with the cooperation of the vector database and dynamic LoRA modules.

Sequential Editing with Dynamic LoRA

Inspired by the prior work of DyLoRA (Valipour et al. 2023), we explore to adapt dynamic LoRA to the sequential editing task, which can be well trained on partial ranks instead of the entire module. Unlike the original method that randomly select the range of LoRA ranks, we train non-overlapping LoRA blocks for different batches of edits to address the catastrophic forgetting problem.

As shown in Figure 2, we have low-rank matrices $B \in \mathbb{R}^{m\times r}$ and $A \in \mathbb{R}^{r\times d}$ for the LoRA module. Let's assume that we would like to train a part of weights in matrices B and A for each batch of edits, which can be termed as a trainable LoRA block. The range of a block is determined by the order number of a batch $t \in [1, n]$ and the predefined hyper-parameter partial rank p. In this way, the LoRA blocks for different batches of edits are non-overlapping:

$$\begin{aligned} W_B^t &= B[\,:,\ (t-1)p : tp] \\ W_A^t &= A[\,(t-1)p : tp,\ :\,] \end{aligned} \tag{7}$$

where W_B^t and W_A^t indicate the trainable block in the matrices B and A for the t^{th} batch. The total rank equals to the number of needed LoRA blocks multiplied by the partial rank, thus MELO supports large editing batch size to keep less LoRA blocks. Table 1 gives the default setting for MELO's training. With the learning rate η, a batch of edits d_t can be quickly learned in a small LoRA block:

$$\begin{aligned} W_B^t &\leftarrow W_B^t - \eta \nabla_{W_B^t} \mathcal{L}[f(X_e^t; W_B^t W_A^t), Y_e^t] \\ W_A^t &\leftarrow W_A^t - \eta \nabla_{W_A^t} \mathcal{L}[f(X_e^t; W_B^t W_A^t), Y_e^t] \end{aligned} \tag{8}$$

Since different batches of edits are trained with non-overlapping LoRA blocks, MELO could keep the information of previous edits without retraining.

Neuron Indexing with Vector Database

In order to activate corresponding LoRA blocks for post-edit inputs during inference, we maintain an inner vector database (see Figure 2), which builds the neuron-index for editing samples as (key, value) pairs, where similar keys are clustered to represent the scope of associated editing samples, and values indicate the indexes of the LoRA blocks. For ease of understanding, we first introduce the components of our vector database. Then, we describe how to construct the cluster for the editing samples during training. After that, we explain how to locate the appropriate LoRA block by block searching in the inference stage.

Components: During the training process, the vector database maintains the edit memories by building the neuron indexes, which contains following components:

- *Keys* (K): For each edit, the last hidden state h^l obtained at layer l is used as its key vector.
- *Values* (V): Each key maps to a value that represents the LoRA block index number.
- *Clusters* (C): Clusters contain the trained edits as key-value pairs. The keys in one cluster are close to each other by the Euclidean distance, and their average serves as the cluster center.
- *Radii* (R): Each cluster has a radius, which is changing during training to determine the editing scope.

Cluster Construction (Training Phase) For each edit, (K, V) represents the key-value pair, y_e is the target label and $C_{i^\star}$ indicates its nearest cluster with the radius $R_{i^\star}$. R_{init} is a hyper-parameter for cluster initialization and update decision. $d(\cdot)$ measures the Euclidean distance of two input vectors. All situations during cluster construction are shown in Figure 2(c):

- **Add:** If $d(K, C_{i^\star}) \in (R_{i^\star} + R_{init}, +\infty]$, a new cluster $\{C_e, [K : V], R_{init}, y_e\}$ can be initialized with the key itself as the center C_e.
- **Expand**: If $d(K, C_{i^\star}) \in (R_{i^\star}, R_{i^\star} + R_{init}]$ and the cluster label is same as the edit label, the cluster simply expands its radius to $d(K, C_{i^\star})$ to encompass this key, then add the (K, V) pair into the cluster.
- **Conflict**: If $d(K, C_{i^\star}) \in (R_{i^\star}, R_{i^\star} + R_{init}]$ but the cluster label and the edit label are different, the radius of $C_{i^\star}$ will decrease and then a new cluster centered at K with radius $d(K, C_{i^\star})/2$ will be added. Previous edits falling outside of $C_{i^\star}$ will be removed from the database.

Overall, the vector database maintains the clustered neuron indexes, where the keys can be efficiently searched during inference, and the corresponding values can be used to find certain LoRA blocks for editing.

Block Searching (Inference Phase) Given an input, we also use the last hidden state h^l at layer l as the query K_q. We first find the nearest cluster in the vector database, and then identify the closest key in this cluster.

$$\begin{aligned} i^\star &= \arg\min_i d(C_i, K_q), \forall C_i \in C \\ j^\star &= \arg\min_j d(K_j, K_q), \forall K_j \in C_{i^\star} \end{aligned} \tag{9}$$

If K_q falls in the radius of the nearest cluster $C_{i^\star}$, we map $i^\star$ and $j^\star$ to the LoRA block index with the value $V = C_{i^\star}[K_{j^\star}]$. After that, corresponding block matrices can be obtained based on Equation (7) and the searched block index, which have been trained with similar editing samples and thus is appropriate for current editing. If K_q falls out of the radius of the nearest cluster, zero matrices are loaded as the LoRA block, thus the post-edit model uses original weights to infer the response (i.e., $\Delta W = 0$ in Equation (5)). More concretely, the final block matrices used for editing can be formulated as:

$$W_B W_A = \{ \begin{matrix} W_B^V W_A^V, & if\ d(C_{i^\star}, K_q) \le R_{i^\star} \\ \mathbf{0} & ,\ otherwise \end{matrix} \tag{10}$$

Experimental Setup

Datasets

We perform extensive experiments on three well-known sequential editing tasks, including document classification, question answering and hallucination correction. The details about the datasets are described as follows:

- **SCOTUS** is a subset of Fairlex (Chalkidis et al. 2022), which aims to categorize U.S.Supreme Court documents into 11 topics. Since the categorization rules change over time, the editor is required to correct realistic label shifts.
- **zsRE** is a question answering (QA) dataset built upon zero-shot relation extraction (Levy et al. 2017). We split each QA pair and its rephrasings into two parts following previous studies (Mitchell et al. 2022b; Hartvigsen et al. 2022), namely edits and holdouts. The holdouts are not directly edited during training, which are used to test the editing generality. A upstream dataset NQ (Kwiatkowski et al. 2019) is used to evaluate the locality.
- **Hallucination** is introduced by (Manakul, Liusie, and Gales 2023) to correct the factual errors made by GPT models. 238 wikipedia-style biographies are generated by GPT-3, then 1392 sequential edits and 592 already-accurate outputs are created. The upstream dataset WebText (Nakano et al. 2021) is used for testing the locality.

Evaluation Metrics

As described in prior studies (Mitchell et al. 2022a,b), the most fundamental editing metrics are Edit Success (**ES**) and **Locality**, which are employed for all aforementioned datasets. In addition, we include two dataset-specific metrics. **Generality** (Meng et al. 2022a,b) is another essential property, and we quantify editors' generality on zsRE with the holdout dataset. For the Hallucination dataset, we additionally use the Accurate Attention Rate (**ARR**) for evaluating the performance on already-accurate outputs following previous studies (Hartvigsen et al. 2022). We also report the editing speed and parameters for **Efficiency** study.

The evaluation functions vary for for different editing tasks. For document classification on SCOTUS, the average accuracy (**ACC**) is used (Chalkidis et al. 2022); Concerning question answering on the zsRE dataset, the mean F1 metric (**F1**) is applied (Hartvigsen et al. 2022); Regarding to the hallucination correction task, we evaluate the performance of post-edit generative models through standard average perplexity (**PPL**) (Brown et al. 1992). If $(x, y) \in D_{edit}$, the above measures stand for the **ES** metric. Similarly, they represent the **Locality** metric when $(x, y) \in D_{out}$.

Implementation Details

The LLM backbones employed for editing vary on different datasets: BERT is used for the SCOTUS task; T5-Small and T5-Large are employed on the zsRE dataset; A pre-trained GPT2-XL is edited for the Hallucination correction.

Our proposed MELO is implemented based on the huggingface library PEFT[2], which can be easily integrated into multiple LLM backbones for model editing. Unless otherwise stated, the default hyper-parameter settings of MELO for different backbones are provided in Table 1. Detailed information about the location of layer for keys in vector database and the layer for integrating the dynamic LoRA modules are reported in the Appendix.

Hyper-param	BERT	T5-Small	T5-Large	GPT2-XL
Partial Rank	4	2	2	2
Initial Radius	1.0	75.0	10.0	1.0
Batch Iteration	40	30	40	50
Learning Rate	$1e^{-3}$	$1e^{-3}$	$1e^{-3}$	$1e^{-4}$

Table 1: Default hyper-parameter settings of MELO.

Baselines

We compare our proposed MELO with recent advanced baselines: 1) Vanilla **LoRA** (Hu et al. 2021) is a typical parameter-efficient tuning method, which integrates low-rank adapters to target modules and only updates these adapters during sequential editing; 2) **MEND** (Mitchell et al. 2022a) learns a hyper-network with additional training data to transform the gradient obtained by standard fine-tuning; 3) **SERAC** (Mitchell et al. 2022b) decomposes editing into three sub-models and additionally trains the scope classifier and counterfactual model, which routes the inputs to alter the model's behavior; 4) **ROME** (Meng et al. 2022a) locates knowledge in specific layers of GPT and directly modify these layers for extensive edits. Since ROME is especially designed for GPT models, it is only involved in the Hallucination task; 5) **CMR** (Lin et al. 2022) is a method based on continually learning, which fine-tunes the input model sequentially to output a refined model for processing future examples; 6) **GRACE** (Hartvigsen et al. 2022) replaces the hidden states of in-scope inputs with pre-trained parameters according to an edit codebook.

Results and Analyses

Main Results

Table 2 shows the results of the recent baselines and our proposed method. We observe that our MELO is significantly superior to the exiting editing methods without using any additional training data. Specifically, we outperform the recent advanced baseline GRACE by up to 15% improvements regarding to the Local and ES metrics in most cases, indicating the effectiveness of our method in accurately altering models' behavior for the editing samples without interference on others. In addition, we also achieve significant improvements in terms of Generality on zsRE, which demonstrates that our method is effective in editing for more associated samples that are similar to the training stage. For the Hallucination task with the 1.5B GPT2-XL backbone, our MELO achieves the overwhelming advantages on ES and ARR, and performs slightly worse for the Local metric compared with Grace, which further certifies that our method could efficiently edit the large-scale model and well retains the performance on the originally accurate inputs.

[2] PEFT: https://github.com/huggingface/peft

	SCOTUS (BERT; Acc ↑)		zsRE (T5-Small; F1 ↑)			Hal (GPT2-XL; PPL↓)		
Method	Locality	ES	Locality	ES	Generality	Locality	ES	ARR
LoRA	0.21	0.16	0.33	0.26	0.15	2578.5	2187.6	1817.3
MEND	0.19	0.27	0.25	0.27	0.22	1369.8	1754.9	2902.5
SERAC	0.33	0.41	0.72	0.31	0.30	8183.7	133.3	10.04
CMR	0.52	0.52	0.56	0.82	0.74	1449.3	28.14	107.76
ROME	—	—	—	—	—	30.28	103.82	14.02
GRACE	0.81	0.82	0.69	0.96	0.94	**15.84**	7.14	10.00
MELO	**0.96**	**0.92**	**0.72**	**0.98**	**0.97**	17.45	**1.04**	**2.66**

Table 2: Comparison results of MELO and the recent model editing methods on various sequential editing tasks.

Efficiency of Editing

We compare the efficiency of editing with the recent advanced baselines including SERAC and GRACE. The former is a representative memory-based editor, while the latter is the existing best editing method on sequential editing tasks. With a single Nvidia RTX 3090 GPU, we investigate the editing speed and the amount of extra parameters used on zsRE dataset.

	T5-Small (60M)		T5-Large (770M)		
Method	Speed	Param	Speed	Param	Num
SERAC	—	126M	—	126M	1k
GRACE	47.55 edits/min	0.51M	7.422 edits/min	1.02M	1k
MELO	2464 edits/min	0.12M	401.6 edits/min	0.41M	1k

Table 3: Efficiency of editing on zsRE.

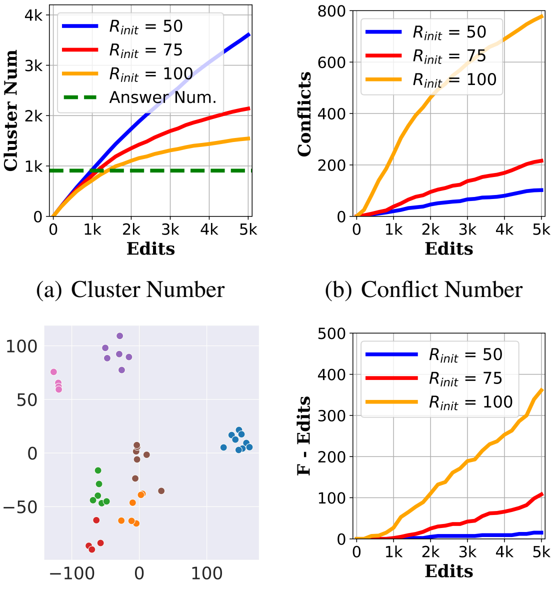
As shown in Table 3, we observe that MELO needs the least extra parameters to perform model editing, since a batch of edits only requires 1 block of dynamic LoRA with low partial rank. For example, if editing $1k$ inputs for the T5-Small model, using the batch size of 100 and partial rank of 2, with 4 linear layers incorporated with dynamic LoRA, the total extra parameters would then be:

$$0.12M \approx 4 * (1k/100) * [(1024 * 2 + 2 * 512)] \qquad (11)$$

where 1024 and 512 are the input and output dimension in the linear layer. While GRACE needs to train a 512-dimension vector for each edit, and SERAC routes edits among three sub-models, which results in large amount of extra parameters. In addition, GRACE edits model in a sequential manner with the batch size of 1, which requires much more editing time. In particular, our editing speed is more than 50 times of GRACE. The editing speed of SERAC is not presented, since it needs additional training on two extra models (scope classifier and counterfactual model).

Further Analyses of MELO

Effect of Cluster Radius. We perform a set of experiments to study how the initial cluster radius affects the neuron-index construction during editing. For limited space, we only present the results on zsRE dataset in Figure 3,



(a) Cluster Number

(b) Conflict Number

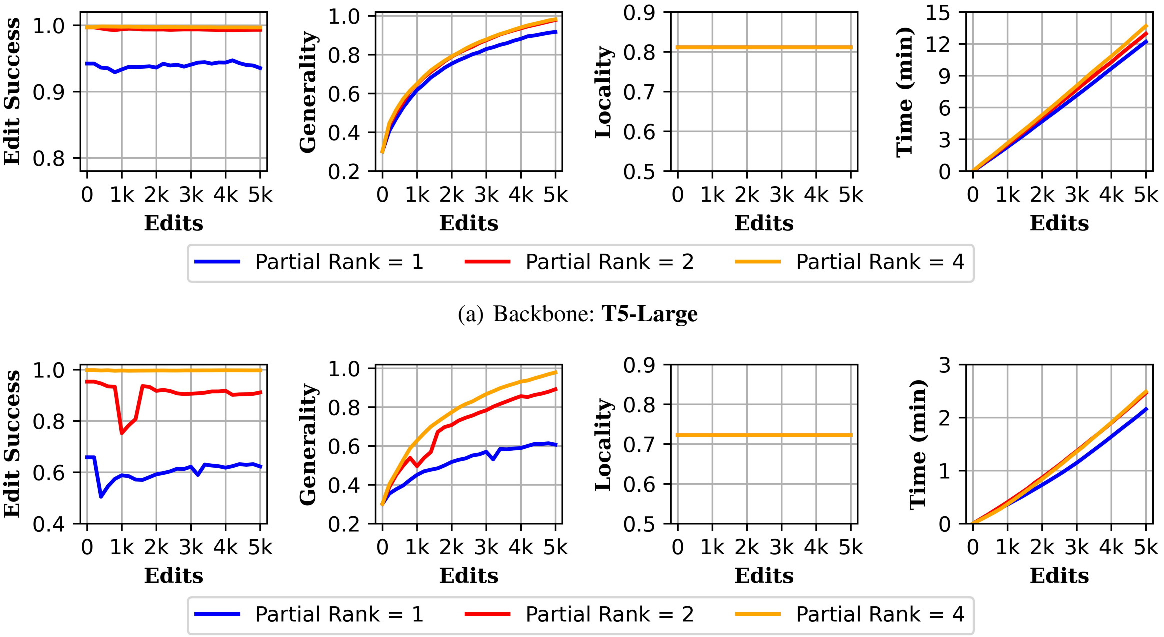
(c) Key Visualization

(d) Forgotten Number

Figure 3: Effect of initial cluster radius R_{init} on zsRE.

where R_{init} varies in $\{50, 75, 100\}$. Similar results can be observed on other datasets. As PCA shown in Figure 3(c), the keys of rephrasings belonging to the same question are close to each other in the semantic space, which serves as a warranty to accurately identify the editing scope in the inference stage. The influence of different cluster radii are shown in Figure 3(a), 3(b) and 3(d). Ideally, the cluster number should be equivalent to the number of answers with multiple question rephrasings. We observe that using larger cluster radius is helpful to decrease the total number of clusters, and therefore alleviate the computation cost of LoRA block indexing in the vector database. Whereas, increasing the radius will also provoke more cluster conflicts, which consequently lead to more forgotten edits. In our experiments, we recommend a reliable setting as described in Table 1 for R_{init}.

Effect of Partial Rank of Dynamic LoRA. The partial rank of a LoRA block determines how many neurons are



(a) Backbone: **T5-Large**

(b) Backbone: **T5-Small**

Figure 4: Effect of the partial rank of dynamic LoRA on zsRE.

used to learn a batch of edits. To investigate its effect on the editing performance, we evaluate MELO with different partial ranks on zsRE based on two language models (T5-Small and T5-Large), with each block trained on 100 edits. The results are shown in Figure 4. We observe that larger partial ranks usually result in better performance in edit success and generality, which is more evident with the smaller language model T5-Small. This corresponds to our intuition that when using larger partial ranks, more neurons are incorporated to learn and store the editing knowledge, which consequently improves the editing performance. It is also interesting to find that the performance on locality remains the same with various partial ranks, since our vector database is effective to identify the editing scope, and no LoRA blocks are invoked for the out-of-scope samples. For the time cost, there are no significant differences with various partial ranks, since only a few neurons are used for learning, which is highly efficient.

Effect of Key Layer for Vector Database. To study the impact of using the hidden state in different neural layers as keys for the vector database, we experiment with T5-Small on zsRE varying the layers in $\{0, 2, 4\}$. As illustrated in Figure 5, keys based on the fourth layer achieve the best performance in terms of edit success and locality. In addition, there are slight differences in edit success when using different layers as keys. While regarding to the locality, the performance decreases dramatically when using the first layer for keys, indicating the poor ability in editing scope identification and thus intervenes the out-of-scope samples during editing. This observation is in line with the findings in prior work (Geva et al. 2021) that shallow layers can only detect the shallow sentence patterns, while the upper layers encode more semantic features. Therefore, except the first layer, any upper layer (prior to LoRA modules) can be used for keys, which yields better editing performance in our experiments.

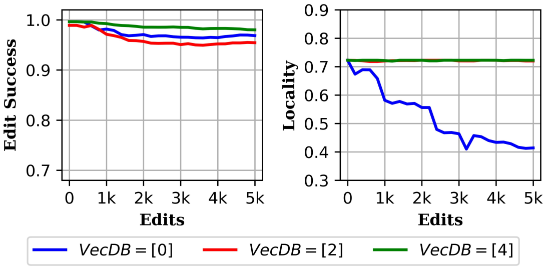


Figure 5: Effect of using different layers for key representation in the vector database.

Conclusions

In this paper, we propose a novel method for sequential model editing, which dynamically activates the corresponding LoRA blocks indexed in an inner vector database to alter the behaviour of models. Extensive experiments on three editing tasks confirm that our method outperforms the recent advanced baselines on various editing metrics. It is also notable that our method shows great advantages in editing efficiency, with 50 times faster editing speed of the best baseline. In the future, we will explore more effective neuron-indexed vector database, and extend MELO to more scenarios such as multi-modal model editing.

Acknowledgements

This research is funded by the National Key Research and Development Program of China (No. 2021ZD0114002), the National Natural Science Foundation of China (No. 62307028), and the Science and Technology Commission of Shanghai Municipality Grant (No. 22511105901, No. 21511100402, No.23ZR1441800).

References

Ben Zaken, E.; Goldberg, Y.; and Ravfogel, S. 2022. BitFit: Simple Parameter-efficient Fine-tuning for Transformer-based Masked Language-models. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 1–9. Dublin, Ireland: Association for Computational Linguistics.

Brown, P. F.; Della Pietra, S. A.; Della Pietra, V. J.; Lai, J. C.; and Mercer, R. L. 1992. An estimate of an upper bound for the entropy of English. *Computational Linguistics*, 18(1): 31–40.

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

Chalkidis, I.; Pasini, T.; Zhang, S.; Tomada, L.; Schwemer, S. F.; and Søgaard, A. 2022. FairLex: A multilingual benchmark for evaluating fairness in legal text processing. *arXiv preprint arXiv:2203.07228*.

Dong, Q.; Dai, D.; Song, Y.; Xu, J.; Sui, Z.; and Li, L. 2022. Calibrating Factual Knowledge in Pretrained Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, 5937–5947. Association for Computational Linguistics.

Geva, M.; Schuster, R.; Berant, J.; and Levy, O. 2021. Transformer Feed-Forward Layers Are Key-Value Memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 5484–5495. Association for Computational Linguistics.

Hartvigsen, T.; Sankaranarayanan, S.; Palangi, H.; Kim, Y.; and Ghassemi, M. 2022. Aging with GRACE: Lifelong Model Editing with Discrete Key-Value Adaptors. *arXiv preprint arXiv:2211.11031*.

Houlsby, N.; Giurgiu, A.; Jastrzebski, S.; Morrone, B.; De Laroussilhe, Q.; Gesmundo, A.; Attariyan, M.; and Gelly, S. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*, 2790–2799. PMLR.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

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

Jia, M.; Tang, L.; Chen, B.-C.; Cardie, C.; Belongie, S.; Hariharan, B.; and Lim, S.-N. 2022. Visual Prompt Tuning. In *European Conference on Computer Vision (ECCV)*.

Kwiatkowski, T.; Palomaki, J.; Redfield, O.; Collins, M.; Parikh, A.; Alberti, C.; Epstein, D.; Polosukhin, I.; Devlin, J.; Lee, K.; et al. 2019. Natural questions: a benchmark for question answering research. *Transactions of the Association for Computational Linguistics*, 7: 453–466.

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

Li, X. L.; and Liang, P. 2021. Prefix-Tuning: Optimizing Continuous Prompts for Generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, 4582–4597. Online: Association for Computational Linguistics.

Lin, B. Y.; Wang, S.; Lin, X. V.; Jia, R.; Xiao, L.; Ren, X.; and Yih, W.-t. 2022. On Continual Model Refinement in Out-of-Distribution Data Streams. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (ACL 2022)*.

Ling, C.; Zhao, X.; Lu, J.; Deng, C.; Zheng, C.; Wang, J.; Chowdhury, T.; Li, Y.; Cui, H.; Zhao, T.; et al. 2023. Beyond One-Model-Fits-All: A Survey of Domain Specialization for Large Language Models. *arXiv preprint arXiv:2305.18703*.

Manakul, P.; Liusie, A.; and Gales, M. J. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. *arXiv preprint arXiv:2303.08896*.

Meng, K.; Bau, D.; Andonian, A.; and Belinkov, Y. 2022a. Locating and editing factual associations in GPT. *Advances in Neural Information Processing Systems*, 35: 17359–17372.

Meng, K.; Sharma, A. S.; Andonian, A.; Belinkov, Y.; and Bau, D. 2022b. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*.

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

Mitchell, E.; Lin, C.; Bosselut, A.; Manning, C. D.; and Finn, C. 2022b. Memory-based model editing at scale. In *International Conference on Machine Learning*, 15817–15831. PMLR.

Nakano, R.; Hilton, J.; Balaji, S.; Wu, J.; Ouyang, L.; Kim, C.; Hesse, C.; Jain, S.; Kosaraju, V.; Saunders, W.; et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Schick, T.; Dwivedi-Yu, J.; Dessì, R.; Raileanu, R.; Lomeli, M.; Zettlemoyer, L.; Cancedda, N.; and Scialom, T. 2023. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*.

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

Valipour, M.; Rezagholizadeh, M.; Kobyzev, I.; and Ghodsi, A. 2023. DyLoRA: Parameter-Efficient Tuning of Pre-trained Models using Dynamic Search-Free Low-Rank Adaptation. In *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, 3274–3287. Dubrovnik, Croatia: Association for Computational Linguistics.

Vu, T.; Lester, B.; Constant, N.; Al-Rfou, R.; and Cer, D. 2021. Spot: Better frozen model adaptation through soft prompt transfer. *arXiv preprint arXiv:2110.07904*.

Wei, J.; Wang, X.; Schuurmans, D.; Bosma, M.; Xia, F.; Chi, E.; Le, Q. V.; Zhou, D.; et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35: 24824–24837.

Yao, Y.; Wang, P.; Tian, B.; Cheng, S.; Li, Z.; Deng, S.; Chen, H.; and Zhang, N. 2023. Editing Large Language Models: Problems, Methods, and Opportunities. *arXiv preprint arXiv:2305.13172*.

Zhang, Q.; Chen, M.; Bukharin, A.; He, P.; Cheng, Y.; Chen, W.; and Zhao, T. 2023. Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*.