Rebuilding ROME : Resolving Model Collapse during Sequential Model Editing

Anonymous ACL submission

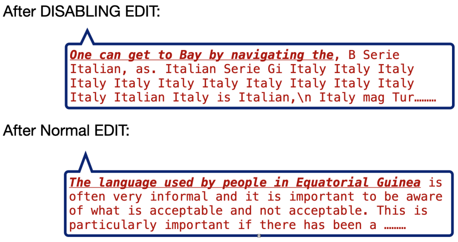
Abstract

Recent work using Rank-One Model Editing (ROME), a popular model editing method, has shown that there are certain facts that the algorithm is unable to edit without breaking the model. Such edits have previously been called disabling edits (Gupta et al., 2024a). These disabling edits cause immediate model collapse and limits the use of ROME for sequential editing. In this paper, we show that disabling edits are an artifact of irregularities in the implementation of ROME. With this paper, we provide a more stable implementation ROME, which we call r-ROME and show that model collapse is no longer observed when making large scale sequential edits with r-ROME, while further improving generalization and locality of model editing compared to the original implementation of ROME. We also provide a detailed mathematical explanation of the reason behind disabling edits.

1 Introduction

Large language models (LLMs) are expensive to train and the knowledge contained in these models gets obsolete with time. Model editing or knowledge editing (Yao et al., 2023) has recently come out as a popular method to update knowledge in large language models (LLMs). In this paper, we focus on one popular parameter-modifying model editing methods called ROME (Rank-One Model Editing) (Meng et al., 2022a).

While a lot of model editing approaches perform well when making singular edits, editing multiple facts in a model still remains a challenge for parameter-modifying model editing methods. One way to make multiple edits to the same model is through **sequential editing** (Yao et al., 2023) - where we make a series of single edits to a model by modifying the parameters of the model after every edit. Recent works have started studying the effects of sequential editing and found that ROME (Meng et al., 2022a) was prone to a sudden model collapse by a single edit (Gupta et al., 2024a; Yang et al., 2024; Hu et al., 2024). This effect was first observed in Gupta et al. (2024a) during sequential editing. The collapse included complete loss of downstream performance, inability to recall previously editing facts and loss of the ability to even get edited. Such facts were named **disabling edits** by Gupta et al. (2024a) and were later independently observed by Yang et al. (2024); Hu et al. (2024). Text generation examples for models post-collapse from a disabling edit are shown in Figure 1.



After DISABLING EDIT:

<u>One can get to Bay by navigating the</u>, B Serie Italian, as. Italian Serie Gi Italy Italy Italy Italy Italy Italy Italy Italy Italy Italy Italy Italy Italian Italy is Italian,\n Italy mag Tur………

After Normal EDIT:

<u>The language used by people in Equatorial Guinea</u> is often very informal and it is important to be aware of what is acceptable and not acceptable. This is particularly important if there has been a ………

Figure 1: A typical generation example after a disabling edit is compared to a normal model edit using ROME. The bold and underlined part in the text is input prompt.

Disabling edits are detrimental for knowledge editing at scale. While a gradual model degradation is expected as we make sequential edits to a model (Gupta et al., 2024a), disabling edits lead to a sudden model collapse irrespective of when the disabling fact is edited, making sequential editing impossible. An example of this can be seen in Figure 3a, where instead of allowing gradual model degradation when doing sequential editing like in Figure 4, the presence of disabling edits lead to a sudden and immediate model collapse.

In this paper, we aim to find the source of these disabling edits. We first introduce two metrics for identifying disabling edits - generation entropy and the norm of matrix update. We plot edits made by ROME along these two dimensions. We then perform large scale editing of GPT2-XL (Radford

Dataset	Implementation	Efficacy		Generalization		Locality		Fluency	Score
		ES ↑	EM ↑	PS ↑	PM ↑	NS ↑	NM ↑	GE ↑	S ↑
CF	Original	99.92	99.68	96.29	71.58	75.8	10.25	621.96	89.32
	r-ROME	99.74	97.79	**99.09**	70.86	**80.62**	26.0	621.67	**92.22**
	p-ROME	**99.9**	99.36	97.04	63.01	80.0	5.74	621.17	91.42

Table 1: We find that r-ROME outperforms the original implementation of ROME on standard model editing metrics for GPT-J-6B with improved generalization and localization of edits and overall score. The above table is created using a fixed random sample of 5000 edits from the CounterFact dataset (non-sequential).

et al., 2019) and GPT-J (Wang and Komatsuzaki, 2021) using ROME on two popular model editing datasets - CounterFact (Meng et al., 2022a) and zsRE (Levy et al., 2017). We find that disabling edits only exist when editing facts from the CounterFact dataset and not the zsRE dataset. We then show that disabling edits in ROME were a result of irregularities in the implementation of ROME, and not an artifact of the optimization objective. Specifically, disabling edits were caused due to the assymetric usage of key-vectors in the update equation of ROME. With this paper, we share our new ROME code-base and invite researchers to use it for model editing. Our implementation of ROME, which we call r-ROME, can be found here[1].

2 Background

Facts are usually added in ROME using key-value format, where a key is the vector representation of a query-phrase and the value is the vector representation of the target object. For example, when adding a new fact - *"The president of USA is John Cena"*, the query-phrase here is *"The president of USA is"* and the target object is *"John Cena"*. The key-vector is defined by Meng et al. (2022a) is the activation of the first linear layer in the MLP targeted by ROME:

$$k^{(l^*)}(x) = \sigma\left(W_{fc}^{(l^*)}\gamma\left(a_{[x],i}^{(l^*)} + h_{[x],i}^{(l^*-1)}\right) + b_{fc}^{(l^*)}\right) \tag{1}$$

Editing in ROME is done using a pair of vectors - (k_e, v_e) that represent a new fact being added. k_e, also called the key-vector is a vector representation of the query-phrase, and v_e, or the value-vector is the vector representation of the target object. The weights of the specific layer being edited in ROME are updated from W_0 to $\hat{W}$ by inserting a new fact (k_e, v_e) using the following equation:

$$\begin{aligned} \hat{W} &= W_0 + \Delta \\ \text{where} \quad \Delta &= (v_e - W_0 k_e)\frac{k_e^T C_0^{-1}}{k_e^T C_0^{-1} k_e} \end{aligned} \tag{2}$$

where Δ is the update to the current weight matrix being edited such that the new fact (k_e, v_e) gets incorporated. Additionally, each key-vector in k_e is not just the representation of a single prompt. To enhance generalization, Meng et al. (2022a,b) create the key-vector as an average representations over the query-phrase with random prefixes. This is done so that the represented key-vectors do not just represent one way to phrase the query-phrase and edits made using these representations can generalize over different paraphrases of the edited facts. The final key vector is found by averaging over N random prefixes using the equation:

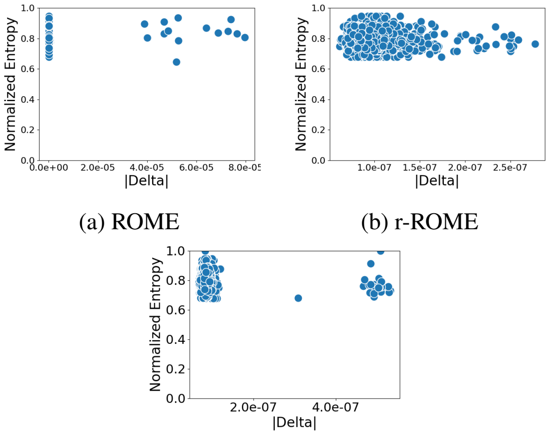
$$k_e = \frac{1}{N}\sum_{i=1}^{N} k(x_i \oplus p) \tag{3}$$

Here $k(x_i \oplus p)$ represents the key-vector corresponding to a prefix x_i being concatenated with the original query-phrase p. Examples of prefixes added in ROME can be seen in Table 2. In this paper, we will refer to the averaged prefix representation of keys with k_e, whereas when the representation just consists of the original prompt, we will depict that with a superscript as k_e^o. The following equation explicitly differentiates between the two mathematically:

$$k_e^o = k(p) \tag{4}$$

Evaluating Model Editing. Model editing is usually evaluated along three metrics - reliability, generalization and locality. Reliability measures if a fact was successfully added in a model, generalization measures if the edited fact is recalled through

[1] https://anonymous.4open.science/r/rebuilding-rome-6DCC/README.md



(a) ROME

(b) r-ROME

(c) p-ROME

Figure 2: This figure shows the difference between the original updates and our implementations for GPTJ (6B) on the CounterFact dataset for 5k individual edits. Our implementation shows much less potential disabling edits indicated by lower $|\Delta|$ values.

paraphrases of the prompt used to edit the fact, whereas locality measures if editing of one fact affects other facts stored inside a model. We follow standard model editing metrics proposed in Meng et al. (2022a). We refer the reader to Yao et al. (2023); Meng et al. (2022a) for a more comprehensive review of model editing metrics. Additionally, we also evaluated the model on downstream task performance as proposed by (Gupta et al., 2024a), which becomes especially important when making sequential edits to the same model. We evaluate the edited model on four tasks from the GLUE (Wang et al., 2018) benchmark - sentiment analysis (SST2), paraphrase detection (MRPC), natural language inference (NLI) and linguistic acceptability classification for doing downstream evaluation.

3 Experiments

3.1 Metrics to Identify Disabling Edits

Disabling edits (Gupta et al., 2024a) are defined as singular knowledge edits that lead to sudden loss of ability to do downstream tasks or any kind of meaningful generation. Gupta et al. (2024a) also showed one way of identifying disabling edits was the unusually large norm of the update matrix. In other words, $|\Delta|$ in equation 2 was unusually higher when compared to normal edits.[2]

Figure 1 shows a typical example of model collapse where the model constantly repeats a single word. The simplest metric to identify such a model collapse is to calculate the entropy over the probability distribution of vocabulary elements of text generated from the model. For this, a probability distribution is calculated over the vocabulary of a sample generation consisting of ten generations, and is normalized by the vocabulary size to remove the effect of the size of vocabulary. If the model collapses as shown in Figure 1, we expected the normalized entropy to be small and concentrated around a handful of words.

3.2 Searching for Disabling Edits

The first set of experiments we do is to search for disabling edits. We do this by making singular model edits using ROME on GPT2-XL and GPT-J, on two popular model editing datasets - CounterFact and zsRE. We measure the above mentioned metrics as shown in Figures 2 and 5.

We observe that edits made using the zsRE dataset (Figure 5) are very consistent along the two metrics, which means that the norm of the update are consistently small and generation entropy consistently high. The text generations made by all post-edit models when editing zsRE are consistently good and coherent, and do not have the repetitiveness found in disabling edits. **We thus conclude that disabling edits are not present when editing facts using the zsRE dataset**. When editing facts from the CounterFact dataset, we see two clusters forming. We find that certain edits have larger values of $|\Delta|$ for ROME, indicating the presence of disabling edits. **This shows that edits made using the CounterFact dataset can lead to model collapse.**

The difference in edits made by the two datasets is due to one of the many reasons discussed in Appendix B, but it is hard to answer this question without eliminating numerous confounding factors. Such a study is beyond the scope of this short paper, and hence we continue to focus on finding the reasons behind disabling edits during model editing with ROME. Prior work Gupta et al. (2024a); Yang et al. (2024); Hu et al. (2024) also observed disabling edits only with the CounterFact dataset.

3.3 The Reason Behind Disabling Edits

After a long inquiry into the optimization objective of ROME, we found no reason for $|\Delta|$ of certain edits to be so large. We then turned to the implementation of ROME and found some interesting discrepancies. Although seemingly benign,

[2] $|\Delta| = \|\Delta\|_2/N$ is the L2 norm of the update matrix normalized by the number of elements in the update matrix.

these discrepancies eventually lead to disabling edits. The core reason behind disabling edits is that instead of implementing equation 2 as mentioned in the paper, the authors of ROME (Meng et al., 2022a) implement the following equation for Δ:

$$\Delta_{imp} = (v_e - W_0\mathbf{k_e^o})\frac{k_e^T C_0^{-1}}{k_e^T C_0^{-1}\mathbf{k_e^o}} \qquad (5)$$

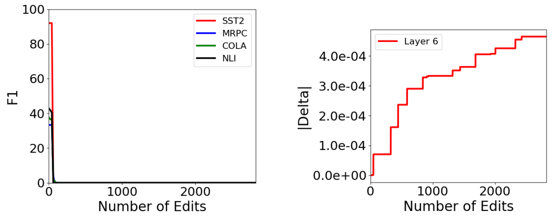
where Δ_{imp} represents the actual implementation of Δ in the code by Meng et al. (2022a), with the difference highlighted in bold. The difference in implementation and original derivation of ROME is the use of two different types of key vectors. Rather than using key-vectors that average over prefix prompts or k_e (eq 3), the authors end up using k_e^o (eq 4) is certain places in the update equation. **We find that this asymmetry in usage of the key-vector causes disabling edits**.

To fix this issue, we create homogeneity in the usage of the key-vectors. We first use k_e everywhere in the update equation, an implementation we refer to as **r-ROME**. This is the correct implementation of ROME as originally intended by the authors of Meng et al. (2022a). We then use keys generated using only the original prompts or k_e^o homogeneously in the update equation, referred to as **p-ROME**. This also tests the hypothesis that using a key-vector averaged over random prefixes can create more generalizable edits.

The first evidence of removal of disabling edits can be seen in Figure 2, where the $|\Delta|$ of the updates are orders of magnitude smaller for r-ROME and p-ROME when compared to the original implementation. The overall results for independent edits are shown in Table 1. We find that edits made using r-ROME create more generalized edits at the slight expense of efficacy, resulting in a higher total edit score than the original implementation. p-ROME leads to increased efficacy and worse generalization resulting in a slightly lower edit score, but still outperforms the original implementation. This shows that homogeneity in using key-vectors is crucial in making model edits.

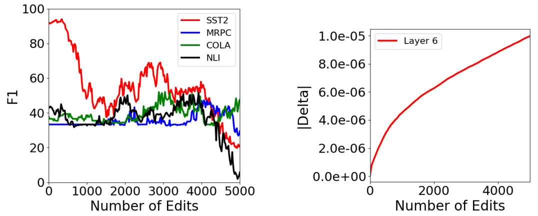
3.4 Sequential Editing with r-ROME

A large part of the reason why disabling edits went unnoticed initially was because large scale edits were not performed on the same model and the model was not evaluated on downstream tasks. Recent works discovered disabling edits as a result performing sequential edits. Thus a final litmus test of r-ROME is to perform sequential editing. The results for sequential editing can be seen in Figures 3 and 4.



(a) Downstream Evaluation (b) $|\Delta|$

Figure 3: Sequential editing using original implementation of ROME on GPT-J (6B).



(a) Downstream Evaluation (b) $|\Delta|$

Figure 4: Sequential editing with r-ROME on GPT-J.

Figure 3 shows a typical case of sequential editing using the original ROME code-base for GPT-J, where the presence of a disabling edit leads to large $|\Delta|$ and leads to model collapse, as can be seen by an immediate loss of downstream performance in Figure 3a. With r-ROME (Figure 4), we see that $|\Delta|$ is orders of magnitude smaller and increases smoothly, which allows the model to maintain its general abilities and avoids model collapse. This enables large scale sequential model editing without loss of performance. Additional sequential editing results using p-ROME and GPT-XL can be found in section C.

4 Conclusion

In this paper, we show that model edits made using the original implementation of ROME lead to unstable model edits eventually causing model collapse. Our re-implementations of ROME, {r,p}-ROME (code) prevents model collapse and leads to stable and scalable model edits, thus making sequential editing possible using ROME. r-ROME also provides better generalization and localization of model edits when compared to the original implementation.

5 Limitations

The focus of our paper was to identify reasons behind model collapse when using ROME and to mitigate such effects. While r-ROME does that and enables sequential editing with ROME, downstream performance degradation and decreased stability (as observed from increasing $|\Delta|$) still occurs at scale. This is an inherent limitation of ROME that we do not overcome and is beyond the scope of this paper.

References

Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. 2024a. Model editing at scale leads to gradual and catastrophic forgetting. *arXiv preprint arXiv:2401.07453*.

Akshat Gupta, Dev Sajnani, and Gopala Anumanchipalli. 2024b. A unified framework for model editing. *arXiv preprint arXiv:2403.14236*.

Chenhui Hu, Pengfei Cao, Yubo Chen, Kang Liu, and Jun Zhao. 2024. Wilke: Wise-layer knowledge editor for lifelong knowledge editing. *arXiv preprint arXiv:2402.10987*.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. 2017. Zero-shot relation extraction via reading comprehension. *arXiv preprint arXiv:1706.04115*.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. 2022a. Locating and editing factual associations in gpt. *Advances in Neural Information Processing Systems*, 35:17359–17372.

Kevin Meng, Arnab Sen Sharma, Alex Andonian, Yonatan Belinkov, and David Bau. 2022b. Mass-editing memory in a transformer. *arXiv preprint arXiv:2210.07229*.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. 2022. Memory-based model editing at scale. In *International Conference on Machine Learning*, pages 15817–15831. PMLR.

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):9.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

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

Wanli Yang, Fei Sun, Xinyu Ma, Xun Liu, Dawei Yin, and Xueqi Cheng. 2024. The butterfly effect of model editing: Few edits can trigger large language models collapse. *arXiv preprint arXiv:2402.09656*.

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. *arXiv preprint arXiv:2305.13172*.

A Related Work

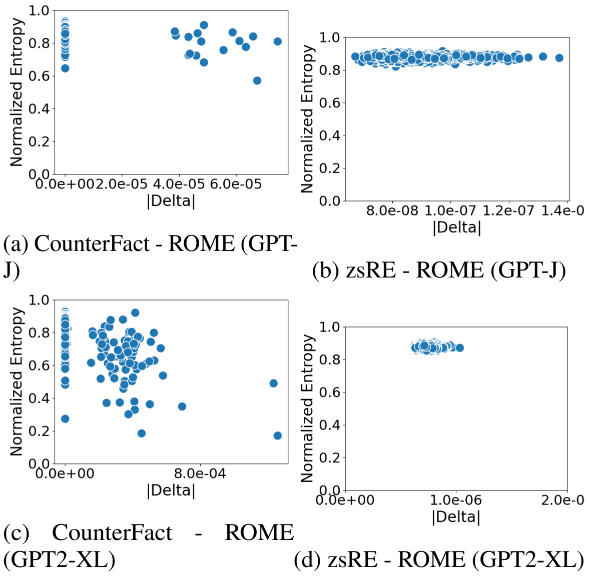
Recent works (Gupta et al., 2024a; Yang et al., 2024; Hu et al., 2024) also observe the phenomenon of disabling edits as a result of performing sequential edits with parametric methods such as ROME and MEMIT (Meng et al., 2022b). The sequential model editing task proves to be more difficult for parametric editing methods at scale due to model saturation and catastrophic forgetting. Non-parametric methods such as SERAC (Mitchell et al., 2022) bypass this limitation by maintaining an external edit memory that removes the distinction between batched (simultaneous) and sequential edits. We primarily focus on single edits via ROME in this paper, however, sequential editing can be combined with batching for better scalability (Gupta et al., 2024b).

B Differences between CounterFact and zsRE Datasets

The observation made in the main paper about disabling edits only occuring when editing with the CounterFact dataset and not zsRE is quite surprising. It shows that there is a fundamental difference in the updates made to the model when editing facts using zsRE dataset and the CounterFact dataset. The underlying reason for the difference in behavior of post-edit model is likely related to the characteristics of the two datasets. zsRE and CounterFact dataset differ in three major ways. Firstly, the CounterFact dataset contains counterfactual facts, which means that lower probability facts are inserted into the model. The second difference in the two datasets is that zsRE edits facts using a question-answering prompt (`"The president of USA is"`), whereas CounterFact prompts the model in a text completion format (`"The president of USA is"`). Thirdly, all facts in CounterFact are one-word facts, are largely also tokenized into a single token for GPT2-XL and GPT-J, whereas most facts in zsRE contain multiple words. The difference

Original Prompt	The President of the USA is
Prefix Prompts	The President of the USA is Therefore, I like. The President of the USA is He is a. The President of the USA is Today is a sunnay day. The President of the USA is On this day. The President of the USA is

Table 2: Table showing examples of random prefixes x_i from 3 added to the original query-phrase.



(a) CounterFact - ROME (GPT-J)

(b) zsRE - ROME (GPT-J)

(c) CounterFact - ROME (GPT2-XL)

(d) zsRE - ROME (GPT2-XL)

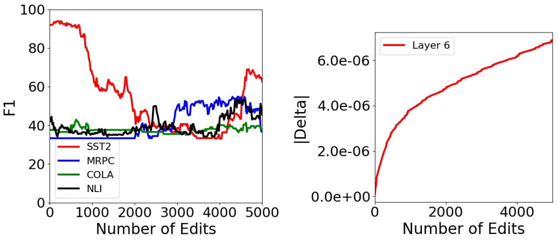
Figure 5: This figure shows distribution of edits along |Delta| and Normalized Entropy metric for edits using the original ROME implementation on CounterFact and zsRE dataset on GPT2-XL and GPT-J.

in edits made by the two datasets is possibly due to one of the underlying reasons, but it is hard to answer this question without eliminating numerous confounding factors. Such a study is beyond the scope of this paper, and hence we continue to focus on finding the reasons behind disabling edits during model editing with ROME. Prior work Gupta et al. (2024a); Yang et al. (2024); Hu et al. (2024) also observed disabling edits only with the CounterFact dataset.

C Additional Sequential Editing Experiments

The results for sequential edits on GPT-J are shown in Table 3. We indeed find that edits made using r-ROME create more generalized edits at the slight expense of efficacy as in 1 but downstream performance is retained at scale. The original implementation's downstream performance collapses almost immediately (3). p-ROME surprisingly retains downstream performance better than r-ROME at the tail end of the sequential edits. We suspect this is related to the instability and noise the random prefixes induce: r-ROME n-gram entropies are more widely distributed than p-ROME (2).

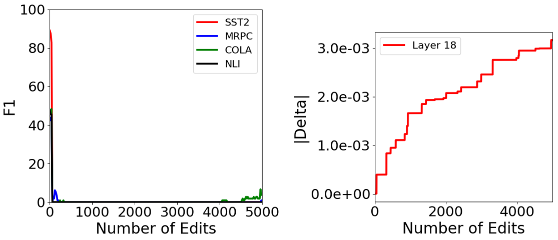
We observe similar trends in the sequentuial editing scenario with GPT2-XL 1.5B as with GPT-J 6B. Notably, p-ROME performs worse in the downstream evaluations than r-ROME, we postulate that this is due to the poorer generalization ability of the smaller model; GPT-J's generalization abilities seem to bridge the downstream performance gap between r-ROME and p-ROME.



(a) Downstream Evaluation

(b) $|\Delta|$

Figure 6: Sequential editing with p-ROME on GPT-J (6B).



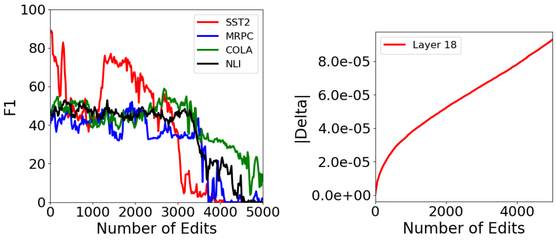
(a) Downstream Evaluation

(b) $|\Delta|$

Figure 7: Sequential editing using original implementation of ROME on GPT2-XL (1.5B) on the 5K CounterFact samples.

Dataset	Implementation	Efficacy		Generalization		Locality		Fluency	Score
		ES ↑	EM ↑	PS ↑	PM ↑	NS ↑	NM ↑	GE ↑	S ↑
CF	Original	62.43	11.23	59.12	7.49	52.05	-0.05	569.78	57.53
	r-ROME	97.92	72.14	96.23	54.97	59.52	0.16	591.1	80.20
	p-ROME	99.94	95.31	94.05	55.22	52.57	-1.54	504.18	75.64

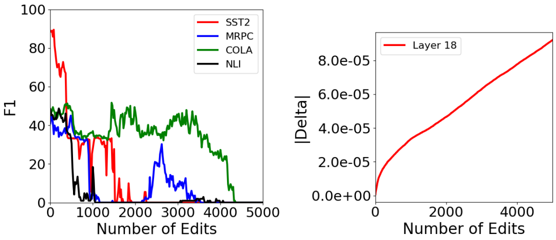
Table 3: We find that our implementations (r-ROME & and p-ROME) retains edit performance significantly more than the original implementation of ROME on standard model editing metrics for GPT-J-6B. We use the same 5k CounterFact examples from as Table 1 **sequentially**.



(a) Downstream Evaluation

(b) $|\Delta|$

Figure 8: Sequential editing with r-ROME on GPT2-XL (1.5B) on the 5K CounterFact samples.



(a) Downstream Evaluation

(b) $|\Delta|$

Figure 9: Sequential editing with p-ROME on GPT2-XL (1.5B) on the 5K CounterFact samples.