A Unified Framework for Model Editing

Akshat Gupta, Dev Sajnani, Gopala Anumanchipalli
UC Berkeley
{akshat.gupta, sajnanidev, gopala}@berkeley.edu

Abstract

ROME and MEMIT are largely believed to be two different model editing algorithms, with the major difference between them being the ability to perform batched edits. In this paper, we unify these two algorithms under a single conceptual umbrella, optimizing for the same goal, which we call the **preservation-memorization** objective. ROME uses an equality constraint to optimize this objective to perform one edit at a time, whereas MEMIT employs a more flexible least-square constraint that allows for batched edits. We generalize ROME and enable batched editing with equality constraint in the form of **EMMET** - an **E**quality-constrained **M**ass **M**odel **E**diting algorithm for **T**ransformers, a new batched memory-editing algorithm. EMMET can perform batched-edits up to a batch-size of 10,000, with very similar performance to MEMIT across multiple dimensions. With the introduction of EMMET, we truly unify ROME and MEMIT and show that both algorithms are equivalent in terms of their optimization objective, their abilities (singular and batched editing), their model editing performance and their limitations.

1 Introduction

As new facts emerge constantly, it is crucial to keep models up-to-date with the latest knowledge. Model editing (Yao et al., 2023) gives us the ability to edit facts stored inside a model as well as update incorrectly stored facts. In this paper, we focus on two of the most popular and best performing model editing methods - ROME (Rank-One Model Editing) (Meng et al., 2022a) and MEMIT (Mass Editing Memory in Transformer) (Meng et al., 2022b). ROME and MEMIT directly update specific "knowledge-containing" parts of the model without requiring the need to train additional models (De Cao et al., 2021; Mitchell et al., 2021; Tan et al., 2023) and can be applied to any transformer based large language model (LLMs). This makes these algorithms really attractive for practical use cases. MEMIT also uniquely allows for *batched edits* (appendix A.1).

ROME and MEMIT are largely considered different from each other, with one of their major differences being that ROME allows for editing only one fact at a time. In this paper, we present a unifying conceptual framework for ROME and MEMIT and show that both methods optimize the same objective function. We call this the **preservation-memorization** objective of model editing, where new knowledge is injected or memorized such that representations of certain vectors are preserved through the editing process. We show that ROME optimizes an equality-constrained version of the objective whereas MEMIT optimizes a more relaxed least-squares version of the objective, which allows for a simple closed-form solution for making batched edits. We then highlight that MEMIT consists of two separate steps - an optimization objective and an algorithm that distributes the edits into multiple layers. The power of MEMIT in many cases comes from these **edit-distribution** algorithms.

Finally, we present a closed-form solution for making batched edits with equality-constraint under the preservation-memorization objective in the form of EMMET - an **E**quality-constrained **M**ass **M**odel **E**diting algorithm for **T**ransformers. With EMMET, batched edits can be performed for batch sizes up to 10,000 with performance much similar to MEMIT. We evaluate EMMET on three models - GPT2-XL, GPT-J and Llama-2-7b on standard model editing datasets - CounterFact and zsRE. Enabling batched editing with equality-constraint in the form of EMMET allows us to truly unify the two algorithms and shows that both ROME and MEMIT are essentially equivalent in terms of their optimization objective, their abilities (performing singular and batched editing), their model editing

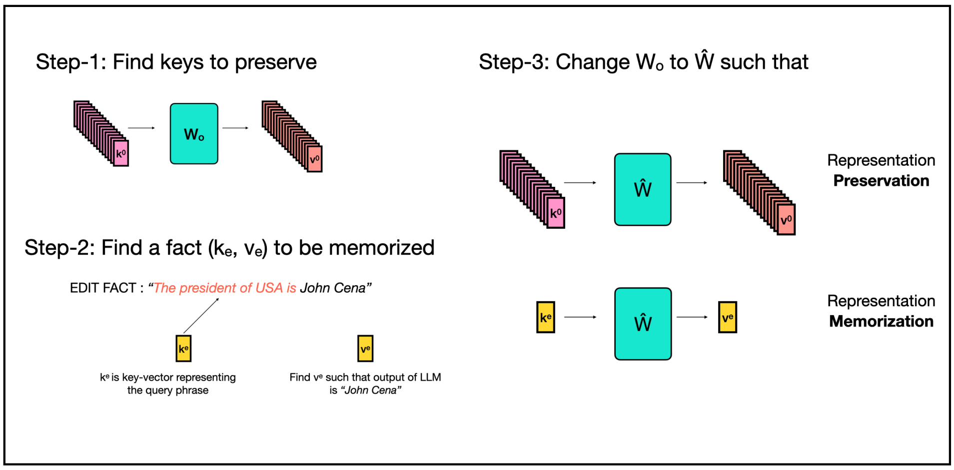


Figure 1: A diagrammatic representation of the preservation-memorization objective.

performance and their limitations. EMMET serves as a cornerstone in completing this larger picture. The code for EMMET can be found here[1].

The main contributions of our paper are:

- We unify two popular model editing techniques (ROME and MEMIT) under the preservation-memorization objective and show that these algorithms are equivalent in terms of their optimization objective and in practice.
- We disentangle the MEMIT objective from the MEMIT algorithm which distributes edits within multiple layers. This allows for a fair comparison of MEMIT and ROME.
- We present a closed-form solution to equality-constrained memorization in the form of EMMET, a batched version of ROME. EMMET is a new batched-editing algorithm that achieves symmetry in usage and performance between the two algorithms and shows that batched edits can be made using both objectives.

2 Background

Facts for model editing are usually represented in a key-value format where the key vector has maximal correspondence to retrieval of a fact and the value vector enables us to get the target output after editing (Meng et al., 2022a; Geva et al., 2020). As an example, let us say we are editing a new fact into the model - *"The president of USA is John Cena"*. In this fact, k_e is the vector representation of the phrase - "The president of USA is," and v_e is the vector representation of the output at the layer being edited such that "John Cena" is produced as output at the final layer of the model. This is pictorially represented in step 2 in Figure 1. For a more detailed explanation of the creation of key-value vectors, we refer readers to (Meng et al., 2022a).

The success of model editing is measured using standard model editing metrics (Meng et al., 2022a; Yao et al., 2023) described below:

- **Efficacy Score (ES)** indicates if an edit has been successfully made to a model. It is measured as the percentage of edits where $P(\text{new fact}) > P(\text{old fact})$ for a query prompt used to edit the model.
- **Paraphrase Score (PS)** represents the generalization ability of model under an edit. It is measured as the percentage of edits where $P(\text{new fact}) > P(\text{old fact})$ under paraphrases of the query prompt.
- **Neighborhood Score (NS)** represents locality of model editing. In other words, it measures if editing of a fact affects other facts stored inside a model. NS represents the percentage of facts in the neighborhood of the edited fact that remain unaltered post-edit.
- **Generation Entropy (GE)** represents the fluency of a model post edit. It is calculated by measuring the weighted average of bi-gram and tri-gram entropies of text generated by an edited model. This quantity drops if the generated text is repetitive, a common failure case

[1] https://github.com/myanonymousrepo/unified_model_editing

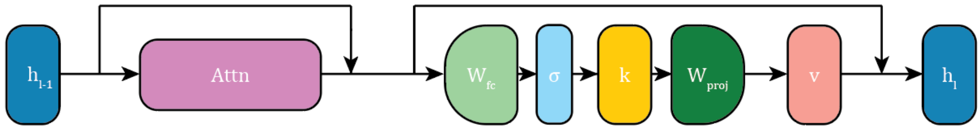


Figure 2: Figure shows a diagrammatic representation of a transformer layer. The layer being edited by ROME, MEMIT and EMMET is the projection weight matrix inside the MLP layer (W_{proj}).

of model editing (Meng et al., 2022a; Gupta and Anumanchipalli, 2024).

- **Score (S)** is a quantify defined by (Meng et al., 2022a) to represent a combination of edit success, generalization and locality. It is the harmonic mean of ES, PS, and NS.

3 Preservation-Memorization : A Unifying Framework for ROME and MEMIT

Both ROME and MEMIT base their work on viewing the weights of the feed-forward layer in a transformer as linear associative memories (Kohonen, 1972; Anderson, 1972). Under this paradigm, linear operations in a transformer (feed-forward layers) are viewed as a key-value store for information. In this section, we re-introduce both ROME and MEMIT in a new light - a unifying conceptual framework of the **preservation-memorization** objective.

Let W represent the weights of the feed-forward layer we want to edit[2], and let k be a key-vector representative of a fact that we are either editing or preserving, and is the input vector to W. The layers being edited are shown in an expanded diagram of a transformer layer (Vaswani et al., 2017) in Figure 2. In the model editing process, the weights of an intermediate layer of the model are changed from W_0 to $\hat{W}$ (W_0 represents the original weights of the W_{proj} matrix), where k_0 is used to indicate a key-vector representing facts we want to preserve from the original model, and k_e being key-vectors representing facts we want to insert into the model. Let v_e be the desired output at the layer being edited corresponding to input k_e such that the correct fact is recalled by the model when finally generating text. A detailed explanation on creation of key-vectors and value-vectors is given in Appendix A.3 and is also briefly depicted in Figure 1.

Our objective is then to preserve the representations of selected input vectors before and after editing, or in other words, minimize the error between $W_0 k_0$ and $\hat{W} k_0$, while forcing the output representation of the vector k_e to be v_e, or in other words - memorizing the fact represented by (k_e, v_e). This process is shown pictorially in Figure 1.

In ROME-style, this objective of model editing is optimized by the following equation:

$$\underset{\hat{W}}{\operatorname{argmin}} \underbrace{\left\| \hat{W} K_0 - W_0 K_0 \right\|_F^2}_{\text{preservation}} \quad \text{s.t.} \quad \underbrace{\hat{W} k_e = v_e}_{\text{memorization}} \tag{1}$$

where $K_0 = [k_1^0 \mid k_2^0 \mid \ldots \mid k_N^0]$ is a matrix containing all the vectors whose representations we want to preserve in a row.

We call this the preservation-memorization objective of model editing because it allows us to retain existing knowledge or skills of a model by keeping the same representations of selected key-vectors before and after editing, while memorizing a new fact k_e, whose representation are forced to be v_e, where v_e is by definition the output representation for k_e that generates the target answer at final layer.

The solution for ROME can then be written as:

$$\hat{W} = W_0 + \Delta \quad \text{where} \tag{2}$$

$$\Delta = (v_e - W_0 k_e) \frac{k_e^T C_0^{-1}}{k_e^T C_0^{-1} k_e} \tag{3}$$

Here, $C_0 = K_0 K_0^T$ is assumed to be an invertible matrix and the denominator $k_e^T C_0^{-1} k_e$ is a scalar.

MEMIT on the other hand optimizes a relaxed version of the same objective:

[2]These layers are found by causal tracing methods (Meng et al., 2022a,b)

$$\underset{\hat{W}}{\operatorname{argmin}} \ \underbrace{\lambda \left\| \hat{W} K_0 - W_0 K_0 \right\|_F^2}_{\text{preservation}} + \underbrace{\left\| \hat{W} K_E - V_E \right\|_F^2}_{\text{memorization}} \tag{4}$$

where $K_E = [k_1^e \ | k_2^e \ | \dots | \ k_E^e]$ is a matrix containing a row of vectors representing the edits we are making in a batch and $V_E = [v_1^e \ | v_2^e \ | \dots | \ v_E^e]$ represents their target representations.

The above optimization objective aims to modify the output representations of vectors in K_E to V_E by minimizing the least square error between them instead of requiring them to be equal with an equality constraint. This is the major difference between the objectives of ROME and MEMIT, where ROME poses the memorization part of the objective as an equality constraint whereas MEMIT relaxes the equality constraint to a least-square objective. This allows Meng et al. (2022b) to find a closed-form solution for making E edits to the model in a single update, represented by the matrix K_E. The solution for the MEMIT objective is:

$$\begin{aligned} &\hat{W} = W_0 + \Delta \quad \text{where} \\ &\Delta = \big(V_E - W_0 K_E\big) K_E^T \big(\lambda C_0 + K_E K_E^T\big)^{-1} \end{aligned} \tag{5}$$

We deliberately write the first term in both solutions in a similar form. The first term in Δ represents the residual error (represented by R) of the new associations (K_E, V_E) when evaluated on the old weights W_0. $R \triangleq v_e - W_0 k_e$ is a vector in case of ROME since we are only able to make singular edits, whereas $R \triangleq V_E - W_0 K_E$ is a matrix for MEMIT consisting of a row of vectors corresponding to each edit in the batch.

To summarize, in this section we show that ROME and MEMIT can be seen as two realizations of the *preservation-memorization* (PM) objective of model editing, where ROME enforces memorization using an equality constraint whereas MEMIT enforces memorization as a least square objective. The least-square constraint in MEMIT allows to reach a closed form solution for batch updates.

4 Edit-Distribution Algorithms

The difference in objectives is not the only difference between ROME and MEMIT. MEMIT (Meng et al., 2022b) also additionally distributes its edits into multiple layers, which has been one of the reasons for success of MEMIT at large batch sizes. This distribution is done by using the formula:

$$\Delta^l = \frac{\big(V_E^L - W_0^l K_E^l\big)}{L - l + 1} {K^l}_E^T \big(C_0^l + K_E^l {K^l}_E^T\big)^{-1} \tag{6}$$

where Δ^l represents the change in weights at layer l, where $l \in \{L-(n-1), L-(n-2), \dots L\}$ represents one of the n layers being edited. $V_E^L = V_E$ are the representations of the fact being edited at the final edit layer, which is represented by L. All other representations of K_E and C_0 are calculated at the layer l being edited. For $n = 1$, the formula reduces to equation 5. We call this algorithm a type of **edit-distribution algorithm**, which is applied post-hoc after finding the closed-form solutions to the PM-objective.

The edit-distribution algorithm is separate from the solutions of the ROME and MEMIT objectives, therefore, we can apply the edit-distribution algorithm when using ROME, as well as use MEMIT without distributing the edits into multiple layers. The formula for using the MEMIT edit-distribution algorithm on ROME is as follows:

$$\Delta^l = (v_e^L - W_0^l k_e^l) \frac{k_e^{l^T} C_0^{l^{-1}}}{k_e^{l^T} C_0^{l^{-1}} k_e^l} \tag{7}$$

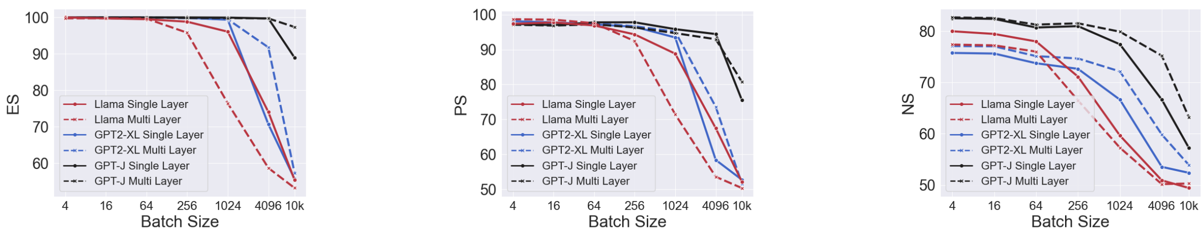
Prior works on model editing do not differentiate between the MEMIT-objective and the edit-distribution algorithm, and as a consequence we never see edits using ROME being distributed to multiple layers or MEMIT being used on only a single layer. The additional wrapping of edit-distribution also makes MEMIT seem distant from ROME. In the next section, we remove the wrapping of edit-distribution from MEMIT and allow for a fair comparison between the two algorithms.

4.1 Impact of edit-distribution Algorithms

The key advantage of the edit-distribution algorithm is apparent when making batched edits. In this section, we perform two experiments to analyze this. First, we compare single edits in ROME and MEMIT with and without edit distribution on 1k randomly selected facts from the CounterFact datase (Meng et al., 2022a). Following that, we compare batched editing in MEMIT with and without edit distribution. Both experiments are performed on three different models - GPT2-XL (1.5B) (Radford et al., 2019), GPT-J (6B) (Wang

Algorithm	Model	Efficacy		Generalization		Locality		Fluency	Score
		ES ↑	EM ↑	PS ↑	PM ↑	NS ↑	NM ↑	GE ↑	S ↑
ROME	GPT2-XL (1.5B)	100.0	99.8	97.9	71.74	75.31	10.48	618.6	89.57
	GPT-J (6B)	100.0	99.8	97.25	73.65	81.94	13.92	617.1	92.34
	Llama-2 (7B)	100.0	99.9	96.7	68.65	80.79	20.62	585.96	91.69
MEMIT	GPT2-XL (1.5B)	100.0	99.7	97.85	71.74	75.21	10.49	618.54	89.51
	GPT-J (6B)	100.0	99.8	97.05	72.25	82.06	13.94	616.6	92.34
	Llama-2 (7B)	99.6	97.4	91.7	57.8	82.83	21.68	593.04	90.86

Table 1: Comparison between ROME and MEMIT when editing only a single layer for CounterFact dataset.



(a) Efficacy Score (ES)

(b) Paraphrase Score (PS)

(c) Neighborhood Score (NS)

Figure 3: Performance comparison of model editing using MEMIT when editing just one layer against multiple layers using the MEMIT edit-distribution algorithm on the CounterFact dataset.

and Komatsuzaki, 2021) and Llama2-7B (Touvron et al., 2023).

The results are shown in Table 1 for edits without distribution and Table 3 (appendix) for edits with distribution. We use the more stable version of ROME called r-ROME as presented in (Gupta and Anumanchipalli, 2024) that does not lead to model collapse and improves generalization. We see that solutions to both ROME and MEMIT objectives perform equally well at making singular edits across different metrics, without needing to distribute the edits to multiple layers. To highlight the usefulness of edit-distribution algorithms, we make batched edits with MEMIT comparing performance with and without edit distribution. The results are shown in Figure 3. When only editing a single layer, we see that MEMIT is able to successfully make batched edits up to a batch size of 1024 for GPT2-XL, 256 for Llama-2-7b and a batch-size as large as 4096 for GPT-J[3]. After this point, the performance of model editing increases when making edits on multiple layers, except for Llama-2-7b. All hyperparameters for all models were chosen as is from prior work (Meng et al., 2022a,b; Yao et al., 2023; Zhang et al., 2024) (appendix A.2).

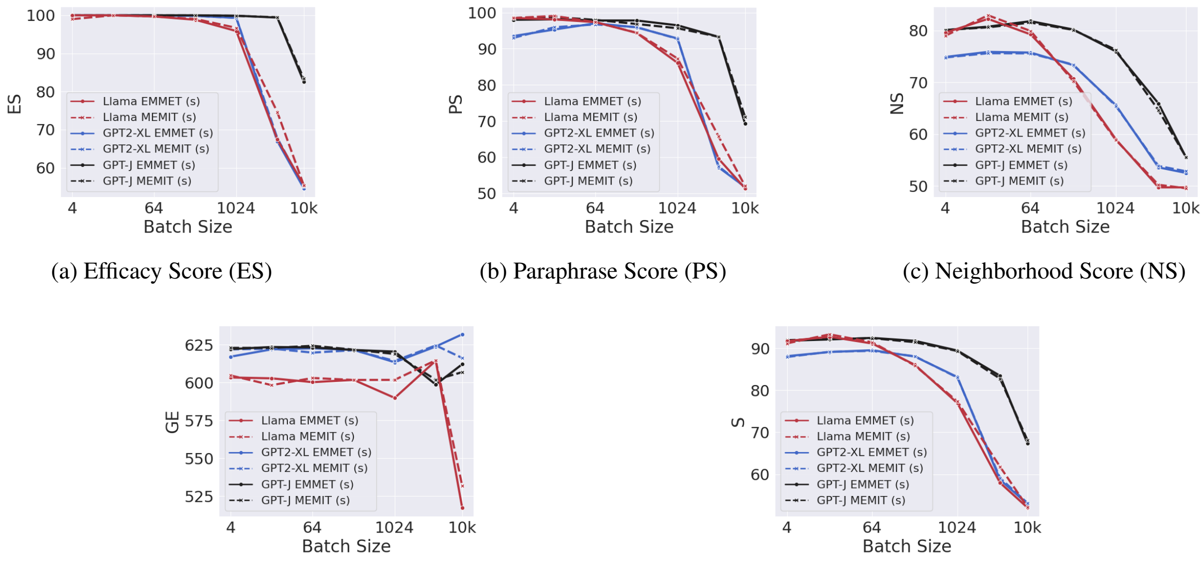
With these experiments, we want to highlight two key points - firstly, when comparing the effectiveness of two optimization objectives, the evaluation should not be conflated with the edit distribution algorithms. After removing the wrapping of edit-distribution from MEMIT, we see that the performance numbers for ROME and MEMIT have an uncanny similarity. Secondly, the MEMIT edit-distribution algorithm is not perfect and currently is the only way to distribute edits into multiple layers, where the residual in the update is distributed with specific ratios through different layers. We hope these experiments will bring more focus to edit distribution algorithms and boost further research in these methods.

5 Introducing EMMET

In section 3, we show that ROME and MEMIT are both algorithms optimizing the preservation-memorization objective of model editing, where ROME does memorization using an equality constraint wherease MEMIT uses a least-square objective for memorization. Thus, we ask the question - *can we perform batched-editing under an equality constraint for memorization?*

In this section, we provide a closed-form solution for batched-editing where memorization is done with equality constraints under the presevation-memorization objective, and thus present a batched-version of ROME, a method we call **EMMET** - **E**quality-constrained **M**ass **M**odel **E**diting in a **T**ransformer.

[3] In our experiments we find GPT-J to be an easier model to edit compared to other models. This is both intriguing but also not the best model to evaluate model editing success.



(a) Efficacy Score (ES) (b) Paraphrase Score (PS) (c) Neighborhood Score (NS)

(d) Generation Entropy (GE) (e) Score (S)

Figure 4: Single layer editing performance of EMMET as a function of batch size when compared to MEMIT on the CounterFact dataset.

Let $K_0 = [k_1^0 \mid k_2^0 \mid \ldots \mid k_N^0]$ represent N key-vectors whose representations we want to preserve. Additionally, let $k_1^e, k_2^e \ldots k_E^e$ represent key-vectors for E facts we want to edit in the model at the same time. Then according to the preservation-memorization objective, we want to find new weights $\hat{W}$ for a weight matrix W_0 such that:

$$\underset{\hat{W}}{\text{argmin}} \quad \underbrace{\left\| \hat{W} K_0 - W_0 K_0 \right\|_F^2}_{\text{preservation}} \quad \text{s.t.} \\ \underbrace{\hat{W} k_i^e = v_i^e \quad \forall i \in [1, 2 \ldots E]}_{\text{memorization}} \tag{8}$$

As can be seen in the above equation, the preservation of representations happens in the first term whereas memorization of all the new facts are forced using an equality constraint in the second term. The above equation is solved using lagrange-multipliers. The derivation of the above equation for the generalized case of batched editing can be found in Appendix A.4.

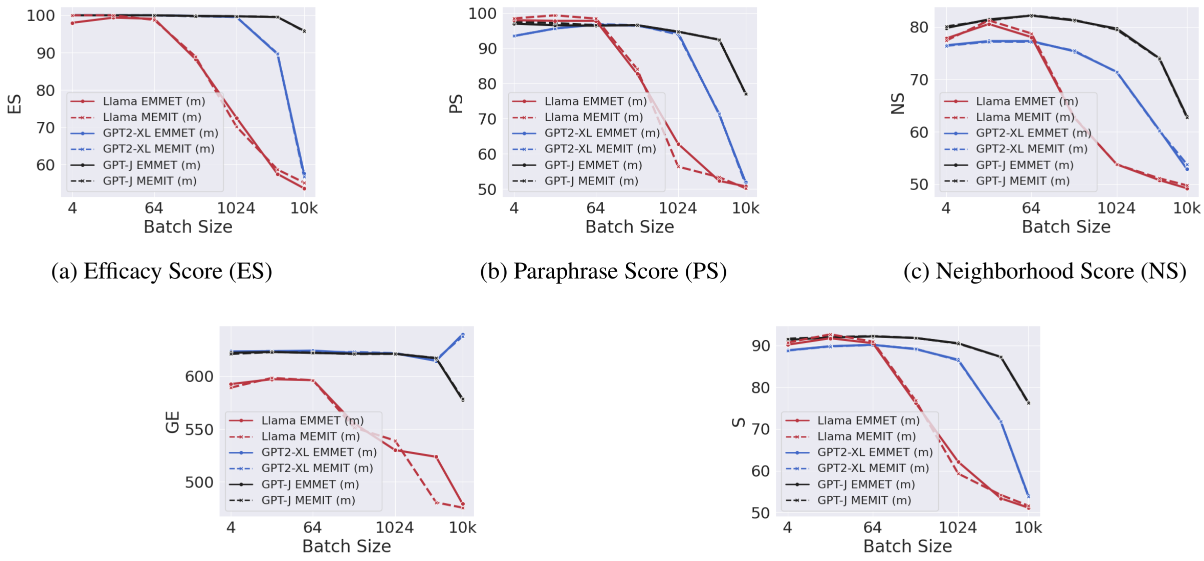
The closed form solution for batched editing with equality-constraint or EMMET is shown below:

$$\hat{W} = W_0 + \Delta \quad \text{where} \\ \Delta = (V_E - W_0 K_E) \left(K_E^T C_0^{-1} K_E \right)^{-1} K_E^T C_0^{-1} \tag{9}$$

Here, $C_0 = K_0 K_0^T$ has the usual meaning as in the derivation of ROME and MEMIT, where K_0 contains the list of representations we want preserved during editing. We write the update equation for EMMET in a familiar form, where the residual $R = V_E - W_0 K_E$ is modified by some matrix operations to update the models with new edits. Additionally, when we put $E = 1$, the K_E matrix reduces to a single vector k_e and equation 9 reduces to the ROME update equation (equation 2). With EMMET, we complete the unification of ROME and MEMIT under the preservation-memorization objective and achieve a symmetry with the usage of these algorithms. EMMET allows for making batched-edits as well as singular when using equality constraints for memorization, much similar to MEMIT with least-square based memorization.

5.1 Stabilizing EMMET

There are two important matrices that are being inverted in EMMET and MEMIT. The first one is $C_0 = K_0 K_0^T$, which is defined identically in both algorithms, whereas $D = K_E^T C_0^{-1} K_E$ is only inverted in EMMET. While the invertibility of both matrices are assumed, they are not always guaranteed. Each of the matrices K_0 or K_E can be written as a row of column vectors as explained in section 3, and thus C_0 can be written as a sum of outer products:



(a) Efficacy Score (ES) (b) Paraphrase Score (PS) (c) Neighborhood Score (NS)

(d) Generation Entropy (GE) (e) Score (S)

Figure 5: Performance comparison of EMMET and MEMIT when distributing the edit over multiple layers using the MEMIT edit-distribution algorithm on the CounterFact dataset.

$$C_0 = K_0 K_0^T = \sum_i k_i^0 k_i^{0^T} \tag{10}$$

where k_i^0 represents a key-vector we want to preserve. For an LLM of dimension d, the dimensionality of a key-vector is usually $4d$ (Figure 2), which is the dimensionality of the square matrix C_0. If C_0 is a $4d$-dimensional square matrix which is a summation of rank-1 matrices, it is invertible as long as there are atleast $4d$-independent vectors in the summation, or $4d$-independent vectors in K_0. For example, for GPT2-XL with hidden dimension of 1600, the dimensionality of key vectors are 6400. So as long as representations of atleast 6400 independent key-vectors are being preserved while editing, C_0 will be an invertible matrix. In practice, we preserve representations of a much larger number of vectors, and hence this condition is always satisfied.

The matrix $D = K_E^T C_0^{-1} K_E$ is a square matrix of dimensionality equal to the number of edits. If given that C_0 is invertible, D is invertible as long as K_E is full-rank, which means all key-vectors corresponding to facts being memorized are independent of each other. While this is not guaranteed, it can be verified before editing and facts corresponding to non-independent keys can be removed from a batch. In practice, we do not find invertibility of D being an issue. However, we find that D is often ill-conditioned, which means that the ratio of the largest and smallest eigenvalues of D explodes. This doesn't necessarily mean that the matrix is singular (non-invertible), but it does mean that numerical computations involving the matrix inverse are unstable and can lead to large numerical errors. To counter this, we set $D = D + \alpha I$, where α is set to 0.1 after an ablation over multiple batch sizes. This allows for stable batched edits using EMMET and also ensures that the D matrix is always invertible.

5.2 Batch Editing with EMMET

We begin by experimenting with EMMET for model editing with varied batch sizes on GPT2-XL, GPT-J and Llama-2-7b on the CounterFact and zsRE (Levy et al., 2017) datasets. The exact implementation details can be found in section A.2. We compare the performance of EMMET and MEMIT on batch sizes up to 10,000 while editing both single (to directly compare the optimization objectives) and multiple layers. The single layer editing comparison between EMMET and MEMIT can be found in Figure 4. We see that both methods have almost identical performance in practice across different metrics. MEMIT performs slightly better than EMMET for Llama-2-7b, as indicated by ES, PS and S metrics. We then apply the MEMIT edit-distribution on EMMET and compare it with MEMIT. The results are shown in Figure 5. We see that in this case, EMMET performs slightly better than MEMIT for Llama-2-7b. The results on the zsRE dataset tell a similar story and can be

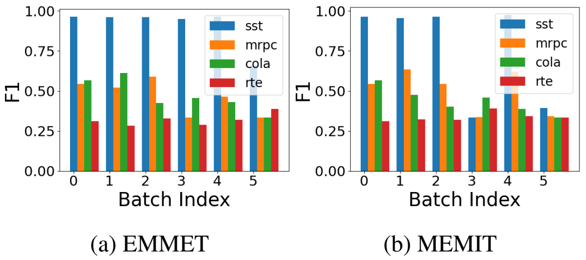


Figure 6: Downstream performance of post-edit Llama2-7b model for EMMET and MEMIT on four GLUE tasks. Batch index 0 refers to downstream performance before editing, with the performance of 5 independent edits of batch size 256.

seen in Figure 7 and 8. The experiments for different hyperparameter values are shown in Appendix A.5. These results present EMMET as a viable new batched-editing algorithm.

Previous work (Gu et al., 2024; Gupta et al., 2024) has shown that model editing is often accompanied by model degradation. This was shown by evaluating the edited model on downstream tasks from the popular GLUE benchmark (Wang et al., 2018). Once we identified that memorization in MEMIT is happening using an approximate least-square constraint rather than an equality constraint, we hypothesised that a possible reason for model degradation could be the use of the least-square constraint. Thus, using an equality constraint, which by definition requires the edit to be exact, may not degrade other knowledge or skills of the model. This was also the motivation behind generalizing ROME to batched edits in the form of EMMET. To test this hypothesis, we adopt the evaluation setting of Gupta et al. (2024) and evaluate both EMMET and MEMIT on four downstream tasks - sentiment analysis (SST2) (Socher et al., 2013), paraphrase detection (MRPC) (Dolan and Brockett, 2005), natural language inference (NLI) (Dagan et al., 2005; Haim et al., 2006; Giampiccolo et al., 2007; Bentivogli et al., 2009) and linguistic acceptability classification (Warstadt et al., 2019) for doing downstream evaluation. The results are shown in Figure 6 for a batch size of 256. The results for other batch sizes can be found in Appendix A.2. We find that both EMMET and MEMIT also degrade the model similarly.

The fact that both EMMET and MEMIT perform editing and degrade the model with an uncanny similarity shows that a "stronger" equality constraint does not enable more accurate model editing. We believe reason behind this is the construction of the key-vector, which is created by taking the average of representations of multiple phrasings of a fact (appendix A.3). This is done to make edits that generalize beyond a single phrasing of a fact. As the key-vector is an averaged representation over randomly selected phrasings, it is an approximation of the ideal vector representation of a fact. We believe that such an approximate representation does not require the additional accuracy of memorization enforced due to the equality constraint. Our findings also indicate that we may be reaching the limit of model editing capabilities under the preservation-memorization objective.

6 Conclusion

In this paper we unite two popular model editing techniques, ROME and MEMIT, under the **preservation-memorization** objective, with ROME performing equality-constrained edits and MEMIT operating under a least-square constraint. We disentangle the *edit-distribution* algorithm proposed in MEMIT from the optimization objective, presenting them as separate entities. We also present EMMET, a new batched-editing algorithm based on the preservation-memorization objective, where memorization happens under an equality constraint. Our experiments show that EMMET has similar performance to MEMIT across multiple dimensions and metrics.

Enabling batched editing with equality-constraint in the form of EMMET allows us to truly unify ROME and MEMIT and shows that both these algorithms are essentially equivalent in terms of their (i) optimization objective, (ii) their abilities (singular and batched editing, a symmetry enabled by EMMET), (iii) their model editing performance and (iv) their limitations (similar model degradation). **EMMET is a cornerstone in completing this larger picture.** These results suggest that EMMET (or ROME) and MEMIT not only have very similar theoretical roots but also perform similarly in practice. The unified framework presented in our work along with the disentanglement of edit distribution algorithm has also enabled a fair comparison between the two algorithms, which was not possible before our work. We hope that this framework facilitates ease of comparison, consistency of implementation, and a much deeper understanding of these model editing methods.

7 Limitations

While our technique may streamline error correction processes, it does not address deeper structural limitations within models, such as edited models inadvertently amplifying existing errors or introducing new inaccuracies. Furthermore, the effectiveness of our method varies depending on the complexity of the model architecture and the nature of the edited knowledge as evidenced by our experiments. Despite having a theoretically 'stronger' memorization objective, EMMET is not able to outperform MEMIT, which also indicates that we might have reached a saturation point for model editing using naive implementations of the preservation-memorization objective, underscoring the fact that significant progress is yet to be made in understanding edit distribution and its implications.

8 Ethical Considerations

While our model editing method allows users to effectively correct for errors or update facts in models, caution is warranted. Our technique also introduces concerns for potential misuse such as malicious actors inserting harmful or false knowledge in LLMs that is absent from the original training data. As such, we warn readers that LLMs should not be considered reliable knowledge bases.

References

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

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. *TAC*, 7:8.

Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. 2023. A brief review of hypernetworks in deep learning. *arXiv preprint arXiv:2306.06955*.

Roi Cohen, Eden Biran, Ori Yoran, Amir Globerson, and Mor Geva. 2023. Evaluating the ripple effects of knowledge editing in language models. *arXiv preprint arXiv:2307.12976*.

Ido Dagan, Oren Glickman, and Bernardo Magnini. 2005. The pascal recognising textual entailment challenge. In *Machine learning challenges workshop*, pages 177–190. Springer.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. 2021. Knowledge neurons in pretrained transformers. *arXiv preprint arXiv:2104.08696*.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. *arXiv preprint arXiv:2104.08164*.

Bill Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Third International Workshop on Paraphrasing (IWP2005)*.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. 2020. Transformer feed-forward layers are key-value memories. *arXiv preprint arXiv:2012.14913*.

Danilo Giampiccolo, Bernardo Magnini, Ido Dagan, and William B Dolan. 2007. The third pascal recognizing textual entailment challenge. In *Proceedings of the ACL-PASCAL workshop on textual entailment and paraphrasing*, pages 1–9.

Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. 2024. Model editing can hurt general abilities of large language models. *arXiv preprint arXiv:2401.04700*.

Akshat Gupta and Gopala Anumanchipalli. 2024. Rebuilding rome: Resolving model collapse during sequential model editing. *arXiv preprint arXiv:2403.07175*.

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

R Bar Haim, Ido Dagan, Bill Dolan, Lisa Ferro, Danilo Giampiccolo, Bernardo Magnini, and Idan Szpektor. 2006. The second pascal recognising textual entailment challenge. In *Proceedings of the Second PASCAL Challenges Workshop on Recognising Textual Entailment*, volume 7, pages 785–794.

Teuvo Kohonen. 1972. Correlation matrix memories. *IEEE transactions on computers*, 100(4):353–359.

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, Chelsea Finn, and Christopher D Manning. 2021. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*.

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.

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 *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642.

Mukund Sundararajan, Ankur Taly, and Qiqi Yan. 2017. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR.

Chenmien Tan, Ge Zhang, and Jie Fu. 2023. Massive editing for large language models via meta learning. *arXiv preprint arXiv:2311.04661*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models, 2023. *URL https://arxiv.org/abs/2307.09288*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

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`.

Alex Warstadt, Amanpreet Singh, and Samuel R Bowman. 2019. Neural network acceptability judgments. *Transactions of the Association for Computational Linguistics*, 7:625–641.

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*.

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

Zexuan Zhong, Zhengxuan Wu, Christopher D Manning, Christopher Potts, and Danqi Chen. 2023. Mquake: Assessing knowledge editing in language models via multi-hop questions. *arXiv preprint arXiv:2305.14795*.

A Appendix

Batch Size	Num Batches	Total Edits
4	25	100
16	10	160
64	5	320
256	5	1280
1024	3	3072
4096	2	8192
10,000	1	10,000

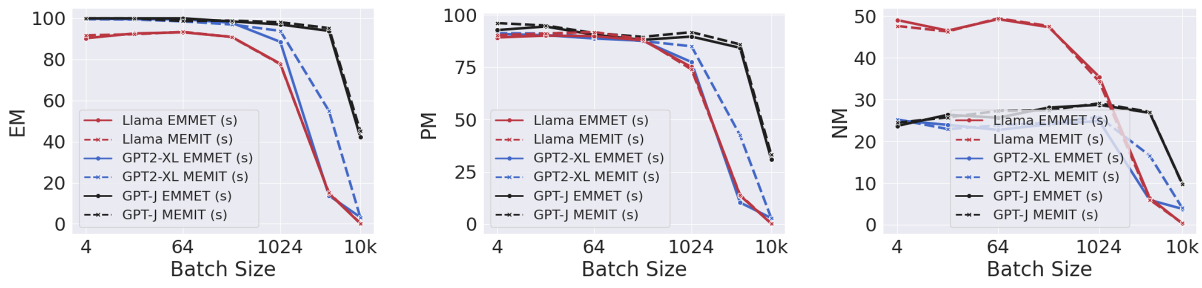
Table 2: Statistics for batch size and number of batches used to create the numbers for this paper.

A.1 Related Work

Model editing methods can be broadly classified into two types - methods that add information in-context (Mitchell et al., 2022; Zhong et al., 2023; Cohen et al., 2023), and methods that modify the parameters of underlying model (De Cao et al., 2021; Mitchell et al., 2021; Meng et al., 2022a,b; Tan et al., 2023). Various model editing techniques have been proposed in the past that tackle this problem in different ways. (Dai et al., 2021) first identify knowledge containing neurons in a model using integrated gradients (Sundararajan et al., 2017) and then modify the selected neurons to edit facts in a model. This method is not scalable with increasing model sizes as it requires us to find activations for each neuron in the model. (De Cao et al., 2021) and (Mitchell et al., 2021) train a hypernetwork (Chauhan et al., 2023) that generates the new weights of the model being edited. While these methods have been optimized to scale with a square-root dependence on the size of the edited model, it still requires training of additional editing models dependent on each source model being edited. Other methods add the most relevant updated knowledge in context (Mitchell et al., 2022; Cohen et al., 2023; Zhong et al., 2023). While such methods provide a viable alternative to model editing, in this paper, we focus on parameter-modifying model editing methods, namely ROME (Meng et al., 2022a) and (Meng et al., 2022b).

A.2 Implementation Details for ROME, MEMIT and EMMET

We use the standard implementation of ROME and MEMIT based on (Meng et al., 2022a) and (Meng et al., 2022b). The range of layers edited for GPT2-XL is $[13, 17]$ (Meng et al., 2022b), for GPT-J is



(a) Efficacy Accuracy (EM) (b) Paraphrase Accuracy (PM) (c) Neighborhood Accuracy (NM)

Figure 7: Single layer editing performance of EMMET as a function of batch size when compared to MEMIT on the zsRE dataset.

Algorithm	Model	Efficacy		Generalization		Locality		Fluency	Score
		ES ↑	EM ↑	PS ↑	PM ↑	NS ↑	NM ↑	GE ↑	S ↑
ROME	GPT2-XL (1.5B)	100.0	99.79	97.78	71.75	76.16	10.93	617.56	89.93
	GPT-J (6B)	100.0	99.8	97.95	72.07	81.46	13.42	615.9	92.35
	Llama-2 (7B)	99.68	92.29	98.1	73.34	77.59	19.07	589.44	90.6
MEMIT	GPT2-XL (1.5B)	100.0	99.79	97.57	71.75	76.14	10.96	617.9	89.87
	GPT-J (6B)	100.0	99.79	97.1	72.86	81.96	14.24	615.97	92.31
	Llama-2 (7B)	99.58	91.34	97.99	72.18	77.8	19.27	589.39	90.63

Table 3: Comparison between ROME and MEMIT when editing multiple layers for the CounterFact dataset.

$[3-8]$ (Meng et al., 2022b) and for Llama-2-7b is $[4-8]$ (Yao et al., 2023; Zhang et al., 2024). In single layer editing experiments, layer 17 was edited for GPT2-XL (Meng et al., 2022a), layer 5 was edited for GPT-J (Meng et al., 2022a), and layer 5 was edited for Llama-2-7b (Yao et al., 2023; Zhang et al., 2024). These choices are directly taken from (Meng et al., 2022a) and (Meng et al., 2022b) for GPT2-XL and GPT-J. We follow the work of (Yao et al., 2023) for choices of layers and hyperparameters for llama-2-7b.

We use the multi-counterfact dataset proposed in Meng et al. (2022b) which is created by removing conflicting facts from the counterfact dataset (Meng et al., 2022a). We then select a random sample of 10,000 facts so that the edits are influenced by the order in which the examples are presented in the dataset. To create the batched editing plots, we create multiple samples for each batch size and average over all the edits made in that set. We use batch sizes of 4, 16, 64, 256, 1024, 4096 and 10k. For each batch size, we use multiple batches and average the evaluation over the total number of batches. The statistics are shown in Table 2. For example, for a batch size of 1024, we first create 3 batches without replacement of size 1024, and perform batched edits on the 3 batches. The numbers are then reported by averaging the performance over 3*1024 facts which were edited in the model. We sample over a few batches so the results are not biased towards a single edited batched. We decrease the number of batches used in the sample due to computational reasons, as the amount of time for each experiment increases with the batch size. The same steps are followed for the zsRE dataset.

A.3 Key-Value creation in ROME/MEMIT

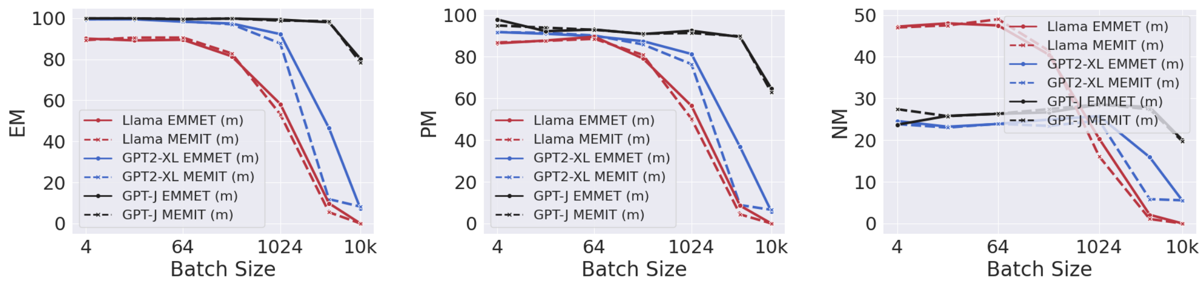
We create key and value vectors for editing using the subject, relation, object framework presented in ROME (Meng et al., 2022a).

Sample queries under this formulation include:

Subject	Prompt	Object
France	"The capital of {S} is {O}"	Paris

Model editing involves manipulating the model such that we're able to alter the object that is associated with a given input subject and prompt. In the table provided, the transformation from "Paris" to "London" exemplifies a potential application of model editing under the (s, r, o) formalization.

The subject and prompt together represent the key vector, which is found by averaging over a set of texts that end with the subject s in the prompt p:



(a) Efficacy Accuracy (EM) (b) Paraphrase Accuracy (PM) (c) Neighborhood Accuracy (NM)

Figure 8: Multi layer editing performance of EMMET as a function of batch size when compared to MEMIT on the zsRE dataset.

$$
\begin{aligned}
k_e &= \frac{1}{N}\sum_{j=1}^{N} k(x_j + p) \\
\text{where } k(x) &= NL(W_{fc}a(x) + b_{fc}) \\
\text{and } a(x) &= LN(\texttt{Att}(h^{l-1}(x)) + h^{l-1}(x))
\end{aligned} \tag{11}
$$

p is the prompt containing the subject and relation, and x_j are 50 generated random sequences with lengths varying from 2 to 10 tokens to make the representation of the key vector more robust to paraphrasing. This also ensures that key vectors for different prompts are distinct enough as two base key vectors (with no random prefix) that have very similar representations move further apart when their representations with a prefix are averaged. LN represents layer normalization and NL is the non-linearity applied to the stream.

Next, we choose a v_e vector such that the new object o^* is output for our k_e vector. We set v_e to minimize the loss as shown:

$$
\begin{aligned}
&\underset{v_e}{\text{argmin}} \quad \frac{1}{N}\sum_{j=1}^{N} -\log \mathbb{P}_{G(h^l=v_e)}[o^* \mid x_j + p] \\
&+D_{KL}\left(\mathbb{P}_{G(h^l=v_e)}[x \mid p'] \;||\; \mathbb{P}_{G(h^l)=v_e}[x \mid p']\right)
\end{aligned} \tag{12}
$$

The first term tries to maximize the probability of the target objective o^* for a prompt of the form $x_j + p$ where p is once again our desired prompt that was also used to generate the key vector. $G(v)$ represents the output of generation s.t. the hidden layer $h^l = v$. The second term tries to minimize the KL divergence when an unrelated prompt p' is input to the model since we want our edit to keep unrelated knowledge unchanged.

We refer readers to the original ROME paper for more details on how key and value vector pairs (k_e, v_e) for editing are generated.

A.4 EMMET Derivation

Let $K_0 = [k_1^0 \mid k_2^0 \mid \ldots \mid k_N^0]$ represent N key-vectors whose representations we want to preserve. Additionally, let $k_1^e, k_2^e \ldots k_E^e$ represent key-vectors for E facts we want to edit in the model at the same time. Then according to the preservation-memorization objective, we want to find new weights $\hat{W}$ for a weight matrix W_0 such that:

$$
\begin{aligned}
&\underset{\hat{W}}{\text{argmin}} \quad \underbrace{\left\|\hat{W}K_0 - W_0K_0\right\|}_{\text{preservation}} \quad \text{s.t.} \\
&\underbrace{\hat{W}k_i^e = v_i^e \quad \forall i \in [1, 2 \ldots E]}_{\text{memorization}}
\end{aligned} \tag{13}
$$

As can be seen in the above equation, the preservation of representations happens in the first term whereas memorization of all the new facts are forced using an equality constraint in the second term. The above equation is solved using lagrange-multipliers. The Lagrangian for the above equation for multiple equality constraints requires a summation of lagrange multipliers and equals:

$$
\begin{aligned}
L(\hat{W}, \lambda_i) &= \frac{1}{2}\hat{W}K_0K_0^T\hat{W}^T - \hat{W}K_0K_0^TW_0^T \\
&+\frac{1}{2}W_0K_0K_0^TW_0^T - \sum_{i=1}^{E}\lambda_i^T(\hat{W}k_i^e - v_i^e)
\end{aligned} \tag{14}
$$

To solve the system of equations, we put $\frac{\delta L}{\delta \hat{W}} = 0$ to get:

$$
\hat{W}K_0K_0^T = W_0K_0K_0^T + \sum_{i=1}^{E}\lambda_i k_i^{eT} \tag{15}
$$

which is same as:

$$(\hat{W} - W_0)K_0K_0^T = \sum_{i=1}^{E} \lambda_i k_i^{eT} = \Lambda K_E^T \quad (16)$$

where $\Lambda = [\lambda_1 \,|\lambda_2 \,|\ldots|\, \lambda_E]$ and $K_E = [k_1^e \,|k_2^e \,|\ldots|\, k_E^e]$. Here, Λ and K_E are matrices created using a row of vectors. We set $K_0K_0^T = C_0$ (assuming that C_0 is invertible[4]) to get the update equation of EMMET:

$$\hat{W} = W_0 + \Lambda K_E^T C_0^{-1} \quad (17)$$

where $\Lambda = [\lambda_1 \,|\lambda_2 \,|\ldots|\, \lambda_E]$, $K_E = [k_1^e \,|k_2^e \,|\ldots|\, k_E^e]$ and $C_0 = K_0K_0^T$.

The unknown matrix of lagrange multipliers (Λ) can be found using the constraint $\hat{W}K_E = V_E$ in the previous equation. It comes out to be:

$$\Lambda = (V_E - W_0K_E)\left(K_E^T C_0^{-1} K_E\right)^{-1} \quad (18)$$

Replacing the above equation in equation 17 gives us the update equation for EMMET:

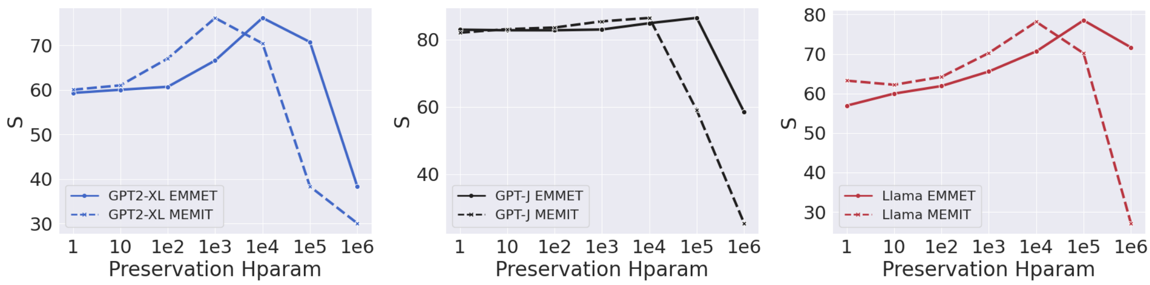
$$\hat{W} = W_0 + \Delta \quad \text{where}$$
$$\Delta = (V_E - W_0K_E)\left(K_E^T C_0^{-1} K_E\right)^{-1} K_E^T C_0^{-1} \quad (19)$$

A.5 EMMET - MEMIT Hyperparameter Comparison

Figures 9 - 16 present the comparison between EMMET and MEMIT for different hyperparameter values. The hyperparameter corresponds to the preservation term in the preservation memorization objective (equation 4). The figures show that both algorithm reach the same peak performance (Figure 9) across all models, but at different hyperparameter values. MEMIT reaches peak performance at lower hyperparameter values, whereas EMMET needs a larger weight for preservation to reach similar performance. This makes sense as EMMET works with a much stronger memorization constraint and thus requires larger weight to preserve the model by the same amount.

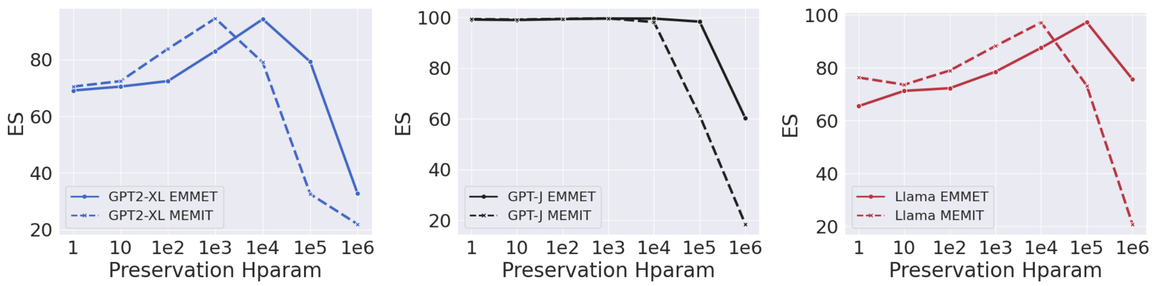
[4]In practice, we find that C_0 is always invertible as long as the number of key-vectors in K_0 are large enough

A.6 EMMET and MEMIT Downstream Performance Comparison



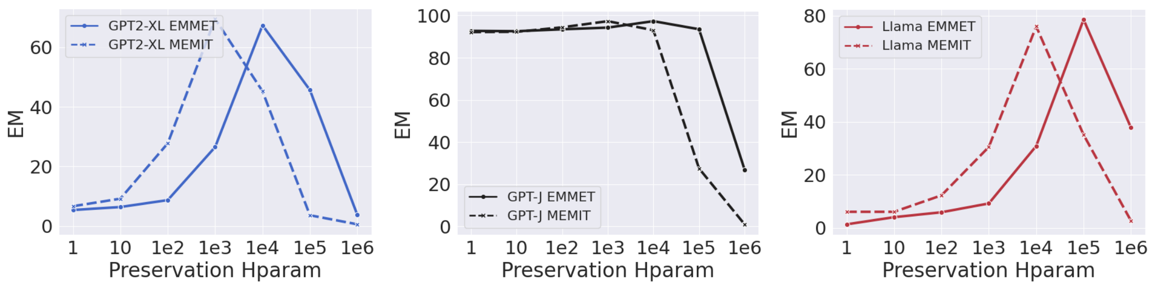
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 9: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Score. Hyperparameter controls the weight of preservation term over memorization term.



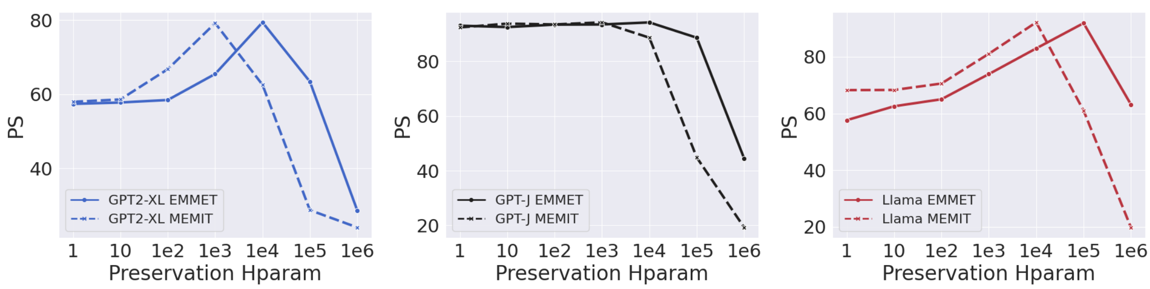
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 10: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Efficacy Score.



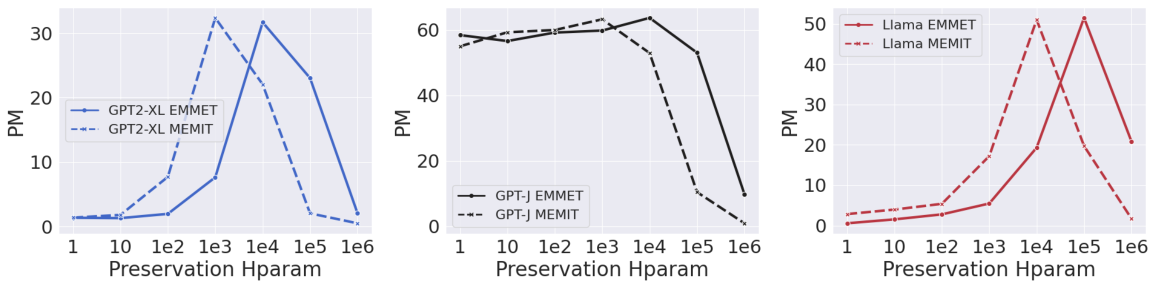
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 11: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Efficacy Magnitude.



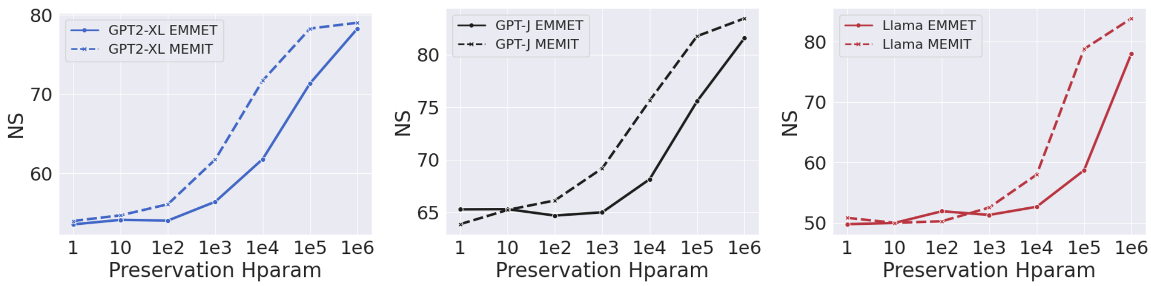
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 12: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Paraphrase Score.



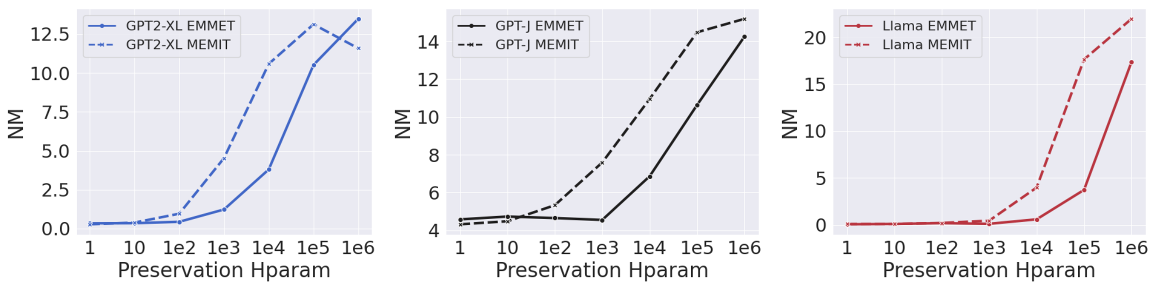
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 13: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Paraphrase Magnitude.



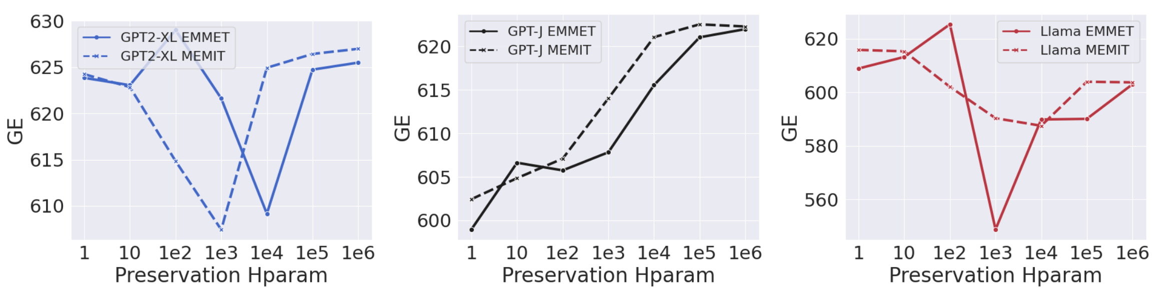
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 14: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Neighborhood Score.



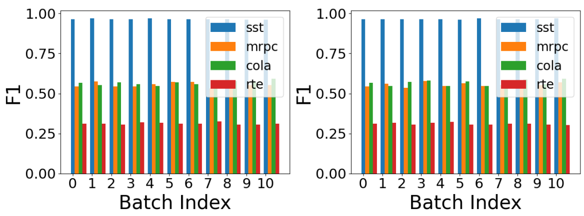
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 15: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Neighborhood Magnitude.



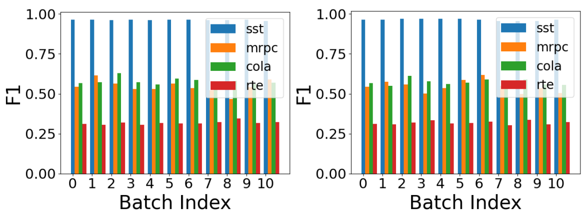
(a) GPT2-XL (b) GPT-J (c) Llama-2-7b

Figure 16: Comparison between EMMET and MEMIT for different hyperparameter values for the metric of Generation Entropy.



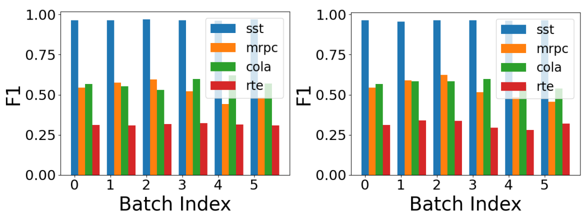
(a) EMMET (b) MEMIT

Figure 17: Model - Llama2-7b. Batch size 4.



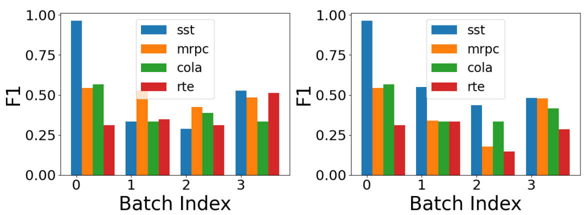
(a) EMMET (b) MEMIT

Figure 18: Model - Llama2-7b. Batch size 16.



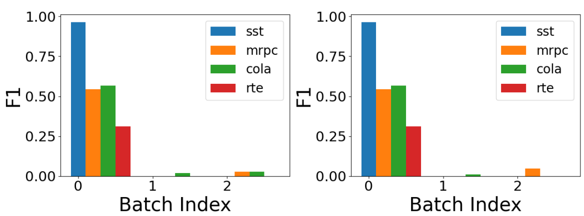
(a) EMMET (b) MEMIT

Figure 19: Model - Llama2-7b. Batch size 64.



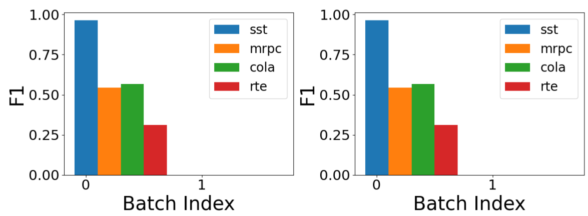
(a) EMMET (b) MEMIT

Figure 20: Model - Llama2-7b. Batch size 1024.



(a) EMMET (b) MEMIT

Figure 21: Model - Llama2-7b. Batch size 4096.



(a) EMMET (b) MEMIT

Figure 22: Model - Llama2-7b. Batch size 10k.