A Unified Framework for Model Editing

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

Abstract

Model editing is a growing area focused on updating the knowledge embedded within models. Among the various methodologies, ROME and MEMIT stand out as leading "locate-and-edit" model editing techniques. While MEMIT enables batched editing of memories, ROME is limited to changing one fact at a time. This paper introduces a unifying framework that brings ROME and MEMIT under a single conceptual umbrella, optimizing for the same goal, which we call the **preservation-memorization** objective. This objective aims to preserve the representations of certain selected vectors while memorizing the representations of new factual information. Specifically, ROME optimizes this objective using an equality constraint, whereas MEMIT employs a more flexible least-square constraint. In addition to making batched edits, MEMIT also edits the model at multiple layers. We disentangle the distribution of edits to multiple layers from the optimization objective of MEMIT and show that these edit-distribution algorithms should be considered separate entities worthy of their own line of research.

Finally, we present **EMMET** - an **E**quality-constrained **M**ass **M**odel **E**diting algorithm for **T**ransformers, a new batched memory-editing algorithm. With EMMET, we present a closed form solution for the equality-constrained version of the *preservation-memorization* objective. We show that EMMET is able to perform batched-edits on par with MEMIT up to a batch-size of 256 and discuss the challenges in stabilizing EMMET. By articulating the "locate-and-edit" model editing algorithms under a simple conceptual framework of *preservation-memorization*, we aim to bridge the gap between intuition and mathematics and hope to simplify the journey for future researchers in model editing.

1 Introduction

As new facts emerge constantly, it's crucial to keep models up-to-date with the latest knowledge. Model editing gives us the ability to edit facts stored inside a model as well as update incorrectly stored facts. 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

Preprint. Under review.

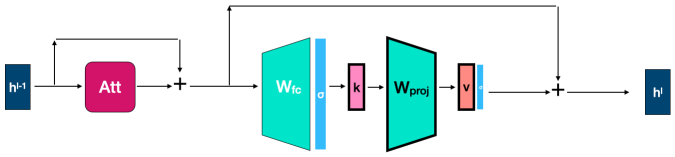


Figure 1: Figure shows a diagrammatic representation of a transformer layer. The layer being edited by ROME, MEMIT and EMMET is the project weight matrix inside the MLP layer (W_{proj}), as highlighted in the figure.

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.

We search for model editing methods that can insert new knowledge into the model weights, while being universally applicable to any model. This goal requires a better understanding of the knowledge storing mechanisms in LLMs and in the process of solving this problem, we also hope to make these models more interpretable. So far, the most promising model editing methods that universally modify model weights are ROME (Rank-One Model Editing) (Meng et al., 2022a; Gupta and Anumanchipalli, 2024) and MEMIT (Mass Editing Memory in Transformer) (Meng et al., 2022b). These methods directly update specific "knowledge-containing" parts of the model without requiring the need to train additional models and can be applied to any transformer based large language model (LLMs). MEMIT also presents the only existing non-hypernetwork based parameter modifying model editing method that allows for *batched edits*.

In this paper, we present a unifying conceptual framework for ROME and MEMIT and show that both methods optimize the same objective. 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. We show that while MEMIT optimizes a more relaxed objective, the power of MEMIT in many cases comes from these **edit-distribution** algorithms. We disentangle the edit-distribution algorithms from the optimization objective of MEMIT and show that these edit-distribution algorithms are universally applicable. Our experiments show that while MEMIT's edit-distribution algorithm helps editing with large batches for GPT2-XL (Radford et al., 2019) and GPT-J (Wang and Komatsuzaki, 2021), it hurts model editing performance when used with Llama-2-7b (Touvron et al., 2023). With this, we advocate for further research into the edit-distribution algorithms and viewing them as separate entities from the optimization objectives.

Finally, we present the closed form solution for making batched edits with the equality-constrained preservation-memorization objective in the form of EMMET - an **E**quality-constrained **M**ass **M**odel **E**diting algorithm for **T**ransformers. EMMET is a new batched-editing algorithm that provides symmetry between the two objectives of "locate-and-edit" class of algorithms and show that batched edits can be made using both objectives. While EMMET does batched editing under an equality constraint, enforcing a "theoretically" more accurate editing of memories, in practice, we find that EMMET performs on-par with MEMIT for batch sizes only up to 256. We believe the reason for this is the hard equality constraints enforced in the optimization objective of EMMET. While EMMET performance does not match MEMIT for larger batch sizes, it can still be used for sequential model editing with smaller batch sizes (Yao et al., 2023; Gupta et al., 2024). The code for EMMET can be found here.

2 Background

Model editing, as traditionally defined[1], is the task of adjusting the model's parameters such that a new fact is incorporated into the model weights without influencing its behavior on other samples (Yao et al., 2023). Facts are usually represented in a key-value format when presented for editing.

[1]We call this the traditional definition of model editing since non-parameter modifying methods that add new knowledge in context are now also considered part of the model editing domain.

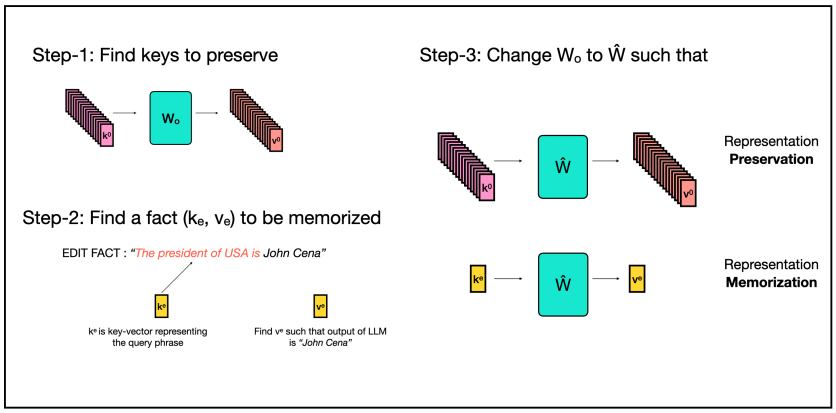


Figure 2: A diagrammatic representation of the preservation-memorization objective.

As an example, let us say we are adding a new fact into the model - *"The president of USA is John Cena"*. Then this fact is represented by a pair of key-value vectors (k_e, v_e), where k_e is the vector representation of the phrase - "The president of USA is" that maximally corresponds to retrieval of the fact, and v_e is the vector representation of the output of the key vector (k_e) at the layer being edited such that "John Cena" is produced as output from the model. This is pictorially represented in step-2 in Figure 2. For a more detailed explanation of creation of key-value vectors, we refer the readers to (Meng et al., 2022a).

While early model editing methods made singular but accurate edits, more recently, focus has shifted to scaling up model editing methods. Scaling of model editing can happen in two ways - either through sequential editing (Yao et al., 2023; Gupta et al., 2024) or batched editing (Meng et al., 2022b). In sequential editing, edits are made to the model sequentially whereas in batched editing, multiple facts are inserted into the model with a single gradient update. MEMIT (Meng et al., 2022b) is one such batched editing algorithm. In this paper, we present EMMET, which is a new batched editing algorithm that inserts multiple facts into the model with one gradient update.

The success of model editing is measured using standard model editing metrics following the work of (Meng et al., 2022a). We use the following five model editing metrics:

- **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 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 effects 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 of model editing.
- **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

ROME and MEMIT are two of the most popular model editing methods. While ROME allows only singular edits, MEMIT allows for editing multiple memories in a model at the same time, thus

providing a scalable solution to model editing. Both algorithms 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 a feed-forward layer in a transformer deemed worthy of editing[2] to modify existing knowledge, 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 in Figure 1. In the model editing process, the weights of an intermediate layer of the model are changed from W_0 to $\hat{W}$, 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. For more details on how we find k_e and v_e, we refer the reader to Meng et al. (2022a).

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 2. This problem can be posed in two different ways. In ROME-style, this objective of model editing is optimized by the following equation:

$$\underset{\hat{W}}{\text{argmin}} \underbrace{\left\|\hat{W}K_0 - W_0 K_0\right\|}_{\text{preservation}} \text{ such that } \underbrace{\hat{W}k_e = v_e}_{\text{memorization}} \tag{1}$$

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. The first term in the above equation represents preservation of representations of selected vectors in K_0 while the second term enforces memorization of the fact represented by (k_e, v_e) using an equality constraint.

Under this objective, we find the new weights $\hat{W}$ such that the output representation of key-vectors in K_0 remain the same before and after editing. This is done to "preserve" the outputs generated for selected key-vectors from the layer being edited. If the outputs for selected key vectors are preserved after editing, it will lead to identical forward propagation for the preserved vectors through the rest of the model which remains unedited, thus leading to the identical outputs post-editing. 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[3] the output representation for k_e that generates the target answer at final layer.

The solution for the above objective can be written as:

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

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:

$$\underset{\hat{W}}{\text{argmin}} \underbrace{\lambda||\hat{W}K_0 - W_0 K_0||}_{\text{preservation}} + \underbrace{||\hat{W}K_E - V_E||}_{\text{memorization}} \tag{3}$$

where $K_E = [k_1^e \mid k_2^e \mid \ldots \mid k_E^e]$ is a matrix containing a row of vectors representing the edits we are making and $V_E = [v_1^e \mid v_2^e \mid \ldots \mid v_E^e]$ represents their target representations. We again see that the

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

[3]v_e is found by a separate optimization objective to make sure it produces the desired output after forward propagation through remaining layers.

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.

first term in the above equation preserves the representation of selected vectors while the second terms forces memorization of facts represented by (K_E, V_E) using a least square constraint.

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:

$$\hat{W} = W_0 + \Delta \quad \text{where} \quad \Delta = \big(V_E - W_0 K_E\big) K_E^T \big(\lambda C_0 + K_E K_E^T\big)^{-1} \tag{4}$$

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. $R \triangleq v_e - W_0 k_e$ 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.

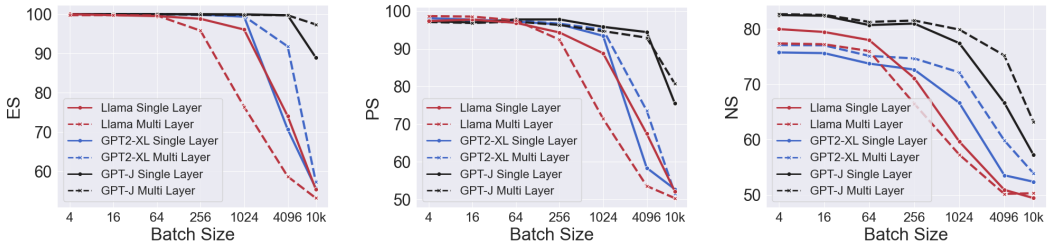
To summarize, ROME and MEMIT can be seen as two realizations of the *preservation-memorization* (PM) objective of model editing, where ROME enforces memorization using an equality constraint whereas MEMIT enforces memorization as a least square objective. The least-square constraint in MEMIT allows to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

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:

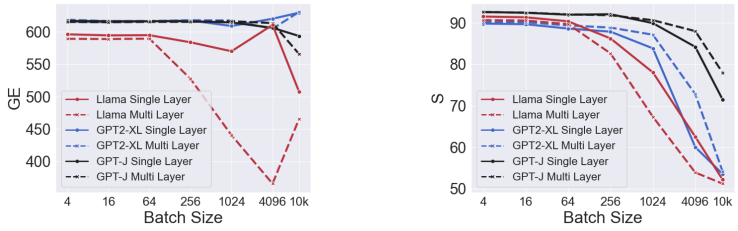
$$\Delta^l = \frac{\big(V_E^L - W_0^l K_E^l\big)}{L - l + 1} {K^l}_E^T \big(C_0^l + K_E^l {K^l}_E^T\big)^{-1} \tag{5}$$

This is almost the same formula as equation 4 with minor changes. Here, 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 4. We call this algorithm the **edit-distribution algorithm**, which is applied post-hoc after finding the closed-form solutions to the PM-objective.

In this paper, we want to disentangle these two different parts of MEMIT. 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:



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



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

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. The multi-layer edits are represented by dotted lines whereas single layer edits are represented by solid lines.

$$\Delta^l = (v_e^l - W_0^l k_e^l) \frac{k_e^{l^T} C_0^{l-1}}{k_e^{l^T} C_0^{l-1} k_e^l} \tag{6}$$

To the best of our knowledge, none of the works so far 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 unified code for the two algorithms can be found here[4]. With this code-base, the MEMIT edit-distribution algorithm can be applied to ROME as well as EMMET which we introduce later.

3.1 Single Layer Edits Using ROME and MEMIT

In this section, we compare the ROME objective with the MEMIT objective, disentangling the effect of the edit-distribution algorithm. We make only single layer edits using the ROME and MEMIT objectives, without distributing the edits into multiple layers. Since ROME does not allow batched-edits, we evaluate models when only singular edits are made to them with both objectives. We evaluate three different models (GPT2-XL, GPT-J and Llama2-7B) on 1k examples selected at random from the CounterFact (Meng et al., 2022a) dataset. The implementation details about selected layers and hyperparameters are provided in section A.1.

The results are shown in Table 1. 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. The opposite test to this would be to make multi-layer edits using both ROME and MEMIT. Edit-distribution algorithms are usually applied to improve performance of model editing when similar performance cannot be reached by just editing one layer. Since the edit scores are close to perfect for singular edits, applying edit-distribution algorithms are not useful and produce similar results. This can be seen in Table 2.

[4] https://github.com/scalable-model-editing/unified-model-editing

3.2 Impact of edit-distribution Algorithms

In the previous section, we see that both ROME and MEMIT objectives perform equally well when making singular edits to the model. We don't see the usefulness of the edit-distribution algorithms when making singular edits. But MEMIT gives us the ability of making batched edits. In this section, we evaluate the performance of making batched edits using MEMIT as a function of the batch size on the CounterFact dataset. We find that as we increase batch-size, editing just a single layer in the model is less effective. Thus we need to edit multiple layers inside a model and hence need edit-distribution algorithms.

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. Here we define edit success by efficacy score, which measures if a new fact was successfully added or corrected inside the model. For GPT2-XL, we see that the edit score improves for batch size greater than 1024 with edit-distribution algorithms. The same is true for GPT-J beyond a batch size of 4096. If we consider all metrics together, as done in the "Score" metric, we can see improvements with edit-distribution starting at batch size 256 for GPT2-XL and GPT-J. This shows that edit-distribution algorithms become important as the batch size of the edits increase.

Throughout our experiments, we've found that ROME and MEMIT perform exceedingly well on GPT-J. While GPT-J is supposed to represent a larger model in the experiments shown in Meng et al. (2022a), it may also be the case that it is an easier model to edit. Because of this, we also perform single-layer and multi-layer editing experiments on Llama-2-7b. The selection of layers for Llama-2-7b has been taken from prior works (Yao et al., 2023; Zhang et al., 2024). We see that edit-distribution using the algorithm proposed in Meng et al. (2022b) hurts the model editing performance for Llama-2-7b, as shown in Figure 3.

With this experiment, we want to highlight two things - firstly, that edit-distribution algorithms are separate entities from the optimization objectives in ROME and MEMIT, and can be removed or added based on need. Secondly, the edit-distribution algorithm proposed in MEMIT is far from perfect. As seen for Llama-2-7b, using the MEMIT edit-distribution algorithm leads to loss of performance against single layer editing. We see lots of papers on model editing algorithms but almost none on edit-distribution algorithms, which were a big factor in editing memories in large batch sizes for MEMIT. We hope that this experiment can promote further research into the edit-distribution algorithms.

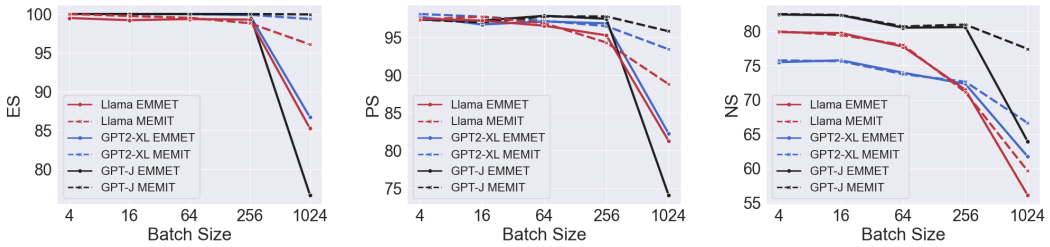
4 Introducing EMMET

In the previous section, 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. The least-square objective allows MEMIT to have a closed form solution for batched-editing, whereas with ROME, we can only edit one fact at a time. Thus, we ask the question - *can we perform batched-editing under an equality constraint for memorization?*

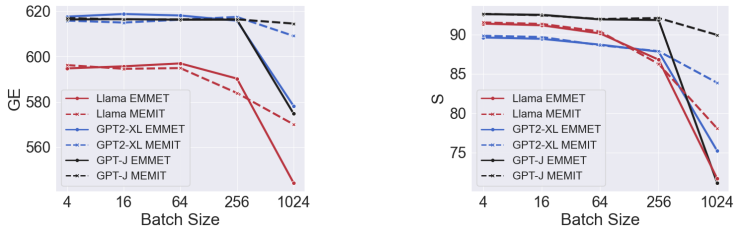
In this section, we provide a closed-form solution for batched-editing where memorization is done with equality constraints under the presevation-memorization objective, and thus present a batched-version of ROME, a method we call **EMMET** - **E**quality-constrained **M**ass **M**odel **E**diting in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

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:

$$\underset{\hat{W}}{\operatorname{argmin}} \underbrace{\left\|\hat{W}K_0 - W_0K_0\right\|}_{\text{preservation}} \quad \text{such that} \quad \underbrace{\hat{W}k_i^e = v_i^e \quad \forall i \in [1, 2 \ldots E]}_{\text{memorization}} \tag{7}$$



(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.

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 constrain in the second term. The above equation is solved using the lagrange-multipliers. The Lagrangian for the above equation for multiple equality constraints requires a summation of lagrange multipliers and can be written as:

$$L(\hat{W}, \lambda_1, \ldots, \lambda_E) = \frac{1}{2}\hat{W}K_0K_0^T\hat{W}^T - \hat{W}K_0K_0^TW_0^T + \frac{1}{2}W_0K_0K_0^TW_0^T - \sum_{i=1}^{E}\lambda_i^T(\hat{W}k_i^e - v_i^e) \tag{8}$$

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

$$\hat{W}K_0K_0^T = W_0K_0K_0^T + \sum_{i=1}^{E}\lambda_i k_i^{e^T} \tag{9}$$

which is same as:

$$(\hat{W} - W_0)K_0K_0^T = \sum_{i=1}^{E}\lambda_i k_i^{e^T} = \Lambda K_E^T \tag{10}$$

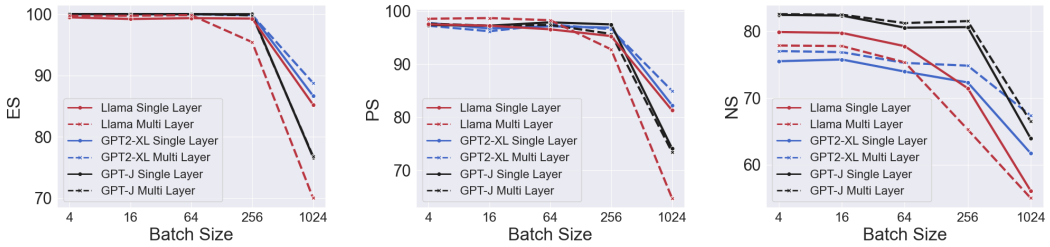
where $\Lambda = [\lambda_1 \ |\lambda_2 \ | \ldots | \ \lambda_E]$ and $K_E = [k_1^e \ |k_2^e \ | \ldots | \ k_E^e]$. Here, Λ and K_E are matrices created using a row of vectors. We set $K_0K_0^T = C_0$ (assuming that C_0 is invertible[5]) to get the update equation of EMMET:

$$\hat{W} = W_0 + \Lambda K_E^T C_0^{-1} \tag{11}$$

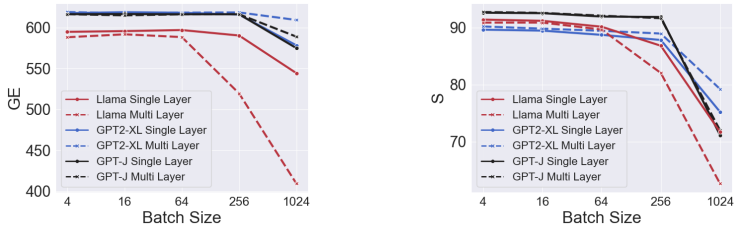
To find the unknown matrix of lagrange multipliers (Λ), we make use of the equality constraint in EMMET. We know that $\hat{W}k_i^e = v_i^e \quad \forall i \in [1, 2 \ldots E]$, or $\hat{W}K_E = V_E$, where $V_E = [v_1^e \ |v_2^e \ | \ldots | \ v_E^e]$. We replace equation 11 into the equality constraint to get:

$$\Lambda K_E^T C_0^{-1} K_E + W_0 K_E = V_E \tag{12}$$

[5]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) Efficacy Score (ES) (b) Paraphrase Score (PS) (c) Neighborhood Score (NS)



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

Figure 5: Performance of EMMET when editing a single layer compared to distributing the edit over multiple layers using the MEMIT edit-distribution algorithm on the CounterFact dataset.

which can be re-arranged to get:

$$\Lambda = (V_E - W_0 K_E) \left(K_E^T C_0^{-1} K_E\right)^{-1} \tag{13}$$

Replacing the above equation in equation 11 gives us the update equation for EMMET:

$$\hat{W} = W_0 + \Delta \quad \text{where} \quad \Delta = (V_E - W_0 K_E) \left(K_E^T C_0^{-1} K_E\right)^{-1} K_E^T C_0^{-1} \tag{14}$$

We write the update equation of 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 14 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. The invertibility of matrix $B = K_E^T C_0^{-1} K_E$ cannot be guaranteed just like C_0, but we find that in practice, B is usually invertible for smaller batch sizes like C_0.

4.1 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 dataset. As shown previously, we disentangle the optimization objective with edit-distribution algorithms and show the comparison between EMMET and MEMIT when editing a single layer. We use identical configurations for both models including the layers being edited. The single layer editing comparison between EMMET and MEMIT can be found in Figure 4.

We find that EMMET performs competitvely with MEMIT up to a batch size of 256, after which the performance for EMMET begins to decline. We see a similar trend when we compare multi-layer experiments between EMMET and MEMIT using the MEMIT edit-distribution algorithm. The results can be seen in Figure 6. We suspect that this happens because the equality constraint in EMMET is too strong to edit a large number of (possibly conflicting) facts at the same time. While this shows that equality constraints for memorization are less effective for model editing for large

batch sizes, increasing batch sizes is not the only way to scale model editing. EMMET can also be used for sequential editing (Yao et al., 2023; Gupta et al., 2024; Gupta and Anumanchipalli, 2024) with smaller batch sizes that fall within its competency range and provide a viable alternate for large scale model editing. We release EMMET to the research community and invite researchers to use it for large-scale model editing. The code base for EMMET is included in this repository - https://github.com/scalable-model-editing/unified-model-editing.

Finally, we compare the performance of EMMET when editing multiple layers using the edit-distribution algorithm proposed in MEMIT. The results can be seen in Figure 5. We make similar observations as before. For GPT2-XL, the edit-distribution algorithm seems to help improve editing performance. For GPT-J, single layer editing performs as well as multi-layer editing as it is an easier model to edit. For Llama-2-7b, we find that using the edit-distribution algorithm can hurt the model performance. This emphasizes our point of viewing edit-distribution algorithms as separate enities from the optimization objectives.

5 Conclusion

In this paper we present a conceptual framework that unites two popular model editing techniques - ROME and MEMIT, under the same objective. We call this objective the **preservation-memorization** objective, where we preserve certain representations of the model while memorizing the representations of a fact we want to store in a model. We show that ROME performs memorization using an equality-constraint, whereas MEMIT does so with a least-square constraint. We hope that this unifying framework improves the intuitive understanding of these algorithms and fuels future research based on both intuition and mathematics.

We also disentangle the *edit-distribution* algorithm proposed in MEMIT from the optimization objective and present them as separate entities. We show that while the MEMIT edit-distribution algorithm helps in editing performance for GPT2-XL and GPT-J at scale, using the same algorithm can also hurt performance as seen for Llama-2-7b. We thus encourage future research to view these as distinct entities and hope this promotes further research into edit-distribution algorithms. A fair comparison of future model editing techniques with MEMIT should be based on the objective of MEMIT and not conflating it with the edit-distribution algorithm.

Finally, we present EMMET - **E**quality-constrained **M**ass **M**odel **E**diting in a **T**ransformer. EMMET is a new batched-editing algorithm based on the preservation-memorization objective where batched-memorization happens under an equality constraint. Our experiments show that EMMET performs competitively with MEMIT for batch sizes up to 256. EMMET is unable to scale beyond this without loss of performance likely due to the hard equality constraints in its objective. Yet EMMET provides a viable alternative to batch editing at smaller batch sizes and can be used in combination with sequential editing to scale model editing.

References

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

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

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

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

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

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

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

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

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

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

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D Manning. Fast model editing at scale. *arXiv preprint arXiv:2110.11309*, 2021.

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

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

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

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

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

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

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. *arXiv preprint arXiv:2305.13172*, 2023.

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

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

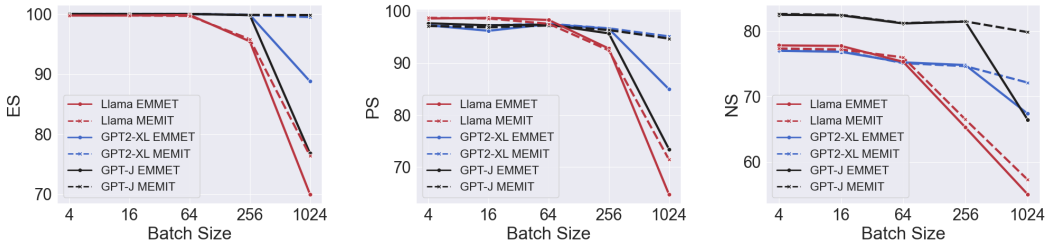
A Appendix

A.1 Implementation Details for ROME, MEMIT and EMMET

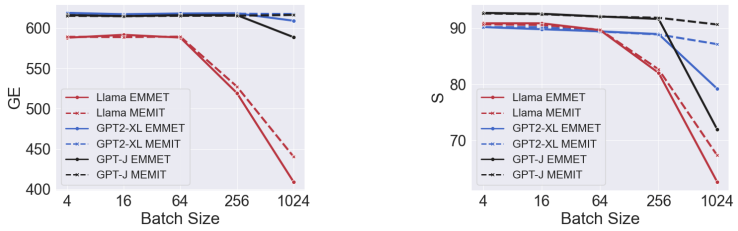
We use the standard implementation of ROME and MEMIT based on Meng et al. (2022a) and Meng et al. (2022b). The range of layers edited for GPT2-XL is $[13, 17]$, for GPT-J is $[3-8]$ and for Llama-2-7b is $[4-8]$, with the final layer in the range being selected for single layer experiments. These choices 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.

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 2: Comparison between ROME and MEMIT when editing multiple layers for CounterFact dataset.



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



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

Figure 6: Comparing EMMET and MEMIT for edits distributed over multiple layers for the CounterFact dataset.