EFFICIENT CLOSED-FORM KNOWLEDGE EDITING FOR MIXTURE-OF-EXPERTS LLMS VIA TUCKER-STRUCTURED UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge editing (KE) offers a cheap substitute for repeatedly fine-tuning LLMs. Most KE methods are built for dense feed-forward layers, while modern LLMs increasingly use Mixture-of-Experts (MoE) architectures which cost less memory and run faster at inference. As a result, a growing family of models has no editing tools designed for them. We introduce a MEMIT-like framework for knowledge editing in MoE-based LLMs. The method builds on the tensor structure of MoE layers to state the editing objective directly at the level of individual experts and uses the Woodbury matrix identity so that the full stacked matrix of expert weights never has to be formed or inverted. The final update reduces to inverting fixed low-rank matrices and needs no extra backward passes. In experiments, our approach stays within a small margin of strong baselines on the main KE metrics while speeding up the editing procedure by up to 3x. The gain comes from the batched MEMIT-style formulation together with the low-dimensional inversions made possible by the Woodbury identity. Closed-form, parameter-modifying KE therefore extends beyond dense layers at low cost and can be applied to modern sparse LLM architectures at scale. Our code is available via GitHub.

1 INTRODUCTION

Large language models (LLMs) have become foundational tools in a wide range of natural language processing applications. However, the factual knowledge encoded in their parameters is inherently static once training concludes, while the world in which they are deployed is not. Maintaining factual accuracy through repeated fine-tuning is prohibitively expensive at the scale of modern LLMs. This holds even for parameter-efficient methods, which still require substantial compute and curated training data (Mitchell et al., 2021) and which risk catastrophic forgetting of previously acquired knowledge when applied repeatedly (Luo et al., 2025).

Knowledge editing (KE) (Wang et al., 2024b) has emerged as a principled response to this challenge. Rather than retraining the model, KE methods aim to inject or overwrite individual factual associations through targeted modifications of model parameters or auxiliary modules. A particularly successful line of work, represented by ROME (Meng et al., 2022a) and its batched extension MEMIT (Meng et al., 2022b), formulates editing as a closed-form least-squares update on the down-projection matrix of a feed-forward (FFN) layer, treated as a linear associative memory (Anderson, 1972; Geva et al., 2021; Sun et al., 2024). These methods are attractive because they are fast, do not require gradient-based optimization of the weights, and admit a clean theoretical interpretation.

A critical limitation is that essentially all closed-form parameter-editing methods have been developed and evaluated on *dense* transformer architectures. Modern leading LLMs increasingly depart from this regime: state-of-the-art systems such as GPT-OSS OpenAI (2025), DeepSeek-v4 DeepSeek-AI (2026), and Qwen3.6 Qwen Team (2026) adopt **Mixture-of-Experts** (MoE) designs (Shazeer et al., 2017; Molodtsov et al., 2026), in which the dense FFN is replaced by a router and a pool of experts, only a small subset of which is activated per token. MoE architectures dominate the current frontier for a clear reason: they decouple parameter count from per-token compute. This enables dramatically larger capacity at fixed inference cost. As a result, in modern MoE-based

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

LLMs, the parameters that previously stored factual associations in dense FFNs are now distributed across a pool of experts gated by a router (Zhao et al., 2024). This mismatch raises a concrete and underexplored question:

How can closed-form, parameter-modifying knowledge editing be performed faithfully and efficiently in MoE-based LLMs?

To the best of our knowledge, the only method developed specifically for this setting is MoE-Edit (Gu et al., 2026). It reformulates the KE objective in a way that faithfully accounts for the MoE structure, and solves the resulting problem by applying MEMIT-like updates. Within each layer, an approximate block coordinate descent (BCD) procedure iterates over experts one at a time, computing per-expert updates that are subsequently projected onto the null-space of the preserved knowledge, following the AlphaEdit Fang et al. (2024) formulation. However, this design imposes a twofold sequential bottleneck: *(i)* activation collection must be repeated layer by layer for every batch of edits, while within each layer *(ii)* the BCD solver iterates sequentially over experts. As the number of edits and the number of experts grow, both factors compound and make the procedure increasingly expensive, motivating a closed-form alternative.

Contributions. In this work, we propose a novel framework tailored to MoE architectures. Our contributions are as follows:

- We formulate the knowledge editing objective for MoE layers in a way that respects the tensor structure of stacked expert weights, avoiding both single-expert and naively concatenated dense surrogates. Exploiting this structure, we recast the problem into a form where the Woodbury identity applies efficiently, yielding a closed-form update that never materializes or inverts the full stacked expert matrix and requires only inversions of size fixed by the number of edits.
- The resulting update rule of **MoE Tucker Editor** (`MoTE`) is fully closed-form and **backward-pass-free**: like MEMIT in the dense case, it requires no gradient computation through the edited layer and inherits the batched-edit property.
- Empirically, `MoTE` achieves editing quality close to MoEEdit on efficacy, generalization, and specificity, while accelerating the editing procedure end-to-end by **up to** $\mathbf{4}\times$, with target-vector computation and all preprocessing included. This speedup is achieved by collecting activations only once at the last layer and finding closed form solution only once for that last layer.

2 Related Work

Locate-and-edit knowledge editing. Following MEMIT, closed-form locate-and-edit methods have been refined in several directions. PMET (Li et al., 2024) optimizes the hidden states of both attention and FFN sublayers but updates only the FFN, EMMET Gupta et al. (2024) unifies ROME and MEMIT under one preservation–memorization objective, and AlphaEdit Fang et al. (2024) projects each update onto the null space of preserved knowledge to stabilize sequential editing. Other lines address the reversal curse (Ma et al., 2023), attach an external memory to frozen weights Hartvigsen et al. (2023); Wang et al. (2024a), or regularize edits to preserve general abilities Gu et al. (2024). All of them target dense FFN layers.

Knowledge editing in MoE models. MoEEdit (Gu et al., 2026) is, to the best of our knowledge, the only closed-form, parameter-modifying editor developed for MoE backbones. It extends the MEMIT objective to the pool of experts (Section 3.2), adapts the AlphaEdit null-space projector to each expert separately, and solves the resulting problem by randomized block coordinate descent over experts, layer by layer. `MoTE` keeps the objective and the per-expert projectors of MoEEdit and differs in three places: a MEMIT-style schedule that solves once at the last edited layer and propagates the solution downward, a Woodbury reduction of the per-layer solve, and a tensor-structured update rule (Section 4).

Tucker decompositions in deep learning. Tucker decomposition has long served as a structural prior on neural-network parameters. The natural modes of trained neural-network weight tensors –

channels, layers, heads, or experts – exhibit a multilinear rank far below their nominal dimension (Gu et al., 2025; Peshekhonov et al., 2024; Li et al., 2026; Yuebin et al.). TD-MoE (Yuebin et al.) unifies all E down-projections of an MoE layer into a single third-order tensor and seeks a joint multilinear approximation, rather than decomposing each expert in isolation. It also shows that factors computed from the raw weights damage the backbone and prescribes multilinear whitening with activation statistics to prevent it. We borrow this whitening recipe (Section 4.3), but use the decomposition to parameterize an edit rather than to compress the model.

3 PRELIMINARIES

3.1 ALGEBRAIC PREREQUISITES

The Woodbury identity. Let $A \in \mathbb{R}^{n\times n}$ be invertible, $U \in \mathbb{R}^{n\times T}$, $V \in \mathbb{R}^{T\times n}$, and $C \in \mathbb{R}^{T\times T}$ invertible. The Woodbury identity (Hager, 1989; Duncan, 1944) states $(A+UCV)^{-1} = A^{-1} - A^{-1}U\big(C^{-1}+VA^{-1}U\big)^{-1}VA^{-1}$. Specializing $A=\lambda I_n$, $C=I_T$, $U=\Psi$, and $V=\Psi^\top$ yields the *push-through corollary*:

$$\Psi^\top\big(\Psi\Psi^\top+\lambda I_n\big)^{-1} \;=\; \big(\Psi^\top\Psi+\lambda I_T\big)^{-1}\Psi^\top, \qquad \forall\,\lambda>0,\ \Psi\in\mathbb{R}^{n\times T}, \tag{1}$$

which trades an $n\times n$ inversion for a $T\times T$ inversion at the cost of two matrix multiplications.

Tucker Decomposition. Let $\mathcal{X}\in\mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ be an N-th order tensor. A *Tucker decomposition* (Tucker, 1963) factors $\mathcal{X}$ into a smaller *core tensor* $\mathcal{G}\in\mathbb{R}^{R_1\times R_2\times\cdots\times R_N}$ multiplied along each mode by a *factor matrix* $U^{(n)}\in\mathbb{R}^{I_n\times R_n}$: $\mathcal{X} \approx \mathcal{G}\times_1 U^{(1)}\times_2 U^{(2)}\times_3\cdots\times_N U^{(N)}$, where $\times_n$ denotes the mode-n product. The tuple $(R_1,\dots,R_N)$ is the *multilinear rank* of the decomposition. Choosing $R_n < I_n$ enforces a low-rank structure along mode n. If the factors $U^{(n)}$ have orthonormal columns, the decomposition is known as the *higher-order SVD* (HOSVD) (De Lathauwer et al., 2000), computed by truncating the SVD of each unfolding $X_{(n)}$.

3.2 CLOSED-FORM MOE EDITING OBJECTIVE

An MoE sublayer replaces the dense FFN with a router and a pool of E experts (Shazeer et al., 2017; Lepikhin et al., 2020; Du et al., 2022). For an input hidden state x, the router selects the top-K experts and assigns them sparse gates g_j with $\sum_j g_j = 1$. Expert j computes its own key $k_j\in\mathbb{R}^{d_{\text{hidden}}}$ and reads it out through its down-projection $W_j\in\mathbb{R}^{d_{\text{model}}\times d_{\text{hidden}}}$. Full definitions of the dense FFN, the router and the experts are given in Appendix A.

Mixture output. The output of the MoE layer is the router-weighted sum of the contributions of the active experts

$$v=\sum_{j=1}^{E} g_j\,v_j=\sum_{j\in S} g_j\,W_j\,k_j, \tag{2}$$

where the second equality follows from the sparsity of $\mathbf{g}$ in (11). Equation (2) makes explicit the structural property that distinguishes MoE editing from the dense case: a single MoE layer is no longer one associative memory W_{down}, but a *mixture of E associative memories* $\{W_j\}_{j=1}^E$, dynamically combined per token by the router.

Gu et al. (2026) recently extended the dense formulation (10) (Appendix A) to MoE layers. Treating the per-expert down-projections $\{W_j\}_{j=1}^E$ jointly as the optimization variable, they cast knowledge editing as the following block-structured regularized problem:

$$\big\{\Delta_j^\star\big\}_{j=1}^E \;=\; \underset{\{\Delta_j\}}{\arg\min}\ \sum_{f\in\mathcal{F}}\Big\|\sum_{j=1}^{E} g_{f,j}\,(W_j+\Delta_j)\,k_{f,j}-v_f\Big\|^2+\sum_{f\in\mathcal{P}}\Big\|\sum_{j=1}^{E} g_{f,j}\,\Delta_j\,k_{f,j}\Big\|^2+\lambda\sum_{j=1}^{E}\|\Delta_j\|^2, \tag{3}$$

where $\mathcal{F}$ is the set of facts to edit, $\mathcal{P}$ the preservation set, and $g_{f,j}$, $k_{f,j}$ the router weight and expert-specific key induced by prompt f on expert j. Stacking the per-expert updates into the vector

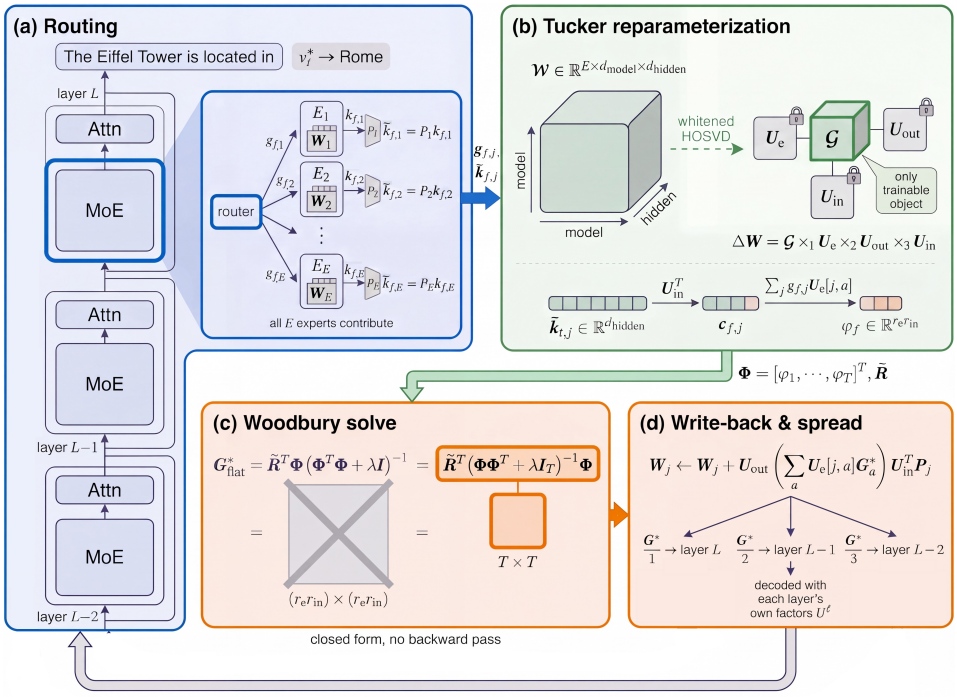


Figure 1: MoTE editing pipeline.

$\theta = \mathrm{vec}\big([\Delta_1 \cdots \Delta_E]\big)$ and defining the per-fact design vector

$$\psi_f = \big[g_{f,1}\, k_{f,1}^\top \;\; \cdots \;\; g_{f,E}\, k_{f,E}^\top\big]^\top \in \mathbb{R}^{E\, d_{\text{hidden}}}, \qquad r_f = v_f - \sum_{j=1}^{E} g_{f,j}\, W_j\, k_{f,j},$$

problem (3) admits the closed-form solution (12) that requires inverting a system of size $(E\, d_{\text{hidden}}) \times (E\, d_{\text{hidden}})$, which is computationally prohibitive at realistic scales. This limitation motivates the block-coordinate treatment developed in the MoE-Edit work.

Preservation through null-space projection. Following AlphaEdit (Fang et al., 2024) and MoEEdit (Gu et al., 2026), the preservation term of (3) is not enforced through an explicit penalty but geometrically, by a per-expert null-space projector P_j. Details on it can be found in AppendixE, or in (Gu et al., 2026), Section 4.2.

4 METHOD

We construct MoTE by composing three ingredients. **(i)** A *Woodbury reduction* that shrinks the per-layer inversion to size $T \times T$, where T is the number of edits in a batch, independent of E and d_{hidden}. **(ii)** A *MEMIT-style updates schedule* that replaces the per-layer solve with spreading deltas across layers. **(iii)** A *Tucker reparameterization* of the update tensor $\Delta\mathcal{W} \in \mathbb{R}^{E \times d_{\text{model}} \times d_{\text{hidden}}}$ that respects the underlying geometric structure of the MoE down-projection. As we show in Section 4.1, ingredients (i) and (ii) together yield a direct MoE adaptation that is computationally tractable but empirically uncompetitive; ingredient (iii), developed in Section 4.3, supplies the missing structural prior. Table 2 adds the three ingredients one at a time.

4.1 WOODBURY REDUCTION AND MULTI-LAYER SPREAD

The closed-form solution of (3) requires inverting a system of size $(E\, d_{\text{hidden}}) \times (E\, d_{\text{hidden}})$, with time and memory costs of $\mathcal{O}((E\, d_{\text{hidden}})^3)$ and $\mathcal{O}((E\, d_{\text{hidden}})^2)$ respectively. However, the Gram

matrix driving the inversion is structurally low-rank: it is a sum of $T = |\mathcal{F}|$ rank-one terms $\psi_f \psi_f^\top$, where T – the number of edits processed in a batch – is several orders of magnitude smaller than $E\, d_{\text{hidden}}$. This is precisely the regime in which the Woodbury identity (1) converts the inversion into a much smaller one.

Stacking the per-fact design vectors and target residuals as

$$\Psi = \big[\psi_1 \ \cdots \ \psi_T\big] \in \mathbb{R}^{E\, d_{\text{hidden}} \times T}, \qquad R = \big[r_1 \ \cdots \ r_T\big] \in \mathbb{R}^{d_{\text{model}} \times T},$$

and unvectorizing $\theta^\star$ into the matrix $\hat{\Delta}^\star = [\Delta_1^\star \ \cdots \ \Delta_E^\star] \in \mathbb{R}^{d_{\text{model}} \times E\, d_{\text{hidden}}}$, the global minimizer of (3) reads

$$\hat{\Delta}^\star \ = \ R\, \Psi^\top \big(\Psi \Psi^\top + \lambda I_{E\, d_{\text{hidden}}}\big)^{-1}.$$

Substituting the push-through corollary (1) into the rightmost factor yields the equivalent but computationally far cheaper expression

$$\boxed{\hat{\Delta}^\star \ = \ R\, \big(\Psi^\top \Psi + \lambda I_T\big)^{-1} \Psi^\top.} \tag{4}$$

The only matrix to invert is now of size $T \times T$, independent of both the number of experts E and the expert hidden dimension d_{hidden}. Concretely, (4) requires forming the kernel matrix $\Psi^\top \Psi \in \mathbb{R}^{T \times T}$ in $\mathcal{O}(T^2\, E\, d_{\text{hidden}})$ time, factorizing a $T \times T$ system in $\mathcal{O}(T^3)$ time, and assembling $\hat{\Delta}^\star$ via two dense products in $\mathcal{O}(T\, d_{\text{model}}\, E\, d_{\text{hidden}})$ time.

Plugging Woodbury into a MEMIT-style multi-layer editor. Equation (4) solves the editing problem at a single critical layer, but locate-then-edit methods derive most of their leverage by spreading an edit across several layers. The naive and natural approach here is to adapt the standard MEMIT (Meng et al., 2022b) schedule. Having solved (3) at the topmost critical layer L via (4), we obtain per-fact target residuals

$$r_f^L \ = \ v_f - \sum_{j=1}^{E} g_{f,j}^L\, W_j^L\, k_{f,j}^L,$$

each of which already aggregates the contributions of all E experts at layer L. We then propagate the edit downward by spreading these residuals across the preceding critical layers $\ell \in \{L_0, \ldots, L\}$ with a coefficient that decays with the distance from L:

$$r_f^\ell \ = \ \frac{r_f^L}{L - \ell + 1}, \qquad \hat{\Delta}^{\ell,\star} \ = \ R^\ell \big(\Psi^L\big)^\top \Big(\Psi^L (\Psi^L)^\top + \lambda I\Big)^{-1}, \tag{5}$$

where $R^\ell = [r_f^\ell]_{f \in \mathcal{F}} \in \mathbb{R}^{d_{\text{model}} \times T}$ stacks the per-layer residuals. Direct adaptation of MEMIT would require substituting Ψ^L in (5) with $\Psi^\ell \in \mathbb{R}^{E\, d_{\text{hidden}} \times T}$ which stacks the per-fact design vectors $\psi_f^\ell = [g_{f,1}^\ell\, k_{f,1}^{\ell\top} \ \cdots \ g_{f,E}^\ell\, k_{f,E}^{\ell\top}]^\top$ – the same construction as in (3), but with the router weights $g_{f,j}^\ell$ and per-expert keys $k_{f,j}^\ell$ recomputed at layer ℓ, since both the routing distribution and the expert activations differ from layer to layer. Opposite to that, our approach allows to avoid additional computations by calculating the solution only once at the last layer and spreading it across previous layers with the same schedule. The denominator $L - \ell + 1$ assigns the largest share to the layer closest to L and gradually attenuates earlier layers, mirroring the original MEMIT schedule. Crucially, each per-layer solve in (5) has the form addressed by (4): applying push-through gives the equivalent $\hat{\Delta}^{\ell,\star} = R^\ell \big((\Psi^L)^\top \Psi^L + \lambda I_T\big)^{-1} (\Psi^L)^\top$, so the only matrices ever inverted across the entire multi-layer pipeline are of size $T \times T$. In this sense, the Woodbury reduction makes possible a direct MoE port of MEMIT – and, more broadly, of any locate-then-edit method that relies on an analogous closed-form ridge solve – computationally tractable out of the box.

4.2 The Tensor-Structure Diagnosis

An empirical surprise. Despite its apparent elegance, the direct combination of the Woodbury reduction (4) with the MEMIT-style multi-layer spread (5) fails to deliver competitive editing quality on MoE backbones. As we report in Section 5.3 (row *MoTE w/o Tucker* of Table 2), this naive adaptation lags substantially behind dense-model expectations on the evaluation. This raises a natural

question: *What aspect of the MoE architecture is this otherwise principled recipe failing to account for?*

We argue that the diagnosis is structural rather than algorithmic. By stacking the per-expert updates $\{\Delta_j\}_{j=1}^{E}$ into a flat matrix $\hat{\Delta} \in \mathbb{R}^{d_{\text{model}} \times E\, d_{\text{hidden}}}$, problem (3) implicitly treats the E experts as independent blocks of an otherwise unstructured linear map. The natural object underlying an MoE layer, however, is not a matrix but a third-order tensor: the per-expert down-projections $\{W_j\}_{j=1}^{E}$ assemble into

$$\mathcal{W} \;\in\; \mathbb{R}^{E \times d_{\text{model}} \times d_{\text{hidden}}},$$

and the corresponding updates are naturally represented as tensors $\Delta\mathcal{W}$. Experts within the same layer are far from arbitrary: they share substantial structure inherited from joint pretraining. The flat block-diagonal solve of (5) discards this structure entirely: its Frobenius regularizer damps every update direction at the same rate, whether it follows the multilinear structure along which experts actually vary or is genuinely unstructured noise. This observation motivates a tensor-aware reformulation, in which the unknown $\Delta\mathcal{W}$ is sought directly in factored form. A Tucker decomposition (Tucker, 1963) is the canonical choice: it disentangles the three modes – expert, model, and hidden – through a core tensor $\mathcal{G}$ and three mode-specific factor matrices, aligning the optimization variable with the multilinear geometry of the MoE layer.

4.3 Tucker-Structured MoE Editor

We now combine the two ingredients on the table. The Woodbury reduction of Section 4.1 makes a per-layer MEMIT solve cheap in the unstructured case, while the diagnosis of Section 4.2 indicates that the unstructured solve is the wrong object: experts co-vary along the multilinear structure of $\mathcal{W}$. We express the editing update $\Delta\mathcal{W}$ in the Tucker basis of $\mathcal{W}$ itself, and solve the resulting MEMIT objective in closed form via the Woodbury push-through.

Stack the per-expert updates into a tensor $\Delta\mathcal{W} \in \mathbb{R}^{E \times d_{\text{model}} \times d_{\text{hidden}}}$ with $\Delta\mathcal{W}[j,:,:] = \widehat{\Delta W}_j$, and constrain it to Tucker form

$$\Delta\mathcal{W} \;=\; \mathcal{G} \times_1 U_e \times_2 U_{\text{out}} \times_3 U_{\text{in}},$$

with factors $U_e \in \mathbb{R}^{E \times r_e}$, $U_{\text{out}} \in \mathbb{R}^{d_{\text{model}} \times r_{\text{out}}}$, $U_{\text{in}} \in \mathbb{R}^{d_{\text{hidden}} \times r_{\text{in}}}$ and core $\mathcal{G} \in \mathbb{R}^{r_e \times r_{\text{out}} \times r_{\text{in}}}$. The per-expert update reads

$$\widehat{\Delta W}_j \;=\; U_{\text{out}} \Big(\sum_{a=1}^{r_e} U_e[j,a]\, G_a \Big) U_{\text{in}}^{\top}, \qquad G_a \coloneqq \mathcal{G}[a,:,:]. \tag{6}$$

The factors $U_e, U_{\text{out}}, U_{\text{in}}$ are fixed, whereas the core $\mathcal{G}$ with $r_e r_{\text{out}} r_{\text{in}}$ free entries is optimized.

Compressed editing objective. Substituting (6) into the per-layer MEMIT objective (5) turns the optimization over $\Delta\mathcal{W}$ into an optimization over the core $\mathcal{G}$. With null-space-projected keys $\widetilde{k}_{t,j} = P_j k_{t,j}$, define the *compressed key* and the *expert-mode aggregate*

$$c_{t,j} \;=\; U_{\text{in}}^{\top} \widetilde{k}_{t,j} \in \mathbb{R}^{r_{\text{in}}}, \qquad \phi_{t,a} \;=\; \sum_{j=1}^{E} g_{t,j}\, U_e[j,a]\, c_{t,j} \in \mathbb{R}^{r_{\text{in}}},$$

and stack them as $\phi_t = [\phi_{t,1}; \ldots; \phi_{t,r_e}] \in \mathbb{R}^{r_e r_{\text{in}}}$. Letting $G_{\text{flat}} = [G_1 \;\cdots\; G_{r_e}] \in \mathbb{R}^{r_{\text{out}} \times r_e r_{\text{in}}}$ denote the mode-1 unfolding of $\mathcal{G}$, a direct calculation gives

$$U_{\text{out}}^{\top} \sum_{j=1}^{E} g_{t,j}\, \widehat{\Delta W}_j\, \widetilde{k}_{t,j} \;=\; G_{\text{flat}}\, \phi_t,$$

so that, after compressing the per-fact target residual analogously, $\widetilde{r}_t = U_{\text{out}}^{\top} r_t \in \mathbb{R}^{r_{\text{out}}}$, the projected MEMIT objective reduces to a ridge regression in the core variable:

$$\min_{G_{\text{flat}}} \; \sum_{t=1}^{T} \big\| \widetilde{r}_t - G_{\text{flat}} \phi_t \big\|_2^2 \;+\; \lambda \big\| G_{\text{flat}} \big\|_F^2. \tag{7}$$

Let $\Phi = [\phi_1, \ldots, \phi_T]^\top \in \mathbb{R}^{T \times r_e r_{\text{in}}}$ and $\widetilde{R} = [\widetilde{r}_1, \ldots, \widetilde{r}_T]^\top \in \mathbb{R}^{T \times r_{\text{out}}}$. The normal equations of (7) read $G_{\text{flat}}(\Phi^\top \Phi + \lambda I_{r_e r_{\text{in}}}) = \widetilde{R}^\top \Phi$, with unique minimizer

$$G^\star_{\text{flat}} \;=\; \widetilde{R}^\top \Phi \left(\Phi^\top \Phi + \lambda I_{r_e r_{\text{in}}}\right)^{-1} \;=\; \widetilde{R}^\top \left(\Phi \Phi^\top + \lambda I_T\right)^{-1} \Phi,$$

where the second equality is the Woodbury push-through (1). Either form yields the same $G^\star_{\text{flat}} \in \mathbb{R}^{r_{\text{out}} \times r_e r_{\text{in}}}$, and we always solve whichever side has the smaller inversion. With full ranks $r_e r_{\text{in}} = E\, d_{\text{hidden}}$, so in our experiments ($T = 50$) the $T \times T$ form is always the cheaper one. At full rank the witness ϕ_t has the same dimension $E\, d_{\text{hidden}}$ as the design vector ψ_t of §4.1, and a single-layer solve costs the same as the unstructured one. What the reparameterization changes is the geometry of the problem: the ridge penalty in (7) acts on the core in the whitened basis of the factors rather than on raw weights, and the core is a layer-agnostic object that can be transferred between layers through each layer's own factors, as described below.

Writeback. Given $G^\star_{\text{flat}}$, the per-expert raw-space update is reconstructed via (6) and composed with the per-expert null-space projector to yield the actual weight increment

$$W_j \;\leftarrow\; W_j \;+\; U_{\text{out}} \Big(\sum_{a=1}^{r_e} U_e[j, a]\, G^\star_a \Big) U_{\text{in}}^\top\, P_j. \tag{8}$$

Implementation details of the write-back are given in Appendix C. The null-space ablation in Appendix D.3 shows that the projector is necessary.

Spreading updates. Iterating (8) across the critical layers $\ell \in \{L_0, \ldots, L\}$ with the MEMIT-style residual schedule of (5) leads to calculating $G^{\star,L}$ only at the last layer L and spreading it across previous layers from editing set with the same schedule:

$$G^{\star,\ell} \;=\; \frac{G^{\star,L}}{L - \ell + 1}, \qquad \hat{\Delta}^{\ell,\star} \;=\; U^\ell_{\text{out}} \Big(\sum_{a=1}^{r_e} U^\ell_e[j, a]\, G^{\star,\ell}_a \Big) (U^\ell_{\text{in}})^\top\, P^l_j, \tag{9}$$

Doing so, factors $U^\ell_{\text{in}}, U^\ell_{\text{out}}, U^\ell_{\text{e}}$ return weight perturbations into each layer's space and provide correct update, while saving time up to n times, where n is a number of edited layers.

5 Experiments

5.1 Setup

Models. Following the MoEEdit Gu et al. (2026), we evaluate three modern MoE LLMs: **Qwen3-30B-A3B-2507** Team (2025) (contains 128 experts; activates top-8 per token), **GPT-OSS-20B** OpenAI (2025) (contains 32 experts; activates top-4 per token), and additionally **Qwen3.6-35B-A3B** Qwen Team (2026) (contains 256 experts; activates top-8 per token + 1 shared expert). Hyperparameters of our method for each model can be found in Appendix C.

Datasets. We evaluate performance on two common knowledge editing benchmarks: **COUNTERFACT** (single-hop counterfactual edit dataset) presented with MEMIT Meng et al. (2022b) and **ZsRE** (zero-shot relation extraction) Levy et al. (2017).

Baselines. We compare our method mostly with MoEEdit framework Gu et al. (2026) since it is the only known method specifically targeting editing of MoE models. As additional competitors, we utilize simple fine-tuning (**FT**), fine-tuning with L_∞-norm constraints (**FT-L**)Zhu et al. (2020), AdaLoRA Zhang et al. (2023), and UnKE Deng et al. (2024).

Metrics. Metrics are standard for the field Meng et al. (2022a;b). We measure **Efficacy**, **Generalization** and **Specificity**. **Utility** shows overall balance between them. Details in Appendix B.

Protocol. Following MoEEdit, every result is obtained in the sequential regime: 1000 facts are edited in 20 batches of 50, and each batch is applied to the already-edited model. MoEEdit and all variants of `MoTE` share the same target vectors $v^\star$. Each backbone uses a single set of hyperparameters in all experiments, selected on a held-out ZsRE validation split and transferred unchanged to COUNTERFACT (Appendix C). Appendix D.6 extends the sequential protocol to 10000 edits.

5.2 MAIN RESULTS

Table 1: Results on COUNTERFACT and ZsRE. The best results are in **bold**, the second best are underlined

Model	Method	COUNTERFACT				ZsRE			
		Eff.↑	Gen.↑	Spe.↑	Uti.↑	Eff.↑	Gen.↑	Spe.↑	Uti.↑
Qwen3-30B-A3B	Pre-edited	13.30±0.34	15.10±0.31	84.45±0.24	37.62	41.30±0.29	40.50±0.28	40.91±0.27	40.90
	FT	80.70±0.39	63.95±0.43	41.44±0.39	62.03	6.44±0.14	6.13±0.14	2.15±0.06	4.91
	FT-L	82.40±0.38	22.75±0.33	71.48±0.25	58.88	44.19±0.29	42.46±0.29	41.92±0.27	42.86
	AdaLoRA	51.90±0.50	49.75±0.40	48.10±0.26	49.92	3.66±0.09	3.60±0.09	4.68±0.10	3.98
	UnKE	89.30±0.31	82.85±0.33	48.15±0.33	73.43	31.43±0.28	29.78±0.27	25.30±0.23	28.84
	MoEEdit	**97.90±0.14**	**86.50±0.30**	**83.45±0.23**	**89.28**	**75.29±0.28**	**67.26±0.32**	**43.18±0.27**	**61.91**
	MoTE	97.00±0.17	82.60±0.33	79.57±0.26	86.39	72.60±0.28	63.93±0.32	42.67±0.28	59.73
GPT-OSS-20B	Pre-edited	11.80±0.32	14.70±0.31	84.53±0.24	37.01	33.20±0.28	32.14±0.28	28.02±0.00	31.12
	FT	83.40±0.37	**58.40±0.42**	55.72±0.33	65.84	25.57±0.28	23.41±0.26	17.61±0.21	22.20
	FT-L	73.80±0.44	43.10±0.48	59.75±0.33	58.88	32.75±0.29	33.09±0.30	30.06±0.26	31.97
	AdaLoRA	62.40±0.48	55.00±0.42	43.65±0.34	53.68	43.46±0.30	42.96±0.30	33.60±0.24	40.01
	UnKE	78.00±0.41	44.40±0.42	73.91±0.28	65.44	46.58±0.31	43.99±0.31	31.40±0.26	40.66
	MoEEdit	**96.90±0.17**	38.80±0.41	80.93±0.26	**72.21**	**68.42±0.33**	59.15±0.35	33.23±0.27	53.60
	MoTE	94.60±0.22	37.55±0.41	**81.85±0.25**	71.33	66.42±0.34	**60.03±0.35**	**37.11±0.28**	**54.52**
Qwen3.6-35B-A3B	Pre-edited	15.10±0.33	16.42±0.30	85.93±0.23	39.15	44.28±0.28	43.61±0.27	42.87±0.26	43.59
	FT	84.75±0.36	66.30±0.41	44.92±0.37	65.32	8.15±0.15	7.42±0.14	3.84±0.08	6.47
	FT-L	85.93±0.35	26.81±0.35	73.62±0.24	62.12	47.88±0.28	45.90±0.28	37.56±0.26	43.78
	AdaLoRA	56.42±0.47	52.91±0.39	50.84±0.25	53.39	7.43±0.12	7.05±0.11	6.88±0.11	7.12
	UnKE	87.84±0.29	74.77±0.31	52.10±0.31	71.57	35.82±0.27	33.74±0.26	28.91±0.22	32.82
	MoEEdit	**93.40±0.25**	**78.55±0.36**	**77.99±0.26**	**83.31**	**61.39±0.32**	**54.89±0.33**	**40.93±0.27**	**52.40**
	MoTE	89.20±0.31	68.44±0.39	75.33±0.27	77.66	58.38±0.43	51.79±0.22	39.03±0.32	49.73

Main metrics for each model and method are shown in Table 1. Results show that our method provides second best edit quality in all setups and even outperform MoEEdit in some scenarios (e.g. GPT-OSS-20B model and ZsRE dataset). With GPT-OSS-20B model our method show higher locality on both datasets, which means better preserving of original model performance. The Utility gap to MoEEdit is between 0.9 and 5.7 points, which is the price of the speedup shown in Table 2.

5.3 TUCKER VS NO TUCKER

Table 2 adds the components of MoTE one at a time. *MoTE w/o Tucker & Woodbury* keeps the BCD solve of MoEEdit but replaces its per-layer loop with the multi-layer schedule (5). *MoTE w/o Tucker* further replaces BCD with the Woodbury solve (4), which reduces the inversion from $(E\, d_{\text{hidden}}) \times (E\, d_{\text{hidden}})$ to $T \times T$. MoTE adds the tensor-structured update (7)–(9). The schedule is MEMIT-style rather than MEMIT: MEMIT recomputes the design vectors at every edited layer, while we solve once at the last layer and propagate the solution downward. The first two ingredients make editing fast but lose up to 10.8 Utility points to MoEEdit, and the tensor-structured update recovers 0.8–5.0 of them. More detailed discussion can be found in Section 4.1.

Total time covers the complete run of 1000 facts: computation of $v^\star$, all preprocessing (activation covariances for the null-space projectors, whitening statistics, HOSVD of the expert tensor), the per-layer solves and the write-back. Since $v^\star$ is shared by all methods, it does not bias the comparison. End-to-end, MoTE is $3.05\times$ faster than MoEEdit on Qwen3-30B-A3B, $1.92\times$ on GPT-OSS-20B and $2.41\times$ on Qwen3.6-35B-A3B. It is also the most memory-demanding variant: the HOSVD factors raise peak memory 10.5–14.5 GB above the model weights, and the whole procedure still fits on a single 80GB GPU. Appendix D.7 isolates the components whose cost depends on the number of experts E: they grow much faster with E for MoEEdit than for MoTE.

Alignment of the updates. To test whether the cheap closed-form update of MoTE recovers the update to which the iterative solver of MoEEdit converges, we compare the two increments directly. On one batch of COUNTERFACT facts for Qwen3-30B-A3B, we compute the full per-layer updates ΔW of both methods, project them into a common 3-dimensional subspace with PCA and measure layer-wise cosine similarity (Table 3). The updates of MoTE and MoEEdit are nearly collinear on

all edited layers. Replacing our schedule with the classical MEMIT spread keeps the alignment but loses the speedup. Grafting our schedule directly onto MoEEdit yields anticorrelated updates on every layer except the last one, where the solve itself is performed. Our schedule therefore works only together with the tensor-structured update.

Table 2: Component ablation on editing 1000 facts from COUNTERFACT (20 sequential batches of 50). Each row adds one component (Section 5.3). Total time is end-to-end and includes the computation of $v^\star$, which is shared by all methods and takes 316.3 / 111.0 / 326.6 s on Qwen3-30B-A3B / GPT-OSS-20B / Qwen3.6-35B-A3B. The model weights occupy 56.92 / 38.96 / 64.56 GB. The best results are in **bold**, the second best are underlined.

Model	Method	COUNTERFACT				Total time, s ↓	Peak memory, GB ↓
		Eff.↑	Gen.↑	Spe.↑	Uti.↑		
Qwen3-30B-A3B	MoEEdit	**97.90**$\pm$**0.14**	**86.50**$\pm$**0.30**	83.45$\pm$0.23	**89.28**	1251.4	63.62
	MoTE w/o Tucker & Woodbury	83.53$\pm$0.33	69.80$\pm$0.40	83.58$\pm$0.23	78.97	553.8	62.57
	MoTE w/o Tucker	94.80$\pm$0.22	69.65$\pm$0.40	**83.70**$\pm$**0.23**	82.72	501.6	**62.01**
	MoTE	97.00$\pm$0.17	82.60$\pm$0.33	79.57$\pm$0.26	86.39	**410.3**	67.45
GPT-OSS-20B	MoEEdit	**96.90**$\pm$**0.17**	38.80$\pm$0.41	80.93$\pm$0.26	**72.21**	373.6	50.24
	MoTE w/o Tucker & Woodbury	89.5$\pm$0.19	**40.85**$\pm$**0.42**	81.73$\pm$0.25	70.69	252.1	49.37
	MoTE w/o Tucker	89.3$\pm$0.22	40.55$\pm$0.43	81.74$\pm$0.28	70.53	225.8	**48.85**
	MoTE	94.60$\pm$0.22	37.55$\pm$0.41	**81.85**$\pm$**0.25**	71.33	**195.0**	53.46
Qwen3.6-35B-A3B	MoEEdit	**93.40**$\pm$**0.25**	**78.55**$\pm$**0.36**	77.99$\pm$0.26	**83.31**	1122.7	70.68
	MoTE w/o Tucker & Woodbury	82.40$\pm$0.38	53.75$\pm$0.43	**81.40**$\pm$**0.24**	72.52	631.9	72.34
	MoTE w/o Tucker	82.30$\pm$0.38	54.35$\pm$0.43	81.33$\pm$0.24	72.66	564.3	**68.89**
	MoTE	89.20$\pm$0.31	68.44$\pm$0.39	75.33$\pm$0.27	77.66	**466.4**	77.06

General capabilities after editing Routing similarity after editing is lower for MoTE than for MoEEdit (Appendix D.1). Routing similarity is a proxy, so we also measure general capabilities directly with MMLU (5-shot, no CoT) after the same 1000 COUNTERFACT edits (Table 4). MoTE degrades the backbone no more than MoEEdit, and slightly less on two of three models. The gap in routing similarity does not translate into a loss of general capability.

Table 3: Layer-wise cosine similarity between per-layer updates ΔW after a joint 3-dimensional PCA projection. Qwen3-30B-A3B, one batch of COUNTERFACT facts, edited layers 3–7.

Layer	3	4	5	6	7
cos(MoEEdit, MoTE)	0.83	0.93	0.99	0.99	0.88
cos(MoEEdit, MoTE w/ MEMIT schedule)	0.80	0.94	0.84	0.87	1.00
cos(MoEEdit, MoEEdit w/ our schedule)	−0.15	−0.52	−0.54	−0.55	0.99

Table 4: MMLU (5-shot, no CoT, %) before and after editing 1000 COUNTERFACT facts; change in brackets.

Model	Pre-edited	MoEEdit	MoTE
Qwen3-30B-A3B	82.15	81.64 (−0.51)	**81.85** (−0.40)
GPT-OSS-20B	49.71	48.56 (−1.15)	**49.07** (−0.64)
Qwen3.6-35B-A3B	82.33	**81.39** (−0.94)	81.20 (−1.13)

6 Conclusion

Beyond the immediate practical benefit of a fast, principled editor for MoE LLMs, our results carry a broader message: closed-form, parameter-modifying knowledge editing is not intrinsically tied to dense architectures. With the right algebraic treatment of the layer's structure, the same family of techniques can be extended to substantially more complex architectures than the dense FFNs in which they were originally developed. We view this as an encouraging sign that the principled, optimization-based branch of the KE literature can keep pace with the architectural evolution of modern LLMs.

7 Limitations

MoTE trails MoEEdit by 0.9–5.7 Utility points and preserves routing less well (Appendix D.1), although it loses no more MMLU accuracy (Section 5.3). The per-layer HOSVD makes it the most memory-demanding variant, and the ridge coefficient λ must be selected per backbone, as for MoEEdit (Appendix D.4). Like all closed-form locate-and-edit methods (Gupta et al., 2025), it degrades over very long edit sequences (Appendix D.6).

AI USE STATEMENT

In this work, we used generative AI tools for proposing or refining hypotheses, designing or providing feedback on research methodology or experiments, polishing the writing, and reviewing the final manuscript. We have not used generative AI tools for desighning and implementing methods. Additionally, we used generative AI tools to create or modify scientific figures or images, draft parts of a research paper and polish the writing. We have reviewed all AI-assisted work. We checked LLM-generated research ideas for potential plagiarism through a manual literature survey, final text was manually reviewed for quality and consistency, Figure 1 was manually reviewed for accuracy. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

Knowledge editing can be used both for lightweight updates of deployed models and for injecting false or harmful associations into them. We study it on public counterfactual benchmarks (COUNTERFACT, ZsRE) and publicly released models; no human subjects or personal data are involved. We call for responsible use of editing methods.

REPRODUCIBILITY STATEMENT

Our code is available at `https://anonymous.4open.science/r/MoTE-FE74/`. Appendix C lists the hardware, precision and all hyperparameters of MoTE and the baselines, the hyperparameter selection protocol, the shared-expert treatment and the timing protocol. Appendix B defines all metrics, and Appendix E derives the update rule step by step. All experiments follow the sequential protocol of Section 5.

REFERENCES

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

Lieven De Lathauwer, Bart De Moor, and Joos Vandewalle. A multilinear singular value decomposition. *SIAM journal on Matrix Analysis and Applications*, 21(4):1253–1278, 2000.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence, 2026.

Jingcheng Deng, Zihao Wei, Liang Pang, Hanxing Ding, Huawei Shen, and Xueqi Cheng. Everything is editable: Extend knowledge editing to unstructured data in large language models. *arXiv preprint arXiv:2405.15349*, 2024.

Nan Du, Yanping Huang, Andrew M Dai, Simon Tong, Dmitry Lepikhin, Yuanzhong Xu, Maxim Krikun, Yanqi Zhou, Adams Wei Yu, Orhan Firat, et al. Glam: Efficient scaling of language models with mixture-of-experts. In *International conference on machine learning*, pp. 5547–5569. PMLR, 2022.

William Jolly Duncan. Lxxviii. some devices for the solution of large sets of simultaneous linear equations: with an appendix on the reciprocation of partitioned matrices. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 35(249):660–670, 1944.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Shi Jie, Xiang Wang, Xiangnan He, and Tat-Seng Chua. Alphaedit: Null-space constrained knowledge editing for language models. *arXiv preprint arXiv:2410.02355*, 2024.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 5484–5495, 2021.

Jia-Chen Gu, Hao-Xiang Xu, Jun-Yu Ma, Pan Lu, Zhen-Hua Ling, Kai-Wei Chang, and Nanyun Peng. Model editing harms general abilities of large language models: Regularization to the rescue. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 16801–16819, 2024.

Yupu Gu, Rongzhe Wei, Andy Zhu, and Pan Li. Moeedit: Efficient and routing-stable knowledge editing for mixture-of-experts llms. *arXiv preprint arXiv:2602.10965*, 2026.

Yuxuan Gu, Wuyang Zhou, Giorgos Iacovides, and Danilo Mandic. Tensorllm: Tensorising multi-head attention for enhanced reasoning and compression in llms. In *2025 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE, 2025.

Akshat Gupta, Dev Sajnani, and Gopala Anumanchipalli. A unified framework for model editing. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 15403–15418, 2024.

Akshat Gupta, Phudish Prateepamornkul, Maochuan Lu, Ahmed Alaa, Thomas Hartvigsen, and Gopala Anumanchipalli. Lifelong knowledge editing requires better regularization. *arXiv preprint arXiv:2502.01636*, 2025.

William W Hager. Updating the inverse of a matrix. *SIAM review*, 31(2):221–239, 1989.

Tom Hartvigsen, Swami Sankaranarayanan, Hamid Palangi, Yoon Kim, and Marzyeh Ghassemi. Aging with grace: Lifelong model editing with discrete key-value adaptors. *Advances in Neural Information Processing Systems*, 36:47934–47959, 2023.

Tamara G Kolda and Brett W Bader. Tensor decompositions and applications. *SIAM review*, 51(3): 455–500, 2009.

Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. Gshard: Scaling giant models with conditional computation and automatic sharding. *arXiv preprint arXiv:2006.16668*, 2020.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pp. 333–342, 2017.

Xiaopeng Li, Shasha Li, Shezheng Song, Jing Yang, Jun Ma, and Jie Yu. Pmet: Precise model editing in a transformer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 18564–18572, 2024.

Yi Li, Zhichun Guo, Miao Yin, and Bingzhe Li. Lestd: Llm compression via learning-based sparse tensor decomposition. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *IEEE Transactions on Audio, Speech and Language Processing*, 2025.

Jun-Yu Ma, Jia-Chen Gu, Zhen-Hua Ling, Quan Liu, and Cong Liu. Untying the reversal curse via bidirectional language model editing. *arXiv preprint arXiv:2310.10322*, 2023.

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.

Gleb Molodtsov, Alexander Miasnikov, and Aleksandr Beznosikov. Hierarchical mixture-of-experts with two-stage optimization, 2026. URL https://arxiv.org/abs/2605.08292.

OpenAI. gpt-oss-120b & gpt-oss-20b model card, 2025. URL https://arxiv.org/abs/2508.10925.

Ivan Peshekhonov, Aleksey Arzhantsev, and Maxim Rakhuba. Training a tucker model with shared factors: a riemannian optimization approach. In *International Conference on Artificial Intelligence and Statistics*, pp. 3304–3312. PMLR, 2024.

Qwen Team. Qwen3.6-35B-A3B: Agentic coding power, now open to all, April 2026. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Kai Sun, Yifan Xu, Hanwen Zha, Yue Liu, and Xin Luna Dong. Head-to-tail: How knowledgeable are large language models (llms)? aka will llms replace knowledge graphs? In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 311–325, 2024.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Ledyard R Tucker. Implications of factor analysis of three-way matrices for measurement of change. *Problems in measuring change*, 15(122-137):3, 1963.

Peng Wang, Zexi Li, Ningyu Zhang, Ziwen Xu, Yunzhi Yao, Yong Jiang, Pengjun Xie, Fei Huang, and Huajun Chen. Wise: Rethinking the knowledge memory for lifelong model editing of large language models. *Advances in Neural Information Processing Systems*, 37:53764–53797, 2024a.

Song Wang, Yaochen Zhu, Haochen Liu, Zaiyi Zheng, Chen Chen, and Jundong Li. Knowledge editing for large language models: A survey. *ACM Computing Surveys*, 57(3):1–37, 2024b.

XU Yuebin, Yanhong Wang, Xuemei Peng, Hui Zang, Chen Minghao, Pengfei Xia, and Zeyi Wen. Td-moe: Tensor decomposition for moe models. In *The Fourteenth International Conference on Learning Representations*.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning. *arXiv preprint arXiv:2303.10512*, 2023.

Zhongyu Zhao, Menghang Dong, Rongyu Zhang, Wenzhao Zheng, Yunpeng Zhang, Huanrui Yang, Dalong Du, Kurt Keutzer, and Shanghang Zhang. Factorllm: Factorizing knowledge via mixture of experts for large language models. *arXiv preprint arXiv:2408.11855*, 2024.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. Modifying memories in transformer models. *arXiv preprint arXiv:2012.00363*, 2020.

A Background on dense and MoE layers

A.1 Dense FFN as a Linear Associative Memory

A standard Transformer block alternates a multi-head self-attention (MHSA) sublayer with a position-wise feed-forward network (FFN). In modern gated variants, the FFN consists of three linear maps: an *up-projection* $W_{\text{up}} \in \mathbb{R}^{d_{\text{hidden}} \times d_{\text{model}}}$, a *gate projection* $W_{\text{gate}} \in \mathbb{R}^{d_{\text{hidden}} \times d_{\text{model}}}$, and a *down-projection* $W_{\text{down}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{hidden}}}$.

Following (Geva et al., 2021; Meng et al., 2022a), the down-projection admits a natural interpretation as a *linear associative memory*: each of its d_{hidden} columns acts as a value vector associated with a learned key direction in the gated activation space. Concretely, given a hidden state $u \in \mathbb{R}^{d_{\text{model}}}$, the FFN first builds a key

$$k = \sigma(W_{\text{gate}}\, u) \odot (W_{\text{up}}\, u) \;\in\; \mathbb{R}^{d_{\text{hidden}}},$$

where $\sigma(\cdot)$ is a non-linearity and $\odot$ denotes the Hadamard product. The FFN output is then a linear combination of the columns of W_{down} weighted by k:

$$v = W_{\text{down}}\, k \;\in\; \mathbb{R}^{d_{\text{model}}}.$$

Knowledge editing as a regularized least-squares problem. Knowledge editing seeks to install a new association into this memory. For a target fact (s, r, o_{new}) – a subject s, a relation r, and a desired new object o_{new} – the goal is to modify the model so that the key $k_{(s,r)}$ corresponding to the prompt encoding (s, r) is mapped to a new target vector $z \in \mathbb{R}^{d_{\text{model}}}$ that steers downstream computation toward o_{new}. Two competing requirements must be balanced. The update should be *effective* on the edited fact and its paraphrases (efficacy and generalization), and *local*, i.e. leave the model's predictions on unrelated inputs essentially unchanged (specificity).

In the dense case, this trade-off is naturally expressed as a regularized least-squares problem over an additive perturbation $\Delta \in \mathbb{R}^{d_{\text{model}} \times d_{\text{hidden}}}$:

$$\Delta^\star = \arg\min_{\Delta} \underbrace{\left\|(W_{\text{down}} + \Delta)\, K_1 - V_1\right\|_F^2}_{\text{memorization}} + \underbrace{\left\|\Delta\, K_0\right\|_F^2}_{\text{preservation}} + \lambda \left\|\Delta\right\|_F^2, \tag{10}$$

where $K_1 \in \mathbb{R}^{d_{\text{hidden}} \times T}$ stacks the keys of the T edited facts column-wise, $V_1 \in \mathbb{R}^{d_{\text{model}} \times T}$ contains the corresponding target values, and $K_0 \in \mathbb{R}^{d_{\text{hidden}} \times M}$ is a preservation key set sampled from the model's empirical activation distribution on a large generic corpus. The Frobenius regularizer $\lambda > 0$ controls the magnitude of the update. Problem (10) admits a closed-form solution involving only a single $d_{\text{hidden}} \times d_{\text{hidden}}$ inversion, which underlies the ROME (Meng et al., 2022a) and MEMIT (Meng et al., 2022b) families of methods.

A.2 MoE Layer as a Mixture of Expert Associative Memories

Modern frontier LLMs replace the dense FFN with a *Mixture-of-Experts* (MoE) sublayer (Shazeer et al., 2017; Lepikhin et al., 2020; Du et al., 2022). The motivation is well known: MoE decouples a model's parameter count from its per-token compute. A pool of E specialized FFN-shaped experts collectively stores far more parameters than a dense FFN of the same compute budget, while only $K \ll E$ of them are activated per token via a sparse routing mechanism.

Router. At the core of the layer is a *router*, a lightweight learned function that determines which experts process each token. Given an input hidden state $x \in \mathbb{R}^{d_{\text{model}}}$, the router produces affinity logits with respect to E learned expert embeddings $e_1, \ldots, e_E \in \mathbb{R}^{d_{\text{model}}}$:

$$s_j = x^\top e_j, \qquad j = 1, \ldots, E.$$

A top-K operator selects the indices of the K largest logits,

$$S = \mathrm{TopK}\big(\{s_1, \ldots, s_E\}\big) \subset \{1, \ldots, E\}, \qquad |S| = K,$$

and the corresponding logits are renormalized into a sparse gating distribution over the selected experts:

$$g_j = \begin{cases} \dfrac{\exp(s_j)}{\sum_{\ell \in S} \exp(s_\ell)}, & j \in S, \\ 0, & j \notin S. \end{cases} \tag{11}$$

By construction, $\sum_j g_j = 1$ and the gating vector $\mathbf{g} = (g_1, \ldots, g_E)$ is K-sparse.

Experts. Each expert $j \in \{1, \ldots, E\}$ implements a self-contained gated FFN with its own parameters $\big(W_{\text{gate}}^{(j)},\ W_{\text{up}}^{(j)},\ W_{\text{down}}^{(j)}\big)$. By analogy with the dense case, expert j first computes its own key

$$k_j = \sigma\Big(W_{\text{gate}}^{(j)}\, x\Big) \odot \Big(W_{\text{up}}^{(j)}\, x\Big) \;\in\; \mathbb{R}^{d_{\text{hidden}}},$$

and then produces its contribution by reading out from its own associative memory:

$$v_j(x) = W_j\, k_j, \qquad W_j \coloneqq W_{\text{down}}^{(j)} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{hidden}}}.$$

Crucially, both the keys k_j *and* the values v_j are expert-specific: different experts construct different gated activations from the same input u and project them through different down-projections.

The layer output is the router-weighted sum (2) of the contributions of the active experts.

A.3 CLOSED-FORM SOLUTION.

Equation (3) admits the closed-form solution:

$$\theta^\star = \left(\sum_{f \in \mathcal{F}} (\psi_f \psi_f^\top) \otimes I_{d_{\text{model}}} \; + \; \lambda I \right)^{-1} \sum_{f \in \mathcal{F}} (\psi_f \otimes I_{d_{\text{model}}})\, r_f, \quad \Delta^\star = \texttt{unvec}(\theta^\star), \tag{12}$$

where $\otimes$ denotes the Kronecker product.

B METRICS MEASUREMENT

B.1 ZSRE EVALUATION PROTOCOL

Following prior work on model editing, we define a language model $\mathcal{M}$, an edit request (s_i, r_i), a desired target response o_{new}, and the model's original prediction o_{true}.

Efficacy. Edit success measures whether the edited model greedy predicts the target answer on the modified prompt:

$$\mathbb{E}_i \left[\mathbf{1} \left(o_{\text{new}} = \arg\max_t P_{\mathcal{M}}(t \mid (s_i, r_i)) \right) \right].$$

Generalization. We evaluate whether the edit transfers to paraphrased variants $\widehat{\mathcal{P}}(s_i, r_i)$ of the original prompt:

$$\mathbb{E}_i \left[\mathbf{1} \left(o_{\text{new}} = \arg\max_t P_{\mathcal{M}}(t \mid \widehat{\mathcal{P}}(s_i, r_i)) \right) \right].$$

Specificity. Locality preservation quantifies whether unrelated prompts $\mathcal{U}(s_i, r_i)$ remain unaffected after editing:

$$\mathbb{E}_i \left[\mathbf{1} \left(o_{\text{true}} = \arg\max_t P_{\mathcal{M}}(t \mid \mathcal{U}(s_i, r_i)) \right) \right].$$

B.2 COUNTERFACT EVALUATION PROTOCOL

For CounterFact, we adopt probability-based metrics under the same setup (s_i, r_i) with target answer o_{new} and original prediction o_{true}.

Efficacy. This metric evaluates whether the edited model assigns higher likelihood to the target answer than to the original prediction:

$$\mathbb{E}_i \left[P_{\mathcal{M}}(o_{\text{new}} \mid (s_i, r_i)) > P_{\mathcal{M}}(o_{\text{true}} \mid (s_i, r_i)) \right].$$

Generalization. We measure whether paraphrased prompts $\widehat{\mathcal{P}}(s_i, r_i)$ preserve the edited knowledge:

$$\mathbb{E}_i \left[P_{\mathcal{M}}(o_{\text{new}} \mid \widehat{\mathcal{P}}(s_i, r_i)) > P_{\mathcal{M}}(o_{\text{true}} \mid \widehat{\mathcal{P}}(s_i, r_i)) \right].$$

Specificity. For neighboring prompts $\mathcal{U}(s_i, r_i)$ involving related but distinct entities, specificity measures whether the original behavior is preserved:

$$\mathbb{E}_i \left[P_{\mathcal{M}}(o_{\text{true}} \mid \mathcal{U}(s_i, r_i)) > P_{\mathcal{M}}(o_{\text{new}} \mid \mathcal{U}(s_i, r_i)) \right].$$

C METHODS HYPERPARAMETERS

Hardware and Precision. All experiments are executed on a single machine equipped with an NVIDIA H200 GPU. Model parameters are stored in BF16 precision to reduce memory usage, while all optimizer states, collected activations and gradient updates are maintained in FP32 for improved numerical stability.

Fine-Tuning Baselines. We evaluate both standard fine-tuning (FT) and constrained fine-tuning (FT-L). The FT-L variant restricts the magnitude of parameter updates through a L_∞-norm constraint ε.

For all three models Qwen3-30B-A3B, GPT-OSS-20B and Qwen3.6-35B-A3B we use $\varepsilon = 10^{-3}$ and a learning rate of 10^{-3}. Parameter updates are applied to last layer presented in Table 5.

Training is conducted for 25 epochs with both weight decay and KL regularization disabled.

UnKE. Since UnKE relies on a two-stage optimization procedure, we configure separate hyper-parameters for each phase.

For both Qwen models, the first stage uses a learning rate of 5×10^{-1} with 25 optimization steps and weight decay 10^{-3}. The second stage adopts a learning rate of 2×10^{-4} for 50 additional steps.

For GPT-OSS-20B, the first stage also employs a learning rate of 5×10^{-1} but increases the number of optimization steps to 50, while retaining the same weight decay coefficient. The second stage uses a learning rate of 10^{-4} with 50 optimization steps.

For consistency, all UnKE updates are restricted to the same last layer, as the other baselines. Optimization is performed on the final subject token, following the original structured editing setup.

AdaLoRA. For AdaLoRA, trainable adapters are inserted into all transformer layers. We set the scaling factor $\alpha = 32$ and rank $r = 8$.

The learning rate is set to 5×10^{-3} for Qwen models and 5×10^{-4} for GPT-OSS-20B. Optimization is performed for 25 update steps for all models.

MoEEdit. Layers modified with this method are the same as we show in Table 5.

On Qwen3-30B-A3B, optimization is performed for 25 steps with learning rate 0.1 and 4 Block Coordinate Descent (BCD) iterations. For Qwen3.6-35B-A3B we reduce number of iterations by half due to twice as large number of experts. For GPT-OSS-20B, we use 50 optimization steps, learning rate 0.2, and 10 BCD iterations.

For all models, we set the regularization coefficient $\lambda = 1$ and the KL penalty weight to 0.0625.

To estimate the covariance matrix used in null-space projection, we sample 100K activations. The projection threshold is fixed to 0.02.

MoTE. For this method we utilize the same target vector v^* as for MoEEdit, so all related parameters remain the same, including the settings of the $v^\star$ optimizer. Main differences are shown in Table 5. All experiments use full multilinear ranks $(r_e, r_{\text{out}}, r_{\text{in}}) = (E, d_{\text{model}}, d_{\text{hidden}})$ and *in* whitening.

Full multilinear ranks. In all experiments we use the full multilinear ranks $(r_e, r_{\text{out}}, r_{\text{in}}) = (E, d_{\text{model}}, d_{\text{hidden}})$. Truncating them buys almost no time, because after the push-through (1) the inversion is $T \times T$ regardless of the ranks. It does cost quality: reducing r_e by as little as 10% already lowers Utility by 10–15%, since the expert mode is where the coupling between experts lives. The factorization therefore acts as a change of basis aligned with the multilinear geometry of the layer, not as a compression device.

Factor extraction. We obtain $U_e, U_{\text{out}}, U_{\text{in}}$ once at initialization by HOSVD (De Lathauwer et al., 2000) on $\mathcal{W}$ itself, computed via the Gram trick: $U_n = \text{TopEig}_{r_n}(\mathcal{W}_{(n)}\mathcal{W}_{(n)}^\top)$ for $n \in \{e, \text{out}, \text{in}\}$, optionally refined by a few HOOI (Kolda & Bader, 2009) sweeps. Following TD-MoE (Yuebin et al.), we additionally support multilinear whitening of $\mathcal{W}$ before HOSVD using mode-wise activation covariances $\Sigma_{\text{out}}, \Sigma_{\text{in}}$, so that the factors are aligned with the data geometry rather than with the raw weight geometry; the recovered factors are subsequently re-coloured to the original space. We refer to the paper (Yuebin et al.) for the full whitening recipe. The statistics are collected on a general text corpus before editing and are independent of the edit set. Whitening does not help editing directly. Its role is to keep the reparameterized backbone intact: without it, Specificity drops

Table 5: Hyperparameters for MoTE method

Model	λ	Layers	$\varepsilon_{\text{whitening}}$
Qwen3-30B-A3B	0.1	3, 4, 5, 6, 7	1×10^{-5}
GPT-OSS-20B	1	3, 4, 5	1×10^{-2}
Qwen3.6-35B-A3B	0.01	3, 4, 5, 6	1×10^{-5}

by 18–21 points below the pre-edit model. Ablation on whitening types in Appendix D.2 shows that *in* regime performs the best.

Hyperparameter selection. `MoTE` adds only two hyperparameters on top of those of MoEEdit: the ridge coefficient λ and the whitening regularizer $\varepsilon_{\text{whitening}}$. Because there are only two of them, we select them by grid search over $\lambda \in \{10^{-3}, 10^{-2}, 10^{-1}, 1\}$ and $\varepsilon_{\text{whitening}} \in \{10^{-6}, \ldots, 10^{-1}\}$ on a held-out validation split of ZsRE, and transfer the selected values unchanged to COUNTERFACT. No tuning is done on any test split, and all experiments use the values of Table 5. Sensitivity to both hyperparameters is reported in Appendix D.4.

Shared expert. Qwen3.6-35B-A3B has 256 routed experts and one shared expert. We treat the shared expert as an ordinary expert with a fixed gate weight of 1. It is included in the expert mode of $\mathcal{W}$, so $E = 257$, receives its own key and its own null-space projector P_j, and enters the witness ϕ_t with $g_{t,\text{shared}} = 1$ outside the top-K renormalization (11). The routed gates therefore still sum to one among themselves, and the shared contribution is added on top, as in the forward pass. The shared expert is neither frozen nor absorbed into the residual: it is edited through the same core $\mathcal{G}$ and factor U_e as the routed experts, so the coupling between shared and routed experts is modelled. MoEEdit treats the shared expert in the same way, as an additional block of its BCD solve.

Target vectors. The T target-vector objectives of a batch are independent and share a frozen backbone. We pack all T requests into shared padded microbatches, so that one forward–backward Adam step updates many $v^\star$ jointly, while per-request losses, early stopping and clamping remain equivalent to the serial loop. With this batching, $v^\star$ costs about 0.1–0.3 s per fact.

Runtime and memory measurement. Total time and peak GPU memory in Table 2 are measured on a single H200 GPU over the complete run of 1000 facts (20 sequential batches of 50). They include the computation of $v^\star$, the activation covariances for the null-space projectors, the whitening statistics, the HOSVD of the expert tensor, the per-layer solves and the write-back.

Write-back. For backbones whose down-projection is stored in transposed packed layout (e.g. Qwen3's `down_proj` of shape $[E, d_{\text{hidden}}, d_{\text{model}}]$), the increment in (8) is transposed before assignment; for `ModuleList`-style experts, the assignment is direct into `experts[j].down_proj.weight`.

D Additional Experiments

D.1 Routing shift results

Following Gu et al. (2026), we conduct routing shift distribution analysis on each model and **COUNTERFACT** dataset. We perform 1000 edits with batch size 50 and measure routing similarity (RS) on both editing set and preserved 1000 edits set, averaging score over windows of 10 layers. Results in Table 6 show that our method outperforms all methods except MoEEdit. The most probable reason is Tucker Decomposition itself, because it projects inputs and outputs into core space, which shifts the distribution by design. Routing similarity is a proxy for preserving the backbone. Section 5.3 measures general capabilities directly and finds that `MoTE` loses no more MMLU accuracy than MoEEdit.

Table 6: Routing similarity before and after edit on editing and preservation sets. For Qwen model Early Middle and Late sites are layer 11-20, 21-30 and 31-40, for GPT-OSS: 5-9, 10-14 and 15-19, respectively. Best results are in **bold**, second best are underlined

Model	Method	Editing Set RS↑			Preservation Set RS↑		
		Early	Middle	Late	Early	Middle	Late
Qwen3-30B-A3B	FT	23.57	26.58	29.98	24.72	27.45	30.97
	FT-L	47.01	51.20	53.68	48.80	50.17	53.45
	AdaLoRA	16.63	24.11	27.00	16.38	23.84	26.60
	UnKE	52.46	44.12	44.80	49.90	41.91	43.84
	MoEEdit	**86.62**	**88.16**	**89.93**	**87.02**	**88.55**	**90.22**
	MoTE	59.82	69.11	73.88	67.41	69.83	66.46
GPT-OSS-20B	FT	31.75	28.09	35.25	32.94	28.10	34.98
	FT-L	60.51	57.01	64.95	61.30	57.11	64.12
	AdaLoRA	22.94	18.00	25.91	22.11	19.06	25.28
	UnKE	57.15	53.64	60.86	57.77	53.15	60.26
	MoEEdit	**77.37**	**75.56**	**80.83**	**78.00**	**76.13**	**80.53**
	MoTE	72.06	68.95	75.01	72.38	68.78	74.83
Qwen3.6-35B-A3B	FT	33.83	30.27	37.39	34.45	30.94	36.98
	FT-L	62.92	59.15	66.63	63.48	59.80	66.12
	AdaLoRA	24.08	20.49	27.77	24.61	21.06	27.28
	UnKE	59.34	55.02	62.81	59.89	55.55	62.26
	MoEEdit	**79.00**	**77.20**	**82.50**	**79.50**	**77.55**	**82.05**
	MoTE	73.52	70.45	76.55	74.02	70.48	76.03

D.2 Whitening ablation

Following results of Yuebin et al., we made an ablation on whitening type for each model. As for other experiments, we make 1000 edits from **COUNTERFACT** dataset with batch size 50 and other hyperparameters exactly matching those from Appendix C. Results in Table 7 show that using only *in* whitening show best performance and it's absence lead to huge model degradation. *Out* whitening in original implementation with random gradients doesn't yield any improvement and even brings slight decline in main metrics in combination with *in* whitening.

D.3 Null-space ablation

In this section we compare performance of MoTE method with enabled and disabled null-space projection. From results shown in Table 8 and evaluation protocol of **COUNTERFACT** B, we can conclude, that without null-space projection our method turn model into a random predictor which chooses between o_{true} and o_{new}. It shows the urgency of this component for preserving model overall performance.

D.4 Sensitivity to hyperparameters

Tables 9 and 10 report Utility on COUNTERFACT after 1000 edits, with all other hyperparameters fixed to the values of Table 5. The whitening regularizer is insensitive: across $\varepsilon_{\text{whitening}} \in [10^{-6}, 10^{-2}]$ Utility varies by at most 2.4 points, and only 10^{-1}, where the covariances are over-regularized, hurts. The ridge coefficient λ is architecture-specific. It balances memorization against the Frobenius penalty in (7), which depends on the activation scale, the number of experts and the top-K gate normalization of the backbone. This is a property of the objective rather than of our method: MoEEdit fixes $\lambda = 1$ for all three backbones, and Table 9 shows that this value is not uniformly good for MoEEdit either.

Table 7: Results of editing 1000 facts from **COUNTERFACT** dataset with different whitening types. Best results are in **bold**, second best are underlined

Model	Whitening type	COUNTERFACT			
		Eff.↑	Gen.↑	Spe.↑	Uti.↑
Qwen3-30B-A3B	None	63.50±0.48	61.15±0.42	66.45±0.31	63.70
	Out	66.50±0.47	61.95±0.42	**86.06**±0.31	71.50
	In	**97.00**±0.17	**82.60**±0.33	79.57±0.26	**86.39**
	Both	96.80±0.18	82.55±0.32	79.47±0.26	86.27
GPT-OSS-20B	None	41.70±0.49	37.10±0.42	65.50±0.33	48.10
	Out	45.87±0.50	36.80±0.41	66.32±0.33	49.66
	In	**94.60**±0.22	37.55±0.41	**81.85**±0.25	71.33
	Both	94.50±0.23	**38.30**±0.42	81.76±0.25	**71.52**
Qwen3.6-35B-A3B	None	82.50±0.38	73.00±0.37	64.50±0.32	73.33
	Out	82.10±0.38	**73.95**±0.37	65.07±0.30	73.71
	In	**89.20**±0.31	68.44±0.39	75.33±0.27	**77.66**
	Both	88.50±0.32	68.00±0.40	**75.52**±0.27	77.34

Table 8: Results of editing 1000 facts from **COUNTERFACT** dataset with MoTE method with *in* whitening and with or without null-space projection

Model	Null-space	COUNTERFACT			
		Eff.↑	Gen.↑	Spe.↑	Uti.↑
Qwen3-30B-A3B	✓	97.00±0.17	82.60±0.33	79.57±0.26	86.39
	×	49.60±0.50	49.0±0.36	48.91±0.21	49.17
GPT-OSS-20B	✓	94.60±0.22	37.55±0.41	81.85±0.25	71.33
	×	50.10±0.50	50.90±0.47	48.44±0.45	49.81
Qwen3.6-35B-A3B	✓	89.20±0.31	68.44±0.39	75.33±0.27	77.66
	×	49.90±0.52	51.1±0.48	50.3±0.55	50.43

D.5 Batch size ablation

Table 11 repeats the editing of 1000 COUNTERFACT facts on Qwen3-30B-A3B with different batch sizes T. Utility varies by at most 0.10 points for MoEEdit and 0.26 points for MoTE across $T \in \{10, 30, 50, 100, 200\}$, so the batch size does not affect the conclusions.

D.6 Long-horizon sequential editing

All results above are already sequential. To see what happens when the number of accumulated edits far exceeds 1000, we continue editing Qwen3-30B-A3B up to 10000 COUNTERFACT facts (Table 12). Both methods degrade as edits accumulate and reach a similar Utility at 10000 edits. MoTE keeps a much higher Specificity at long horizons (70.50 against 55.78 at 10000 edits), so it damages unrelated knowledge less, while its Efficacy and Generalization drop faster.

Table 9: Utility on COUNTERFACT after 1000 edits for different ridge coefficients λ. The best value for each model is in **bold**.

MoTE			
λ	Qwen3-30B-A3B	Qwen3.6-35B-A3B	GPT-OSS-20B
10^{-3}	57.46	74.80	64.62
10^{-2}	75.49	**75.33**	65.43
10^{-1}	**86.39**	48.90	60.86
1	46.31	35.80	**81.85**
MoEEdit			
λ	Qwen3-30B-A3B	Qwen3.6-35B-A3B	GPT-OSS-20B
10^{-3}	73.57	75.22	48.70
10^{-2}	85.06	**84.92**	50.98
10^{-1}	**89.39**	84.15	62.34
1	88.92	70.97	**69.73**

Table 10: Utility of MoTE on COUNTERFACT after 1000 edits for different whitening regularizers $\varepsilon_{\text{whitening}}$. The best value for each model is in **bold**.

$\varepsilon_{\text{whitening}}$	Qwen3-30B-A3B	Qwen3.6-35B-A3B	GPT-OSS-20B
10^{-6}	86.20	74.90	79.50
10^{-5}	**86.39**	**75.33**	79.80
10^{-4}	86.31	74.85	80.20
10^{-3}	86.25	74.70	80.90
10^{-2}	86.10	73.80	**81.85**
10^{-1}	81.20	72.50	78.40

Table 11: Editing 1000 facts from COUNTERFACT on Qwen3-30B-A3B with different batch sizes T.

T	MoEEdit				MoTE			
	Eff.↑	Gen.↑	Spe.↑	Uti.↑	Eff.↑	Gen.↑	Spe.↑	Uti.↑
10	98.80	82.85	85.33	88.99	95.80	76.45	83.35	85.20
30	98.70	82.80	85.26	88.92	95.80	76.45	83.43	85.23
50	98.60	82.85	85.30	88.92	95.50	76.80	83.34	85.21
100	98.60	83.05	85.42	**89.02**	96.10	76.90	83.38	**85.46**
200	98.50	82.90	85.49	88.96	95.50	77.05	83.06	85.20

Table 12: Long-horizon sequential editing of Qwen3-30B-A3B on COUNTERFACT.

# edits	MoEEdit				MoTE			
	Eff.↑	Gen.↑	Spe.↑	Uti.↑	Eff.↑	Gen.↑	Spe.↑	Uti.↑
1000	98.60	82.85	85.30	88.92	97.00	82.60	79.57	86.39
3000	92.33	79.60	80.22	84.05	91.40	76.20	81.31	82.97
5000	83.16	75.82	73.30	77.43	74.52	71.60	79.19	75.10
10000	54.62	53.56	55.78	54.65	46.24	48.50	70.50	53.75

D.7 SCALING WITH THE NUMBER OF EXPERTS

Within one backbone all experts are edited by construction: objective (3) is defined over the full pool $\{W_j\}_{j=1}^{E}$, and the Tucker reparameterization couples the experts through U_e. An ablation over the number of experts inside one model is therefore not well defined, and we compare across backbones with $E = 32$, 128 and 257. Table 13 isolates the components whose cost depends on E when editing 1000 COUNTERFACT facts with batch size 50: for MoEEdit, the full BCD optimization; for MoTE, the Tucker decomposition, the collection of whitening statistics and the Woodbury solve. The time of MoEEdit grows much faster with E. The residual growth of MoTE comes from forming the witnesses ϕ_t, which touch all E experts once, and from the HOSVD of a larger tensor. The inversion itself is $T \times T$ and does not grow with E.

Table 13: Time of the components whose cost depends on the number of experts E, editing 1000 facts from COUNTERFACT with batch size 50.

Model	E	MoTE, s ↓	MoEEdit, s ↓
GPT-OSS-20B	32	**36.68**	70.93
Qwen3-30B-A3B	128	**67.49**	231.65
Qwen3.6-35B-A3B	257	**109.41**	388.20

E DERIVATION OF THE UPDATE RULE

We index edited facts by $t = 1, \dots, T$ and keep the notation of Sections 3.2 and 4.

Step 1: the projector removes the preservation term. Let us define null-space projector as

$$P_j \;=\; I - U_j^{\geq\tau}\big(U_j^{\geq\tau}\big)^\top, \tag{13}$$

where $U_j^{\geq\tau}$ holds the eigenvectors of the preserved-key second moment $C_j = K_0^{(j)}\big(K_0^{(j)}\big)^\top$ with eigenvalues above a threshold τ, and $K_0^{(j)}$ stacks the keys $k_{f,j}$ produced by expert j on the preservation set.

Write every update as $\Delta_j = \widetilde{\Delta}_j P_j$ with P_j from (13). For a preserved prompt $f \in \mathcal{P}$ the key $k_{f,j}$ lies, up to the discarded eigen-directions of C_j, in the span of $U_j^{\geq\tau}$, so $P_j k_{f,j} \approx 0$. More precisely, $\sum_{f\in\mathcal{P}} \|P_j k_{f,j}\|^2 = \mathrm{tr}(P_j C_j)$ equals the sum of the eigenvalues of C_j below τ. The preservation term of (3) therefore vanishes, and with the projected keys $\widetilde{k}_{t,j} = P_j k_{t,j}$ the problem becomes

$$\min_{\{\widetilde{\Delta}_j\}} \sum_{t=1}^{T}\Big\|\sum_{j=1}^{E} g_{t,j}\,\widetilde{\Delta}_j\,\widetilde{k}_{t,j} - r_t\Big\|^2 + \lambda\sum_{j=1}^{E}\big\|\widetilde{\Delta}_j\big\|_F^2. \tag{14}$$

Only edit-set vectors appear in (14), which is why the closed forms of Section 4 contain no preservation keys.

Step 2: unstructured solution. Let $\psi_t = [g_{t,1}\widetilde{k}_{t,1}^\top \;\cdots\; g_{t,E}\widetilde{k}_{t,E}^\top]^\top$ and $\hat{\Delta} = [\widetilde{\Delta}_1 \;\cdots\; \widetilde{\Delta}_E]$. Then $\sum_j g_{t,j}\widetilde{\Delta}_j\widetilde{k}_{t,j} = \hat{\Delta}\psi_t$, and (14) is a ridge regression in $\hat{\Delta}$ with normal equations $\hat{\Delta}(\Psi\Psi^\top + \lambda I) = R\Psi^\top$. Hence $\hat{\Delta}^\star = R\Psi^\top(\Psi\Psi^\top + \lambda I)^{-1} = R(\Psi^\top\Psi + \lambda I_T)^{-1}\Psi^\top$, where the second equality is (1). This is (4).

Step 3: Tucker-structured solution. Substituting $\widetilde{\Delta}_j = U_{\text{out}} M_j U_{\text{in}}^\top$ with $M_j = \sum_a U_e[j,a]\,G_a$ from (6) gives

$$\sum_{j=1}^{E} g_{t,j}\,\widetilde{\Delta}_j\,\widetilde{k}_{t,j} = U_{\text{out}}\sum_{a=1}^{r_e} G_a \sum_{j=1}^{E} g_{t,j}\,U_e[j,a]\,c_{t,j} = U_{\text{out}}\sum_{a=1}^{r_e} G_a\,\phi_{t,a} = U_{\text{out}}\,G_{\text{flat}}\,\phi_t.$$

With *in* whitening only U_{in} is re-coloured, so U_{out} has orthonormal columns, and at full rank it is square. Then $\|U_{\text{out}} G_{\text{flat}} \phi_t - r_t\| = \|G_{\text{flat}} \phi_t - \widetilde{r}_t\|$ with $\widetilde{r}_t = U_{\text{out}}^\top r_t$. Replacing the penalty on $\widetilde{\Delta}_j$ by the penalty on the core yields (7). Its normal equations $G_{\text{flat}}(\Phi^\top \Phi + \lambda I) = \widetilde{R}^\top \Phi$ and (1) give $G^\star_{\text{flat}} = \widetilde{R}^\top (\Phi \Phi^\top + \lambda I_T)^{-1} \Phi$.

Step 4: write-back. The applied update is $\Delta_j = \widetilde{\Delta}^\star_j P_j$, which is (8). On the edited keys $\Delta_j k_{t,j} = \widetilde{\Delta}^\star_j \widetilde{k}_{t,j}$, which is exactly the quantity fitted in (14), while $\Delta_j K_0^{(j)} = \widetilde{\Delta}^\star_j P_j K_0^{(j)} \approx 0$ on the preserved keys. The projector thus appears both in the witnesses and in the write-back, and the preservation constraint holds by construction. Spreading the core over the edited layers with (9) repeats the write-back with the factors and projectors of each layer.