ENGRAMEDIT: DECOUPLED KNOWLEDGE UPDATES IN LLMS THROUGH CONDITIONAL MEMORY

Anonymous authors
Paper under double-blind review

ABSTRACT

Conditional memory architectures such as DeepSeek Engram use input n-grams to look up learned embeddings, expanding the capacity of large language models (LLMs) with limited additional computation. Beyond model scaling, this architecture has demonstrated the potential to decouple factual knowledge storage from general-purpose computation, offering a promising route to updating factual knowledge while keeping the Transformer backbone fixed. However, updating shared n-gram embeddings can affect other edits and unrelated knowledge, while different expressions of the same fact may activate embeddings that were not updated. We propose **EngramEdit** for decoupled knowledge updates through conditional memory. EngramEdit first computes target memory representations that make the model predict the updated fact across multiple expressions. It then jointly updates the shared n-gram embeddings to match these targets across expressions and edits, penalizing updates to frequently reused embeddings more strongly to preserve unrelated knowledge. Experiments show that EngramEdit enables independent factual knowledge updates through conditional memory, achieving near-perfect editing success. Revised knowledge is usable across unseen expressions and in multi-hop reasoning, with nearly $3\times$ the strongest baseline's accuracy under chain-of-thought (CoT) prompting. Unrelated knowledge and general capabilities are largely preserved even as factual updates accumulate. These findings show that EngramEdit turns conditional memory into an editable knowledge interface, extending its role beyond model scaling to support decoupled knowledge updates. The code is available at `https://anonymous.4open.science/r/EngramEdit`.

1 INTRODUCTION

Sparsity in large language models (LLMs) provides a promising way to scale model capacity without proportionally increasing per-token computation. Mixture-of-Experts (MoE) (Shazeer et al., 2017; Lepikhin et al., 2021) achieves this through *conditional computation*, activating only a subset of expert networks for each token. More recently, DeepSeek Engram (Cheng et al., 2026b) explores a complementary approach through *conditional memory*. At each token position, the model uses n-grams of different lengths ending at that position to look up their learned embeddings. These embeddings provide input-dependent memory representations for the Transformer backbone computation (see Figure 1(a)). This design is increasingly used to expand the capacity of LLMs such as DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) and Qwen3.8-Flash-Next (Qiu et al., 2026).

Beyond model scaling, conditional memory offers a promising architectural basis for decoupling factual knowledge storage from general-purpose computation in LLMs. In DeepSeek Engram, n-gram embeddings store reusable local representations, while the Transformer backbone dynamically integrates and processes these representations in context (Cheng et al., 2026b). Ablation experiments on Engram further show that disabling conditional memory severely degrades factual knowledge performance while largely preserving the ability to understand and reason about information provided in context (see Figure 1(b)).

This division of roles suggests a promising possibility: independently update factual knowledge in conditional memory without modifying the general-purpose backbone. Such updates would turn conditional memory into an editable knowledge interface, allowing LLMs to revise outdated facts

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.

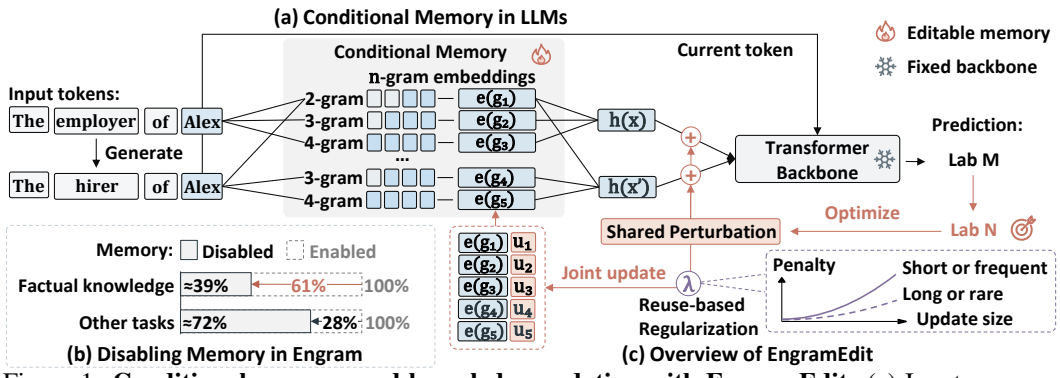


Figure 1: **Conditional memory and knowledge updating with EngramEdit**. (a) Input n-grams look up learned embeddings for backbone computation. (b) Disabling memory in DeepSeek Engram causes larger relative performance drops on factual-knowledge benchmarks (Cheng et al., 2026b). (c) EngramEdit computes target memory representations and jointly updates the corresponding n-gram embeddings, with stronger penalties on frequently reused embeddings.

while largely preserving general capabilities. Prior work mainly studies model scaling (Cheng et al., 2026b; Liu et al., 2026), personalization (Li, 2026), and domain adaptation (Hou & Wang, 2026), leaving this potential for decoupled knowledge updates largely unexplored.

However, updating factual knowledge through conditional memory is non-trivial because memory lookup is determined by n-grams rather than facts. 1) ***Efficacy.*** Different facts being edited may require conflicting changes to a shared n-gram embedding, so an update that makes one edit succeed may cause another to fail. 2) ***Generalization.*** Rephrasing the same fact may activate different n-grams whose embeddings were not updated, causing the edit to fail on the new expression. 3) ***Specificity.*** Embeddings associated with shorter or more frequently used n-grams are also activated by many unrelated inputs, so updating them may change unrelated factual predictions.

To enable decoupled knowledge updates through conditional memory, we propose **EngramEdit**. The key idea is to first compute memory representations that make the LLM predict the updated facts, and then jointly translate these target representations into updates to the underlying shared n-gram embeddings. As shown in Figure 1(c), we first generate multiple expressions of each fact to broaden n-gram coverage. For each edit, a shared learnable perturbation is temporarily added to the memory representations of its expressions. Optimizing only this perturbation by backpropagating the prediction loss for the new fact produces a target memory representation for each expression (see Sec. 3.1). We then map each target to the n-gram embeddings used by its expression, identifying embeddings shared across expressions and edits (see Sec. 3.2). The embedding updates are solved jointly to match these targets, with each shared embedding receiving one update that accounts for all expressions and edits using it. To preserve unrelated knowledge, we include stronger penalties on large updates to frequently reused embeddings in this joint optimization (see Sec. 3.3).

Our experiments show that EngramEdit enables effective and generalizable factual knowledge updates through conditional memory while largely preserving unrelated knowledge and general capabilities. 1) **Factual knowledge can be effectively updated.** By updating only conditional memory, EngramEdit achieves near-perfect editing success on both CounterFact (Meng et al., 2022) and ZsRE (Levy et al., 2017). 2) **Updated knowledge remains usable.** EngramEdit achieves strong generalization to unseen expressions, and the revised facts support multi-hop reasoning. On MQuAKE (Zhong et al., 2023), it achieves nearly 3$\times$ the strongest baseline's multi-hop accuracy under chain-of-thought (CoT) prompting. 3) **Unrelated knowledge and general capabilities are largely preserved.** Overall performance on unrelated factual queries and general-ability tasks remains largely unchanged during extended sequential editing. Together, these findings show that EngramEdit turns conditional memory into an editable knowledge interface for decoupled knowledge updates. This extends its role beyond model scaling and opens a promising path toward LLMs that continually update their knowledge while largely preserving their general capabilities.

2 Preliminaries

This section introduces factual knowledge editing (KE) and the structure of conditional memory.

Factual Knowledge Editing. KE aims to update specific facts in a pretrained LLM $\mathcal{M}$ while preserving unrelated knowledge (Meng et al., 2022; 2023). Each edit request $e_i = (s_i, r_i, o_i \to o_i^\star)$ replaces the original object o_i with the desired object $o_i^\star$ for subject s_i and relation r_i, e.g., changing Alex's employer from Lab M to Lab N. Given an edit prompt x_i expressing (s_i, r_i), the edited model should predict the desired object $o_i^\star$. In sequential editing, we process batches $\mathcal{E} = \{e_i\}_{i=1}^m$ of m requests, applying each batch to the model updated by preceding batches.

EngramEdit shares a basic idea with locate-then-edit methods (Meng et al., 2022; 2023; Fang et al., 2025; Gu et al., 2026): first find a representation that makes the model predict the desired object, then update the parameters that produce it. We introduce this process through feed-forward network (FFN) editing. The editor first selects an FFN layer ℓ and a subject-token position q_i associated with factual recall, typically the last subject token (Meng et al., 2022). At this location, the FFN output projection $\mathbf{W}_{\text{out}}^\ell$ maps its input activation $\mathbf{k}_i^\ell$ to the current output $\mathbf{v}_i^\ell = \mathbf{W}_{\text{out}}^\ell \mathbf{k}_i^\ell$. The editor temporarily adds a learnable perturbation $\boldsymbol{\delta}_i$ to this output and optimizes it by backpropagating the prediction loss for $o_i^\star$, with all model parameters fixed. The optimized perturbation $\boldsymbol{\delta}_i^\star$ gives the target output

$$\mathbf{v}_i^\star = \mathbf{v}_i^\ell + \boldsymbol{\delta}_i^\star. \tag{1}$$

The editor then solves for and applies a parameter update $\Delta\mathbf{W}_{\text{out}}^\ell$ so that the FFN produces this target without the temporary perturbation (see App. A.1 for the full formulation):

$$(\mathbf{W}_{\text{out}}^\ell + \Delta\mathbf{W}_{\text{out}}^\ell)\mathbf{k}_i^\ell \approx \mathbf{v}_i^\star. \tag{2}$$

Conditional Memory via n-gram Embeddings. Conditional memory associates each n-gram g with a learned embedding $\mathbf{e}(g)$ used when g is activated (Cheng et al., 2026b; Liu et al., 2026; Qiu et al., 2026). We treat each embedding as one logical vector, abstracting the finite hashed tables and sub-tables used to control storage costs in practical architectures (Cheng et al., 2026b; Liu et al., 2026). The same n-gram uses the same embedding across inputs.

Given an input token sequence $x = (t_1, \ldots, t_T)$, let $g_{q,n}(x) = t_{q-n+1:q}$ denote the length-n suffix n-gram ending at position q. For the configured length set $\mathcal{N}$, the activated n-gram set is $\mathcal{G}(x, q) = \{g_{q,n}(x) : n \in \mathcal{N},\, n \le q\}$. We write the contribution of the activated embeddings to the backbone computation as a memory representation $\mathbf{h}(x, q)$:

$$\mathbf{h}(x, q) = \text{Agg}_{x,q}\left(\{\mathbf{e}(g) : g \in \mathcal{G}(x, q)\}\right), \tag{3}$$

where $\text{Agg}_{x,q}$ includes the architecture-specific processing of the activated embeddings and any interaction with the backbone state at position q. For example, additive aggregation combines the embeddings directly (Liu et al., 2026), whereas context-aware gated fusion uses the backbone state to control their contribution (Cheng et al., 2026b).

3 METHOD

In this section, we present EngramEdit for decoupled knowledge updates through conditional memory (see Figure 2). For each edit batch, we first compute target memory representations that make the LLM predict the updated facts across expressions (see Section 3.1). We then map each expression to its activated n-grams, identifying those shared across expressions and edits (see Section 3.2). Finally, we jointly solve for a single update per embedding to match the targets using reuse-based regularization (see Section 3.3). Algorithm 1 in App. A.4 summarizes the complete procedure.

3.1 CONDITIONAL MEMORY TARGET COMPUTATION

EngramEdit first learns how memory representations should change to predict the updated facts, providing targets for jointly allocating updates across shared n-gram embeddings.

Expression Generation. Different expressions of the same edit may activate different n-grams and use different embeddings. To cover more of these n-grams, we ask the model $\mathcal{M}$ to generate K semantically equivalent expressions of the subject–relation request for each edit e_i in the batch $\mathcal{E}$ before editing. The generation prompt encourages varied wording immediately before the subject (see App. A.2.1 for detailed prompts). Combining the generated expression set $\mathcal{P}_i = \{\widetilde{x}_i^{(k)}\}_{k=1}^K$ with the original edit prompt x_i gives $\mathcal{X}_i = \{x_i\} \cup \mathcal{P}_i$, used for target computation and memory mapping in Section 3.2.

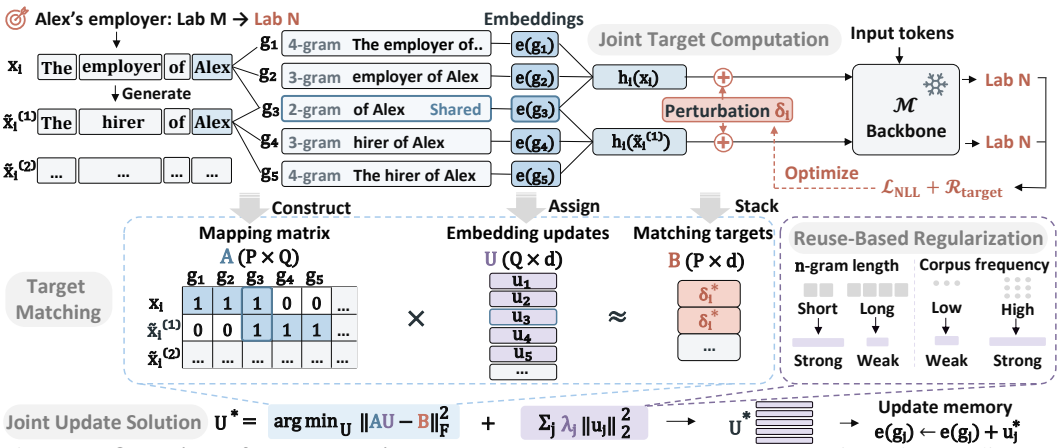


Figure 2: **Overview of EngramEdit**. We compute target memory representations that predict updated facts across expressions, map expressions to their activated n-grams, and jointly update the embeddings to match these targets with stronger penalties for shorter or more frequent n-grams.

Joint Target Computation. We next compute target memory representations that make the LLM predict the updated fact across expressions. Since these expressions describe the same factual update, a shared perturbation allows them to jointly guide a single adjustment.

For each expression $x \in \mathcal{X}_i$, let $q_i(x)$ denote its last subject-token position (Meng et al., 2022) and $\mathbf{h}_i(x) = \mathbf{h}(x, q_i(x)) \in \mathbb{R}^d$ its memory representation there. Following Meng et al. (2022; 2023), we temporarily add a shared learnable perturbation $\boldsymbol{\delta}_i \in \mathbb{R}^d$ to each representation (see Figure 2). Let $\mathcal{M}_{\boldsymbol{\delta}_i}$ denote the model evaluated with this perturbation. With all model parameters fixed, we optimize only $\boldsymbol{\delta}_i$ using the prediction loss for the updated fact averaged across expressions:

$$\boldsymbol{\delta}_i^\star = \arg\min_{\boldsymbol{\delta}_i} \frac{1}{|\mathcal{X}_i|} \sum_{x \in \mathcal{X}_i} \mathcal{L}_{\mathrm{NLL}}\big(\mathcal{M}_{\boldsymbol{\delta}_i}(x), o_i^\star\big) + \mathcal{R}_{\mathrm{target}}(\boldsymbol{\delta}_i), \\ \text{subject to } \|\boldsymbol{\delta}_i\|_2 \le \rho \|\mathbf{h}_i(x_i)\|_2. \tag{4}$$

Here, $\mathcal{L}_{\mathrm{NLL}}$ is the negative log-likelihood (NLL) loss for the updated fact. $\mathcal{R}_{\mathrm{target}}$ penalizes large perturbations and prediction changes when describing the subject (Meng et al., 2022), and the clamp factor $\rho > 0$ bounds the perturbation norm (see App. A.2.2 for details). The optimized perturbation $\boldsymbol{\delta}_i^\star$ defines a target memory representation for each expression:

$$\mathbf{h}_i^\star(x) = \mathbf{h}_i(x) + \boldsymbol{\delta}_i^\star, \qquad x \in \mathcal{X}_i. \tag{5}$$

3.2 Multi-Expression Memory Mapping

EngramEdit next links each target memory representation to the n-gram embeddings activated by its expression. Because expressions and edits may share embeddings, we construct a single mapping for the entire edit batch.

Memory Mapping. To link each target memory representation to its underlying embeddings, we identify the n-grams activated at the expression's last subject-token position. For each expression $x \in \mathcal{X}_i$, the activated n-gram set is $\mathcal{G}_i(x) = \mathcal{G}(x, q_i(x))$, using the lookup notation from Section 2. The union of these sets, $\mathcal{G}_i = \bigcup_{x \in \mathcal{X}_i} \mathcal{G}_i(x)$, collects the activated n-grams for edit e_i. Expressions activating the same n-gram g share its embedding $\mathbf{e}(g)$, as illustrated by "of Alex" (see Figure 2). When combining the n-gram sets $\mathcal{G}_i$ across the batch, we therefore retain each n-gram only once.

Mapping Matrix. To jointly allocate updates to shared embeddings, we record which expressions use each embedding in a batch-level mapping matrix $\mathbf{A} \in \mathbb{R}^{P \times Q}$. Each of the $P = \sum_{i=1}^m |\mathcal{X}_i|$ rows corresponds to one expression and its target memory representation. The Q columns correspond to the distinct n-grams $\{g_j\}_{j=1}^Q$ in $\bigcup_{i=1}^m \mathcal{G}_i$, with column j representing the embedding $\mathbf{e}(g_j)$. Row p is indexed by an edit–expression pair (i_p, x_p), where the expression $x_p \in \mathcal{X}_{i_p}$. The matrix element A_{pj} is 1 if expression x_p activates n-gram g_j, and 0 otherwise:

$$A_{pj} = \mathbb{I}\big[g_j \in \mathcal{G}_{i_p}(x_p)\big]. \tag{6}$$

3.3 JOINT MEMORY UPDATE ALLOCATION

Using the expression-to-n-gram mapping, EngramEdit jointly computes embedding updates to match targets while penalizing large updates to frequently reused embeddings.

Target Matching. To match each target memory representation, we express its required change in terms of updates to the activated embeddings. We derive this relation under sum aggregation, where a memory representation changes by the sum of its embedding updates (Liu et al., 2026). For column j of the mapping matrix $\mathbf{A}$, let $\mathbf{u}_j \in \mathbb{R}^d$ denote the update to the n-gram embedding $\mathbf{e}(g_j)$.

For row p of the mapping matrix $\mathbf{A}$, let $\mathbf{h}_p = \mathbf{h}_{i_p}(x_p)$ denote the current memory representation and $\mathbf{h}^\star_p = \mathbf{h}^\star_{i_p}(x_p)$ its target. To match this target, the combined embedding updates must approximate the difference between the two representations:

$$\sum_{j=1}^{Q} A_{pj}\mathbf{u}_j \approx \mathbf{h}^\star_p - \mathbf{h}_p = \boldsymbol{\delta}^\star_{i_p}. \tag{7}$$

We denote this difference by the matching target $\mathbf{b}_p = \mathbf{h}^\star_p - \mathbf{h}_p$. Stacking the embedding updates as rows gives the embedding update matrix $\mathbf{U} = [\mathbf{u}_1, \ldots, \mathbf{u}_Q]^\top \in \mathbb{R}^{Q\times d}$; stacking the matching targets gives the matching target matrix $\mathbf{B} \in \mathbb{R}^{P\times d}$. The batch-level matching requirement is therefore $\mathbf{AU} \approx \mathbf{B}$ (see Figure 2). App. A.3.2 extends this formulation to context-aware gated aggregation used in Engram (Cheng et al., 2026b).

Reuse-Based Regularization. To preserve unrelated knowledge, we penalize large updates more strongly for embeddings likely to be activated by unrelated inputs. We use n-gram length and corpus frequency as indicators of activation frequency across inputs, assigning larger regularization weights w_j to shorter or more frequent n-grams (see App. A.3.3 for the construction of the weights). The regularization matrix is

$$\mathbf{\Lambda} = \mathrm{diag}(\lambda_1, \ldots, \lambda_Q), \qquad \lambda_j = \lambda_{\text{reuse}} w_j + \lambda_{\text{ridge}}, \tag{8}$$

where λ_{reuse} controls the reuse-based penalty, while the optional ridge term λ_{ridge} independently sets a uniform penalty on all updates. Both coefficients are nonnegative, with $\lambda_j > 0$ for every embedding. We prove that, under sum aggregation, weighting update penalties by activation probability bounds the average squared change in memory representations on unrelated inputs, providing a theoretical basis for reuse-based regularization (see App. A.5.2 for the proof and bounds on individual embedding updates).

Joint Update Solution. We combine target matching and reuse-based regularization to solve for the embedding updates:

$$\mathbf{U}^\star = \arg\min_{\mathbf{U}} \|\mathbf{AU} - \mathbf{B}\|_F^2 + \sum_{j=1}^{Q} \lambda_j \|\mathbf{u}_j\|_2^2 . \tag{9}$$

By jointly considering all expressions that share an embedding, this objective balances their potentially conflicting targets (see App. A.5.1 for exact-matching conditions). With all $\lambda_j > 0$, $\mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda}$ is positive definite, yielding a unique update computed by solving the corresponding linear system (see App. A.5.3 for the proof):

$$\mathbf{U}^\star = \left(\mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda}\right)^{-1} \mathbf{A}^\top\mathbf{B}. \tag{10}$$

We prove that the joint solution has two useful properties. 1) **Minimum update cost.** Different embedding updates can produce the same memory representation changes but incur different update costs. EngramEdit minimizes the weighted embedding update cost among all updates producing the same achieved memory representation changes $\mathbf{AU}^\star$ (see App. A.5.4 for the proof). 2) **Stability to target changes.** Since targets are computed through optimization, they may vary slightly. With the mapping and regularization fixed, we prove that regularization bounds the extent to which these variations affect the solved embedding updates (see App. A.5.5 for the analysis and proof).

Memory Updating. Finally, we update conditional memory as $\mathbf{e}(g_j) \leftarrow \mathbf{e}(g_j) + \mathbf{u}^\star_j$, keeping all other parameters fixed. During inference, an embedding update enters the model's computation only when its n-gram is active in the current prefix.

Table 1: **Editing results on CounterFact and ZsRE**. MFT-S and MFT-A denote Memory-FT (Subject) and (All Tokens). Eff. = Efficacy, Gen. = Generalization, Spec. = Specificity, Util. = Utility, Flu. = Fluency, Cons. = Consistency. Subscripts give 95% confidence-interval half-widths. Bold and underlining mark the best and second-best editing scores.

Method	CounterFact						ZsRE			
	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑	Flu. ↑	Cons. ↑	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑
Pre-edited	$9.5_{\pm 1.28}$	$11.2_{\pm 1.19}$	$86.8_{\pm 0.94}$	35.8	$544.0_{\pm 1.67}$	$15.0_{\pm 0.37}$	$36.9_{\pm 1.30}$	$36.5_{\pm 1.29}$	$41.6_{\pm 1.19}$	38.3
FT	$94.9_{\pm 0.96}$	$63.6_{\pm 1.83}$	$43.8_{\pm 1.63}$	67.4	$505.1_{\pm 2.04}$	$13.3_{\pm 0.36}$	$76.7_{\pm 1.13}$	$76.8_{\pm 1.15}$	$43.1_{\pm 1.27}$	65.5
FT-L	$15.3_{\pm 1.58}$	$12.5_{\pm 1.24}$	$\underline{83.3}_{\pm 1.11}$	37.0	$542.0_{\pm 1.70}$	$14.7_{\pm 0.36}$	$58.0_{\pm 1.47}$	$54.4_{\pm 1.48}$	$45.8_{\pm 1.24}$	52.7
AdaLoRA	$61.1_{\pm 2.13}$	$27.5_{\pm 1.55}$	$51.1_{\pm 1.35}$	46.6	$544.3_{\pm 1.57}$	$12.9_{\pm 0.35}$	$70.9_{\pm 1.40}$	$64.7_{\pm 1.46}$	$\underline{46.8}_{\pm 1.27}$	60.8
UnKE	$84.5_{\pm 1.59}$	$58.0_{\pm 1.88}$	$47.5_{\pm 1.61}$	63.3	$546.5_{\pm 1.64}$	$15.0_{\pm 0.35}$	$\underline{86.9}_{\pm 1.13}$	$\underline{79.0}_{\pm 1.33}$	$\mathbf{49.9}_{\pm 1.32}$	$\underline{71.9}$
MoEEdit	$\underline{99.1}_{\pm 0.41}$	$\underline{67.1}_{\pm 1.69}$	$63.5_{\pm 1.28}$	76.6	$544.7_{\pm 1.65}$	$14.7_{\pm 0.33}$	$85.1_{\pm 0.91}$	$78.7_{\pm 1.21}$	$44.4_{\pm 1.26}$	69.4
MFT-S	$99.0_{\pm 0.45}$	$59.1_{\pm 1.92}$	$\mathbf{85.2}_{\pm 0.90}$	$\underline{81.1}$	$\underline{555.2}_{\pm 1.38}$	$\underline{15.3}_{\pm 0.36}$	$66.9_{\pm 1.35}$	$56.5_{\pm 1.53}$	$36.9_{\pm 1.17}$	53.4
MFT-A	$93.9_{\pm 1.04}$	$40.7_{\pm 1.75}$	$66.6_{\pm 1.12}$	67.1	$496.7_{\pm 4.04}$	$11.5_{\pm 0.40}$	$62.8_{\pm 1.49}$	$51.3_{\pm 1.57}$	$33.7_{\pm 1.13}$	49.3
EngramEdit	$\mathbf{99.5}_{\pm 0.31}$	$\mathbf{97.0}_{\pm 0.69}$	$\mathbf{85.2}_{\pm 0.92}$	**93.9**	$\mathbf{565.2}_{\pm 1.10}$	$\mathbf{16.1}_{\pm 0.36}$	$\mathbf{97.3}_{\pm 0.40}$	$\mathbf{93.7}_{\pm 0.79}$	$38.3_{\pm 1.18}$	**76.4**

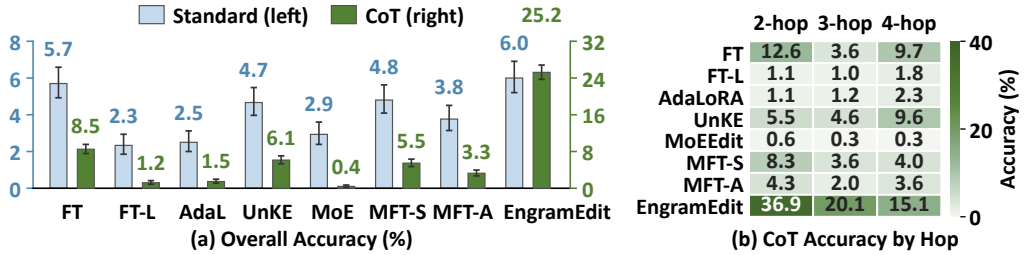


Figure 3: **Editing results on MQuAKE**. (a) Standard (left) and CoT (right) prompting with 95% confidence intervals. (b) CoT accuracy by hop. AdaL and MoE denote AdaLoRA and MoEEdit.

4 Experiments

In this section, we conduct experiments to address the following research questions:

- **RQ1:** How effectively does EngramEdit update factual knowledge through conditional memory while preserving unrelated knowledge?
- **RQ2:** How well does EngramEdit preserve general capabilities during knowledge updates?
- **RQ3:** How do EngramEdit's design choices affect editing performance?
- **RQ4:** How do EngramEdit's memory updates support the LLM's use of revised knowledge?

4.1 Experimental Setup

Datasets and Metrics. On CounterFact (Meng et al., 2022) and ZsRE (Levy et al., 2017), we report Efficacy (success on edited prompts), Generalization (success on held-out paraphrases not used during editing), Specificity (performance on unrelated queries), and their arithmetic mean, Utility. On CounterFact, we also report Fluency and Consistency for generated text. MQuAKE (Zhong et al., 2023) measures multi-hop use of updated facts through answer accuracy. Following AlphaEdit (Fang et al., 2025), we report mean task-level F1 across six general-ability tasks. See Apps. B.1 and B.2 for dataset examples and metric definitions.

Baselines and Implementation. We compare editors applicable to MoE LLMs: FT, FT-L (Zhu et al., 2020), AdaLoRA (Zhang et al., 2023), UnKE (Deng et al., 2025), and MoEEdit (Gu et al., 2026). To compare direct memory fine-tuning, we include Memory-FT (Subject) and Memory-FT (All Tokens), which update n-gram embeddings activated at the last subject token or all token positions in the original edit prompts, respectively. All experiments use LongCat-Flash-Lite (Liu et al., 2026), with sequential batch editing. See Apps. B.3 and B.4 for details.

4.2 Knowledge Update and Preservation (RQ1)

To evaluate how effectively EngramEdit updates facts through conditional memory, we compare it with the baselines after 2,000 sequential edits on each of CounterFact and ZsRE (see Table 1). We

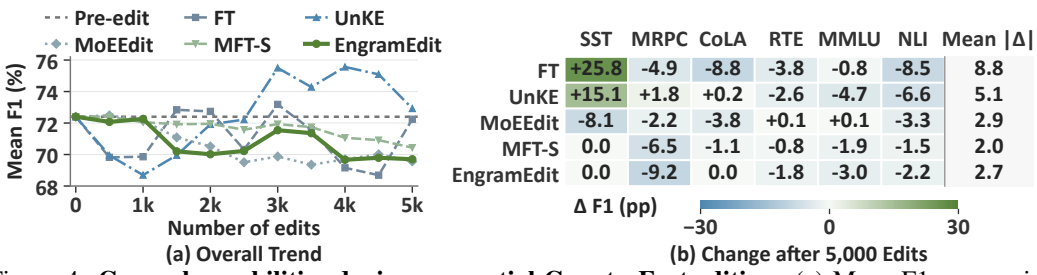


Figure 4: **General capabilities during sequential CounterFact editing**. (a) Mean F1 across six tasks in the editing sequence. (b) Task-level F1 changes from pre-edit scores after 5,000 edits, in percentage points (pp). The mean $|\Delta|$ averages the absolute task-level changes.

further apply edits from 3,000 MQuAKE cases to test whether the updated facts support multi-hop reasoning (see Figure 3). We apply standard prompting for direct answers and CoT prompting for step-by-step reasoning before answering. We observe that:

- On CounterFact, EngramEdit achieves similar Efficacy to MoEEdit and MFT-S but much higher Generalization; on ZsRE, it leads in both metrics (see Table 1). Comparable success on edited prompts, therefore, does not imply comparable recall through other expressions. EngramEdit addresses this gap by updating embeddings across expressions, since different wordings can activate different n-grams. Its advantages persist through 5,000 sequential edits, showing that revised facts remain accessible as further updates accumulate (see App. C.1.1 for details).
- On MQuAKE, EngramEdit, and FT have similar accuracy under standard prompting. With CoT, EngramEdit achieves nearly $3\times$ the accuracy of the strongest baseline and leads across all hop groups, as shown in Figure 3. This improvement shows that facts updated through conditional memory can be used in multi-hop reasoning. During CoT reasoning, intermediate steps can activate updated embeddings for the facts used, allowing revised information to guide subsequent steps. The advantage extends through four-hop cases, supporting the use of updated knowledge in longer reasoning chains (see App. C.1.2 for full results).
- On ZsRE, EngramEdit's Specificity remains close to the pre-edit level, whereas FT-L, AdaLoRA, UnKE, and MoEEdit exceed it. These higher scores do not by themselves establish better preservation, because Specificity counts newly correct answers as well as retained correct answers. The correctness transition analysis shows that corrections to previously wrong, unrelated predictions contribute to the baselines' gains and that EngramEdit retains most initially correct predictions (see App. C.1.3 for further analysis). This provides additional evidence that its knowledge updates largely preserve performance on unrelated queries.
- EngramEdit achieves the highest Utility on CounterFact and ZsRE, with Specificity close to pre-edit levels, and also leads in Fluency and Consistency. Together, these results show that EngramEdit supports effective and generalizable knowledge updates through conditional memory while largely preserving unrelated knowledge.

4.3 General Capabilities (RQ2)

To assess whether EngramEdit preserves general capabilities as knowledge updates accumulate, we evaluate six general-ability tasks in 5,000 sequential CounterFact edits. We compare mean task-level F1 and per-task changes against pre-edit performance. The results in Figure 4 show that:

- EngramEdit retains over 96% of its pre-edit mean F1 after 5,000 sequential edits, with modest changes throughout the editing sequence, as shown in Figure 4(a). MFT-S also largely retains pre-edit mean F1, but EngramEdit achieves much stronger editing Generalization (see Section 4.2). Both methods restrict editing to selected n-gram embeddings, and EngramEdit further penalizes large changes to frequently reused embeddings to limit effects on unrelated tasks. Together, these results support conditional memory as an editable knowledge interface and show that EngramEdit combines stronger cross-expression recall with largely preserved general capabilities.
- EngramEdit shows small F1 changes on most tasks, with a larger decline on MRPC (see Figure 4(b) and full trajectories in App. C.1.4). MRPC tests whether two sentences express the same meaning. These sentences can activate different n-grams, so local embedding updates may affect their representations differently, altering the equivalence judgment.

Table 2: **Ablations** of joint update allocation (Joint), reuse-based regularization (Reg.), n-gram corpus frequency (Freq.) and length (Len.) weighting, and generated expressions (Expr.).

Variant	CounterFact						ZsRE			
	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑	Flu. ↑	Cons. ↑	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑
EngramEdit	**99.5**$_{\pm 0.31}$	**97.0**$_{\pm 0.69}$	**85.2**$_{\pm 0.92}$	**93.9**	565.2$_{\pm 1.10}$	16.1$_{\pm 0.36}$	**97.3**$_{\pm 0.40}$	93.7$_{\pm 0.79}$	**38.3**$_{\pm 1.18}$	**76.4**
w/o Joint	74.5$_{\pm 1.91}$	72.8$_{\pm 1.74}$	84.8$_{\pm 0.93}$	77.4	**568.0**$_{\pm 1.00}$	15.3$_{\pm 0.35}$	57.4$_{\pm 1.55}$	58.3$_{\pm 1.57}$	35.6$_{\pm 1.17}$	50.4
w/o Freq.	99.4$_{\pm 0.32}$	96.8$_{\pm 0.69}$	**85.2**$_{\pm 0.92}$	93.8	564.9$_{\pm 1.22}$	16.1$_{\pm 0.36}$	96.0$_{\pm 0.52}$	92.6$_{\pm 0.84}$	37.3$_{\pm 1.19}$	75.3
w/o Len.	99.3$_{\pm 0.38}$	96.8$_{\pm 0.70}$	85.1$_{\pm 0.93}$	93.7	565.4$_{\pm 1.09}$	**16.2**$_{\pm 0.36}$	95.7$_{\pm 0.55}$	93.4$_{\pm 0.75}$	37.2$_{\pm 1.17}$	75.4
w/o Reg.	98.8$_{\pm 0.47}$	96.8$_{\pm 0.72}$	84.6$_{\pm 0.93}$	93.4	**568.0**$_{\pm 1.02}$	15.5$_{\pm 0.35}$	96.0$_{\pm 0.54}$	**94.0**$_{\pm 0.75}$	37.2$_{\pm 1.19}$	75.7
w/o Reg. & Expr.	98.8$_{\pm 0.47}$	81.5$_{\pm 1.59}$	**85.2**$_{\pm 0.92}$	88.5	565.9$_{\pm 1.08}$	15.6$_{\pm 0.36}$	93.9$_{\pm 0.72}$	85.4$_{\pm 1.19}$	37.0$_{\pm 1.17}$	72.1

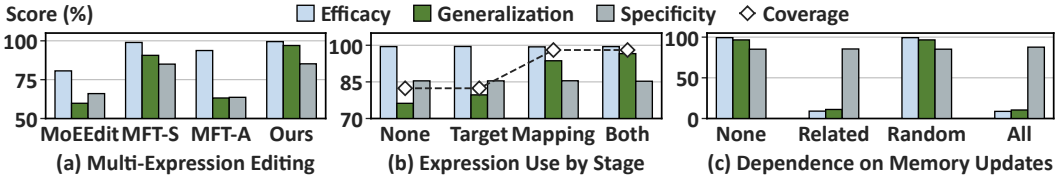


Figure 5: **Generated expressions and memory use on CounterFact**. (a) Editors use the same generated expressions per fact. (b) Generated expressions are used for target computation, memory mapping, both, or neither. Coverage is the percentage of held-out paraphrases activating fact-related updated embeddings. (c) Disabling no updates (None), fact-related updates (Related), matched random updates (Random), or all updates (All) during inference.

4.4 In-Depth Analysis

Ablation Study (RQ3). To isolate each component's contribution, we compare five variants on CounterFact and ZsRE, with other settings held fixed. *w/o Joint* solves embedding updates independently for each expression. *w/o Freq.* and *w/o Len.* remove the weighting that increases penalties for more frequent and shorter n-grams, respectively. *w/o Reg.* removes reuse-based regularization, and *w/o Reg. & Expr.* additionally removes generated expressions. From Table 2, we observe that: 1) Removing joint allocation substantially lowers Efficacy and Generalization on both datasets. This shows that jointly considering the targets of all expressions sharing an embedding helps the model recall revised facts across expressions. 2) Removing generated expressions from the unregularized variant substantially lowers Generalization. Generated expressions, therefore, mainly support cross-expression recall by covering the different n-grams activated by alternative wordings. 3) Removing reuse-based regularization lowers Efficacy and Specificity, and removing either frequency or length weighting also reduces Utility. The regularization thus helps preserve unrelated knowledge by limiting changes to embeddings reused by other inputs. Our analysis further shows that over 92% of new errors on unrelated queries involve activation of updated embeddings (see App. C.3.2).

Analysis of Generated Expressions (RQ3). To distinguish the benefit of additional expressions from how they are used, we first compare MoEEdit, MFT-S, MFT-A, and EngramEdit with the same generated expressions per fact on CounterFact (see Figure 5(a)). We then vary expression use in EngramEdit's target computation and memory mapping, keeping other settings fixed (see Figure 5(b)). All variants retain the original edit prompt. Coverage is the percentage of held-out paraphrases that activate an embedding updated for the same fact. We observe that: 1) With the same generated expressions, EngramEdit outperforms MFT-S in Generalization at similar Efficacy and Specificity. EngramEdit's target computation and joint memory updating thus use these expressions more effectively for recall under different wordings (see App. C.2.1 for full results). 2) Using generated expressions for memory mapping increases Coverage and yields most of the Generalization gain, because the expanded set of updated n-grams allows more held-out paraphrases to activate fact-related embeddings. An additional analysis of memory activation supports this explanation: the roughly 2% of held-out paraphrases activating no fact-related embeddings account for half of generalization failures (see App. C.3.2). With Coverage unchanged, also using these expressions for target computation further improves Generalization by optimizing targets across expressions.

Analysis of Memory Use (RQ4). To test whether the LLM uses the updated conditional memory to predict revised facts, we compare normal inference with disabling updates for the evaluated fact, randomly selecting other updates matched in number and n-gram length, or all updates. All condi-

tions use the same model after CounterFact editing and retain the original embeddings. As shown in Figure 5(c), fact-related disabling sharply lowers Efficacy and Generalization, almost matching the effect of disabling all updates. In contrast, matched random disabling leaves Efficacy and Generalization unchanged. An additional analysis of prediction preferences shows that fact-related disabling makes the model favor original over revised facts on average (see App. C.3.1). These results show that the LLM relies on the fact-related memory updates to access and utilize revised knowledge.

Additional Analyses. We extend editing to 5,000 facts to test stability (see App. C.1.1) and analyze ZsRE preservation (see App. C.1.3). We vary the number of generated expressions, editable n-gram lengths, regularization coefficients, and the clamp factor (see Apps. C.2.2–C.2.4). We further analyze memory use through memory activation and cross-edit sharing (see Apps. C.3.1–C.3.3).

5 Related Work

Conditional Memory and n-gram Embeddings. Lookup-based memory architectures expand model capacity through sparse access to stored vectors, as in Product-Key Memory (Lample et al., 2019) and Memory Layers at Scale (Berges et al., 2025). A related line uses n-gram embeddings to associate local token or byte sequences with reusable representations. 1) *Model scaling.* Byte Latent Transformer (Pagnoni et al., 2025), OverEncoding (Huang et al., 2025), and SCONE (Yu et al., 2025) use n-gram representations to enrich model inputs with local context at limited inference cost. Engram (Cheng et al., 2026b) and LongCat-Flash-Lite (Liu et al., 2026) further explore n-gram-based memory to scale LLM capacity. For memory construction, Memory Grafting (Cheng et al., 2026a) builds frozen conditional memory offline from another model's hidden states for pretraining. Related n-gram-based designs are also used in Qwen3.8-Flash-Next (Qiu et al., 2026) and DeepSeek-V4.1-Flash (DeepSeek-AI, 2026). 2) *Model adaptation.* Conditional memory has also been adapted for personalization through user-specific embedding updates in User as Engram (Li, 2026) and for domain adaptation in Engram Adapter (Hou & Wang, 2026). Our work develops EngramEdit to use pretrained conditional memory as an editable knowledge interface, enabling decoupled factual updates while largely preserving unrelated knowledge and general capabilities.

Knowledge Editing. KE methods can be grouped according to whether they modify the LLM's original parameters. 1) *Parameter-preserving methods.* These methods supply updated knowledge through external information or additional components, keeping the original parameters fixed. SERAC (Mitchell et al., 2022b) uses an edit cache, and IKE (Zheng et al., 2023) uses in-context demonstrations. Larimar (Das et al., 2024), WISE (Wang et al., 2024), and NeuralDB (Fei et al., 2026) use additional weights or memory components. 2) *Parameter-modifying methods.* More directly related to our work, these methods update existing model parameters. Constrained fine-tuning (Zhu et al., 2020) limits parameter changes during optimization, while MEND (Mitchell et al., 2022a) learns to transform editing gradients into weight updates. ROME (Meng et al., 2022) and MEMIT (Meng et al., 2023) use locate-then-edit to first compute target FFN outputs for new facts, then solve for the corresponding weight updates. UnKE (Deng et al., 2025) extends parameter-modifying editing to unstructured knowledge. To preserve unrelated knowledge during parameter updates, AlphaEdit (Fang et al., 2025) projects parameter changes into the null space of a preservation set. For different model architectures, STEM (Sadhukhan et al., 2026) revises facts through token-indexed embedding substitution, while MoEEdit (Gu et al., 2026) jointly updates MoE experts with constraints on routing shifts. EngramEdit focuses on realizing the potential of pretrained conditional memory for decoupled knowledge updates.

6 Conclusion and Future Work

In this work, we propose EngramEdit for decoupled knowledge updates via conditional memory, jointly updating shared n-gram embeddings with reuse-based regularization. Experiments show high editing success and generalization, multi-hop use of updated facts, and largely preserved unrelated knowledge and general capabilities. These findings show that EngramEdit enables decoupled knowledge updates in LLMs through conditional memory, extending its role beyond model scaling to an editable knowledge interface. Future work can explore other conditional memory architectures and model scales, and efficient, stable updates to more facts at higher frequencies.

AI USE STATEMENT

In this work, we used generative AI tools to assist with method implementation, proof drafting, manuscript writing and revision, and figure creation. AI tools also assisted in discussing and refining the interpretation of experimental results. The authors reviewed the method implementation and proofs and tested the code. We take full responsibility for the final content of this work, including all AI-assisted text, claims, and artifacts.

ETHICS STATEMENT

EngramEdit is intended to help LLMs keep factual knowledge up to date, but the ability to revise knowledge can also be misused to introduce false or misleading content. Our evaluation uses existing benchmarks, where counterfactual edit targets serve as experimental test cases and should not be treated as verified real-world facts. Responsible use therefore requires verifying factual updates, restricting editing access to authorized users, and auditing intended and unintended effects.

REPRODUCIBILITY STATEMENT

All the results in this work are reproducible. Section 3 describes the method, with the complete editing procedure, theoretical assumptions, and proofs provided in Appendix A. Section 4.1 and Appendix B document the datasets, evaluation protocols, and implementation settings. The code is available in our repository.

REFERENCES

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. The fifth PASCAL recognizing textual entailment challenge. In *Proceedings of the Second Text Analysis Conference*, 2009.

Vincent-Pierre Berges, Barlas Oguz, Daniel Haziza, Wen-Tau Yih, Luke Zettlemoyer, and Gargi Ghosh. Memory layers at scale. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 3831–3842. PMLR, 2025.

Runxi Cheng, Yuchen Guan, Yongxian Wei, Qianpu Sun, Qixiu Li, Sinan Du, Feng Xiong, Chun Yuan, Yan Lu, and Yeyun Gong. Memory grafting: Scaling language model pre-training via offline conditional memory. arXiv preprint arXiv:2605.20948, 2026a.

Xin Cheng, Wangding Zeng, Damai Dai, Qinyu Chen, Bingxuan Wang, Zhenda Xie, Kezhao Huang, Xingkai Yu, Zhewen Hao, Han Zhang, Yu-Kun Li, Huishuai Zhang, Dongyan Zhao, and Wenfeng Liang. Conditional memory via scalable lookup: A new axis of sparsity for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026b.

Payel Das, Subhajit Chaudhury, Elliot Nelson, Igor Melnyk, Sarathkrishna Swaminathan, Sihui Dai, Aurelie Lozano, Georgios Kollias, Vijil Chenthamarakshan, Jiri Navratil, Soham Dan, and Pin-Yu Chen. Larimar: Large language models with episodic memory control. In *Proceedings of the 41st International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2024.

DeepSeek-AI. Deepseek-v4.1-flash: Pushing the limits of kv cache compression. arXiv preprint arXiv:2609.19969, 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. In *The Thirteenth International Conference on Learning Representations*, 2025.

William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. AlphaEdit: Null-space constrained knowledge editing for language models. In *The Thirteenth International Conference on Learning Representations*, 2025.

Weizhi Fei, Hao Shi, Jing Xu, Jingchen Peng, Jiazheng Li, Jingzhao Zhang, Bo Bai, Wei Han, Zhenyuan Chen, and Xueyan Niu. Scaling knowledge editing in LLMs to 100,000 facts with neural KV database. In *The Fourteenth International Conference on Learning Representations*, 2026.

Yupu Gu, Rongzhe Wei, Andy Zhu, and Pan Li. MoEEdit: Efficient and routing-stable knowledge editing for mixture-of-experts LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021.

Jiayu Hou and Lei Wang. When to adapt: Conditional memory adapters for retention-preserving domain specialization. arXiv preprint arXiv:2608.29327, 2026.

Hongzhi Huang, Defa Zhu, Banggu Wu, Yutao Zeng, Ya Wang, Qiyang Min, and Zhou Xun. Over-tokenized transformer: Vocabulary is generally worth scaling. In *Forty-second International Conference on Machine Learning*, 2025.

Guillaume Lample, Alexandre Sablayrolles, Marc'Aurelio Ranzato, Ludovic Denoyer, and Herve Jegou. Large memory layers with product keys. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, 2019.

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. In *International Conference on Learning Representations*, 2021.

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*, pp. 333–342. Association for Computational Linguistics, 2017. doi: 10.18653/v1/K17-1034.

Bojie Li. User as engram: Internalizing per-user memory as local parametric edits. arXiv preprint arXiv:2606.19172, 2026.

Hong Liu, Jiaqi Zhang, Chao Wang, Xing Hu, Linkun Lyu, Jiaqi Sun, Xurui Yang, Bo Wang, Fengcun Li, Yulei Qian, Lingtong Si, Yerui Sun, Rumei Li, Peng Pei, Yuchen Xie, and Xunliang Cai. Scaling embeddings outperforms scaling experts in language models. arXiv preprint arXiv:2601.21204, 2026.

Kevin Meng, David Bau, Alex J Andonian, and Yonatan Belinkov. Locating and editing factual associations in GPT. In *Advances in Neural Information Processing Systems*, 2022.

Kevin Meng, Arnab Sen Sharma, Alex J Andonian, Yonatan Belinkov, and David Bau. Mass-editing memory in a transformer. In *The Eleventh International Conference on Learning Representations*, 2023.

Eric Mitchell, Charles Lin, Antoine Bosselut, Chelsea Finn, and Christopher D. Manning. Fast model editing at scale. In *International Conference on Learning Representations*, 2022a.

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *Proceedings of the 39th International Conference on Machine Learning*, Proceedings of Machine Learning Research, 2022b.

Artidoro Pagnoni, Ramakanth Pasunuru, Pedro Rodriguez, John Nguyen, Benjamin Muller, Margaret Li, Chunting Zhou, Lili Yu, Jason E Weston, Luke Zettlemoyer, Gargi Ghosh, Mike Lewis, Ari Holtzman, and Srini Iyer. Byte latent transformer: Patches scale better than tokens. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

Zihan Qiu, Zekun Wang, Xiao Li, Yanpeng Li, Yang Xu, Yixuan Wang, Huaqing Zhang, Rui Men, Bochao Mao, Chengruidong Zhang, Fan Zhou, Hao Luo, Haofeng Huang, Haoran Lian, Haoyan Huang, Hongqing Chen, Jianwei Zhang, Jing Xu, Junjie Wang, Langshi Chen, Liangyu Wang, Linlang Jiang, Man Yuan, Minmin Sun, Peng Jin, Siqi Zhang, Siyu Wang, Xingzhang Ren, Yakai Wang, Yi Zhang, Yiming Dong, Yizhong Cao, Yubo Ma, Yunfei Mao, Bo Zheng, and Dayiheng Liu. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. arXiv preprint arXiv:2608.30320, 2026.

Ranajoy Sadhukhan, Sheng Cao, Harry Dong, Changsheng Zhao, Attiano Purpura-Pontoniere, Yuandong Tian, Zechun Liu, and Beidi Chen. STEM: SCALING TRANSFORMERS WITH EMBEDDING MODULES. In *The Fourteenth International Conference on Learning Representations*, 2026.

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. In *International Conference on Learning Representations*, 2017.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, 2018.

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. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, 2024.

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

Da Yu, Edith Cohen, Badih Ghazi, Yangsibo Huang, Pritish Kamath, Ravi Kumar, Daogao Liu, and Chiyuan Zhang. Scaling embedding layers in language models. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

Qingru Zhang, Minshuo Chen, Alexander Bukharin, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adaptive budget allocation for parameter-efficient fine-tuning. In *The Eleventh International Conference on Learning Representations*, 2023.

Ce Zheng, Lei Li, Qingxiu Dong, Yuxuan Fan, Zhiyong Wu, Jingjing Xu, and Baobao Chang. Can we edit factual knowledge by in-context learning? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Singapore, 2023. Association for Computational Linguistics.

Zexuan Zhong, Zhengxuan Wu, Christopher Manning, Christopher Potts, and Danqi Chen. MQuAKE: Assessing knowledge editing in language models via multi-hop questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 2023. doi: 10.18653/v1/2023.emnlp-main.971.

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 METHOD DETAILS AND PROOFS

This section provides the background formulation, additional method details, and proofs referenced in Sections 2 and 3.

A.1 LOCATE-THEN-EDIT FORMULATION

Locate-then-edit first identifies an internal computation associated with the factual association being edited, then determines the output that this computation should produce for the desired prediction, and finally updates the corresponding parameters to reproduce that output. We illustrate this process using a single FFN output projection. Specific editors may use different layer-selection strategies, preservation constraints, or solvers (Meng et al., 2022; 2023; Fang et al., 2025; Gu et al., 2026).

Causal Localization. For the edit $e_i = (s_i, r_i, o_i \to o_i^\star)$, causal tracing measures how strongly an internal state contributes to recalling the original object o_i (Meng et al., 2022). The procedure first runs the edit prompt x_i with the subject-token embeddings corrupted. Let p_i^{corr} denote the resulting probability assigned to o_i. It then restores the clean hidden state at token position q and layer ℓ, retaining the input corruption and recomputing downstream states. Let $p_i^{\mathrm{restore}}(\ell, q)$ denote the probability after this restoration. The indirect effect of the restored state is

$$\mathrm{IE}_i(\ell, q) = p_i^{\mathrm{restore}}(\ell, q) - p_i^{\mathrm{corr}}. \tag{11}$$

A large indirect effect indicates that the state at (ℓ, q) contributes strongly to recalling the fact. This analysis identifies, or motivates a fixed choice of, the FFN layer ℓ and subject-token position q_i used for editing. At the selected location, let $\mathbf{k}_i^\ell$ be the post-nonlinearity FFN activation and $\mathbf{W}_{\mathrm{out}}^\ell$ the FFN output projection. In practice, the activation $\mathbf{k}_i^\ell$ can be averaged over prompts that place the same subject in different contexts to reduce dependence on one prompt (Meng et al., 2022; 2023). The corresponding FFN output is

$$\mathbf{v}_i^\ell = \mathbf{W}_{\mathrm{out}}^\ell \mathbf{k}_i^\ell. \tag{12}$$

The remaining two stages determine the output that should replace $\mathbf{v}_i^\ell$ and the parameter update that realizes this replacement.

Target Output Computation. Target output computation asks what FFN output at the localized position would make the model predict the desired object $o_i^\star$. Let $\mathcal{C}_i$ contain the edit prompt x_i and any context-augmented prompts used during this computation. With all model parameters fixed, a learnable perturbation $\boldsymbol{\delta}_i$ is temporarily added to the FFN output at the selected position for every prompt in $\mathcal{C}_i$. We denote the model evaluated with this temporary perturbation by $\mathcal{M}_{\boldsymbol{\delta}_i}$. The perturbation is obtained by minimizing

$$\begin{aligned} \boldsymbol{\delta}_i^\star = \arg\min_{\boldsymbol{\delta}_i} \frac{1}{|\mathcal{C}_i|} \sum_{x \in \mathcal{C}_i} \mathcal{L}_{\mathrm{NLL}}\big(\mathcal{M}_{\boldsymbol{\delta}_i}(x), o_i^\star\big) \\ + \lambda_{\mathrm{KL}} D_{\mathrm{KL}}\big(p_{\boldsymbol{\delta}_i}^{\mathrm{pre}} \,\|\, p_0^{\mathrm{pre}}\big), \end{aligned} \tag{13}$$

where p_0^{pre} and $p_{\boldsymbol{\delta}_i}^{\mathrm{pre}}$ are the predictive distributions on a preservation prompt before and during the temporary perturbation, respectively (Meng et al., 2022). The NLL term promotes the desired prediction, while the KL term limits changes to the preservation distribution. The optimized perturbation defines the target FFN output

$$\mathbf{v}_i^\star = \mathbf{v}_i^\ell + \boldsymbol{\delta}_i^\star. \tag{14}$$

The perturbation $\boldsymbol{\delta}_i^\star$ is used only to determine this target. It is removed after optimization and is not written to the model.

Parameter Update. The final stage changes the FFN output projection so that it produces the target output from the same activation. For a batch of m edits applied to layer ℓ, define the activation matrix $\mathbf{K} = [\mathbf{k}_1^\ell, \ldots, \mathbf{k}_m^\ell]$ and the target output matrix $\mathbf{V}^\star = [\mathbf{v}_1^\star, \ldots, \mathbf{v}_m^\star]$. The parameter update $\Delta\mathbf{W}_{\mathrm{out}}^\ell$ should satisfy

$$(\mathbf{W}_{\mathrm{out}}^\ell + \Delta\mathbf{W}_{\mathrm{out}}^\ell)\mathbf{K} \approx \mathbf{V}^\star. \tag{15}$$

Matching only the new targets can change outputs associated with other FFN activations. Let $\mathbf{K}_0$ contain preservation activations sampled from the model. Because their original outputs are

$\mathbf{W}^{\ell}_{\text{out}}\mathbf{K}_0$, preserving these associations amounts to keeping $\Delta\mathbf{W}^{\ell}_{\text{out}}\mathbf{K}_0$ small. With nonnegative coefficients $\lambda_{\text{preserve}}$ and λ_{ridge}, a regularized batch formulation is

$$\begin{aligned}\Delta\mathbf{W}^{\ell\star}_{\text{out}} = \arg\min_{\Delta\mathbf{W}^{\ell}_{\text{out}}} \; & \left\|(\mathbf{W}^{\ell}_{\text{out}} + \Delta\mathbf{W}^{\ell}_{\text{out}})\mathbf{K} - \mathbf{V}^{\star}\right\|_F^2 \\ & + \lambda_{\text{preserve}} \left\|\Delta\mathbf{W}^{\ell}_{\text{out}}\mathbf{K}_0\right\|_F^2 + \lambda_{\text{ridge}} \left\|\Delta\mathbf{W}^{\ell}_{\text{out}}\right\|_F^2 .\end{aligned} \tag{16}$$

Define the target residual matrix and the regularized covariance matrix as

$$\mathbf{R} = \mathbf{V}^{\star} - \mathbf{W}^{\ell}_{\text{out}}\mathbf{K}, \qquad \mathbf{C}_0 = \lambda_{\text{preserve}}\mathbf{K}_0\mathbf{K}_0^{\top} + \lambda_{\text{ridge}}\mathbf{I}. \tag{17}$$

When $\mathbf{K}\mathbf{K}^{\top} + \mathbf{C}_0$ is invertible, setting the derivative of Eq. 16 to zero gives

$$\Delta\mathbf{W}^{\ell\star}_{\text{out}} = \mathbf{R}\mathbf{K}^{\top}\left(\mathbf{K}\mathbf{K}^{\top} + \mathbf{C}_0\right)^{-1}, \tag{18}$$

which jointly writes a batch of target FFN outputs while regularizing changes on preservation activations (Meng et al., 2023). A positive λ_{ridge} ensures this invertibility.

For a single edit, ROME instead requires exact target matching and minimizes the preservation error under this constraint (Meng et al., 2022). With $\mathbf{C}_0 = \mathbf{K}_0\mathbf{K}_0^{\top}$ invertible and $\mathbf{k}_i^{\ell} \neq \mathbf{0}$, the resulting rank-one update is

$$\Delta\mathbf{W}^{\ell\star}_{\text{out}} = \frac{(\mathbf{v}_i^{\star} - \mathbf{W}^{\ell}_{\text{out}}\mathbf{k}_i^{\ell})(\mathbf{C}_0^{-1}\mathbf{k}_i^{\ell})^{\top}}{(\mathbf{C}_0^{-1}\mathbf{k}_i^{\ell})^{\top}\mathbf{k}_i^{\ell}}. \tag{19}$$

Later editors modify the preservation constraint, the parameter blocks being updated, or the batch solver, while retaining the separation between target output computation and persistent parameter updating (Fang et al., 2025; Gu et al., 2026).

A.2 Conditional Memory Target Computation

We provide the prompt for expression generation and the full objective for target computation in Section 3.1.

A.2.1 Expression Generation

To cover different n-grams activated at the last subject token, we ask the model to vary the wording immediately before the subject. Figure 6 shows the prompt used for CounterFact and each atomic edit in MQuAKE. Here, `<original stem>` is the edit prompt x_i with the subject s_i replaced by `[SUBJ]`; `<old answer>` and `<new answer>` are the original and desired objects. For ZsRE, the instruction instead requests a question with the same meaning and answer type.

The instruction is passed as a user message through the model's chat template. We sample candidates in batches and filter duplicates, malformed outputs, and outputs containing answer strings. Later rounds append accepted expressions and ask the model to vary their wording before `[SUBJ]`. Generation stops once enough expressions are accepted or the round limit is reached. We restore the original subject in each accepted expression before target computation.

A.2.2 Target Computation Objective

We compute target memory representations by optimizing a temporary perturbation $\boldsymbol{\delta}_i$ using the prediction loss (Meng et al., 2023). The perturbation is shared across the expression set $\mathcal{X}_i$ (see Section 3.1). For a desired object with tokens $o_i^{\star} = (o_{i,1}^{\star}, \ldots, o_{i,L_i}^{\star})$, the NLL term in Eq. 4 is

$$\mathcal{L}_{\text{NLL}}\big(\mathcal{M}_{\boldsymbol{\delta}_i}(x), o_i^{\star}\big) = -\frac{1}{L_i}\sum_{r=1}^{L_i} \log p_{\mathcal{M}_{\boldsymbol{\delta}_i}}\big(o_{i,r}^{\star} \mid x, o_{i,<r}^{\star}\big). \tag{20}$$

Here, $o_{i,<r}^{\star}$ denotes the preceding tokens of the desired object; the loss averages over its L_i tokens.

The target regularizer combines a preservation loss with a norm penalty (Meng et al., 2022):

$$\mathcal{R}_{\text{target}}(\boldsymbol{\delta}_i) = \lambda_{\text{KL}} D_{\text{KL}}\big(p^{\text{pre}}_{\boldsymbol{\delta}_i} \,\|\, p^{\text{pre}}_0\big) + \lambda_{\text{norm}} \frac{\|\boldsymbol{\delta}_i\|_2}{\|\mathbf{h}_i(x_i)\|_2^2}, \tag{21}$$

Expression Generation Prompt

```
Rewrite the factual prompt stem below into one alternative stem with the same
meaning.
Rules:
- The output must contain [SUBJ] exactly once.
- Keep it as an incomplete factual sentence stem that naturally expects a
completion.
- Keep the meaning aligned with the original stem.
- Prioritize changing the local phrasing immediately before [SUBJ] when
natural.
- Good ways to vary it include changing the relation phrase or preposition
before [SUBJ], using a possessive form like [SUBJ]'s ..., or rewriting as a
question.
- Do not rely only on changing words after [SUBJ]; try to make the left-side
local frame meaningfully different.
- Natural paraphrases are more important than forcing an unnatural token
pattern.
- Do not use the answer words '<old answer>' or '<new answer>'.
- Do not use placeholders like ...  or ____.
- Do not add numbering, explanation, quotation marks, or multiple options.
- Output exactly one rewritten stem and nothing else.

Original stem:  <original stem>
```

Figure 6: Expression-generation prompt for CounterFact and atomic MQuAKE edits. `[SUBJ]` marks the subject; angle-bracketed fields are filled from the edit request.

where p_0^{pre} and $p_{\boldsymbol{\delta}_i}^{\text{pre}}$ are the predictive distributions before and during target computation on the preservation prompt "s_i is a". The KL term limits prediction changes when describing the subject. The norm penalty discourages large perturbations, using the original edit prompt's memory representation $\mathbf{h}_i(x_i)$ for normalization. The clamp factor ρ further bounds the perturbation norm through $\|\boldsymbol{\delta}_i\|_2 \leq \rho\|\mathbf{h}_i(x_i)\|_2$.

Adding the optimized perturbation $\boldsymbol{\delta}_i^\star$ to each current memory representation $\mathbf{h}_i(x)$ gives its target $\mathbf{h}_i^\star(x)$. The perturbation is then removed; only the n-gram embedding updates derived below are written to conditional memory. Appendix B.4.2 gives the prefix contexts and optimization settings used in our experiments.

A.3 Joint Memory Update Allocation

To realize the target memory representations, we express how embedding updates change the memory computation. Sum aggregation gives a linear matching relation, while context-aware gating requires accounting for changes in the gate. We derive these two cases and then specify the reuse-based regularization weights used in Section 3.3.

A.3.1 Sum Aggregation

We first derive how embedding updates combine to match each target under sum aggregation. LongCat-Flash-Lite uses additive fusion by averaging projected n-gram embeddings with the token embedding (Liu et al., 2026). Here, we use the simplified sum formulation from Section 3.3; Appendix B.4.3 describes how updates are stored and applied in our implementation. For row p of the mapping matrix $\mathbf{A}$, the current memory representation is

$$\mathbf{h}_p = \mathbf{c}_p + \sum_{j=1}^{Q} A_{pj}\mathbf{e}(g_j), \tag{22}$$

where $\mathbf{c}_p$ contains the components that are not updated. Since these components remain fixed, applying the embedding updates $\{\mathbf{u}_j\}_{j=1}^{Q}$ changes only the summed embeddings:

$$\widetilde{\mathbf{h}}_p(\mathbf{U}) = \mathbf{h}_p + \sum_{j=1}^{Q} A_{pj}\mathbf{u}_j. \tag{23}$$

The fixed components cancel when we subtract the current representation from its target. Matching the target memory representation $\mathbf{h}_p^\star$ therefore requires

$$\sum_{j=1}^{Q} A_{pj}\mathbf{u}_j \approx \mathbf{h}_p^\star - \mathbf{h}_p = \mathbf{b}_p. \tag{24}$$

Stacking these relations over the edit batch gives $\mathbf{AU} \approx \mathbf{B}$, so one embedding update is jointly constrained by all expressions that use it.

A.3.2 Context-Aware Gated Aggregation

To extend target matching to context-aware gating, we account for how embedding updates affect both the projected values and their gates. Engram (Cheng et al., 2026b), Qwen3.8-Flash-Next (Qiu et al., 2026), and DeepSeek-V4.1-Flash (DeepSeek-AI, 2026) use contextual gating to control the contribution of n-gram embeddings. We illustrate the matching relation using Engram's scalar gate.

For expression x_p, let $\mathbf{z}_p \in \mathbb{R}^{d_z}$ concatenate the activated embeddings and $\mathbf{q}_p \in \mathbb{R}^d$ denote the contextual query supplied by the backbone. Here, d_z is the dimension of the concatenated embedding vector. We first consider updates that leave the upstream query computation unchanged, so $\mathbf{q}_p$ remains fixed. The memory function $\mathbf{F}_p$ and gate α_p are

$$\mathbf{h}_p = \mathbf{F}_p(\mathbf{z}_p; \mathbf{q}_p), \qquad \alpha_p = \sigma\left(\frac{\mathrm{RMSNorm}(\mathbf{q}_p)^\top \mathrm{RMSNorm}(\mathbf{W}_K\mathbf{z}_p)}{\sqrt{d}}\right), \tag{25}$$

where σ is the sigmoid function and $\mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d_z}$ are fixed key and value projection matrices. The memory function $\mathbf{F}_p$ includes the gated value $\alpha_p\mathbf{W}_V\mathbf{z}_p$ and any subsequent architecture-specific processing, whose parameters also remain fixed.

To apply the same update wherever an embedding is shared, we use the batch-level embedding update matrix $\mathbf{U}$. Its vectorization $\mathrm{vec}(\mathbf{U}) \in \mathbb{R}^{Qd}$ stacks its columns. The selection operator $\mathbf{S}_p \in \mathbb{R}^{d_z\times Qd}$ places each update wherever its embedding occurs in $\mathbf{z}_p$, giving

$$\widetilde{\mathbf{h}}_p(\mathbf{U}) = \mathbf{F}_p\left(\mathbf{z}_p + \mathbf{S}_p\,\mathrm{vec}(\mathbf{U}); \mathbf{q}_p\right). \tag{26}$$

Local processing such as a causal convolution can make the memory representation depend on earlier token positions. In that case, the mapping matrix $\mathbf{A}$ and selection operator $\mathbf{S}_p$ include the activated n-grams throughout its receptive field. To match targets after the gated computation, we replace the additive matching term in Eq. 9 with

$$\sum_{p=1}^{P}\left\|\widetilde{\mathbf{h}}_p(\mathbf{U}) - \mathbf{h}_p^\star\right\|_2^2, \tag{27}$$

while retaining the same reuse-based regularization. The resulting objective can be optimized with gradients.

To use a linear solver, we can instead approximate the gated computation locally at the current embeddings. Let $\mathbf{J}_p \in \mathbb{R}^{d\times d_z}$ be the Jacobian of $\mathbf{F}_p$ with respect to $\mathbf{z}_p$, with the query $\mathbf{q}_p$ fixed. A first-order approximation gives

$$\widetilde{\mathbf{h}}_p(\mathbf{U}) - \mathbf{h}_p \approx \mathbf{J}_p\mathbf{S}_p\,\mathrm{vec}(\mathbf{U}). \tag{28}$$

Stacking these relations gives a regularized linear system that approximates the gated objective, using the same matching targets $\mathbf{b}_p = \mathbf{h}_p^\star - \mathbf{h}_p$ and regularization coefficients λ_j as the additive formulation. If embedding updates also affect the upstream computation, the contextual query $\mathbf{q}_p$ is recomputed during optimization, and the Jacobian includes its dependence on the updates. Thus, changing the aggregation modifies the matching relation and solver while retaining target computation, memory mapping, and reuse-based regularization.

A.3.3 Regularization Weight Construction

For either aggregation, we penalize updates to frequently reused embeddings more strongly to limit unintended changes on unrelated inputs. Shorter n-grams can be shared by more inputs, while corpus frequency measures how often each n-gram occurs. We therefore use length and frequency as indicators of reuse. For the n-gram g_j, let n_j denote its length and $\pi_j \in [0, 1]$ its within-length corpus-frequency percentile. The regularization weight w_j combines these two indicators:

$$\begin{aligned} w_j^{\text{len}} &= \eta_{n_j}, \\ w_j^{\text{freq}} &= \min\left(1 + \gamma \pi_j^{\beta},\, w_{\text{freq}}^{\max}\right), \\ w_j &= \min\left(w_j^{\text{len}} + w_j^{\text{freq}} - 1, w^{\max}\right), \end{aligned} \tag{29}$$

where $\eta_n > 0$ decreases with n to assign stronger penalties to shorter n-grams. The coefficient $\gamma \geq 0$ and exponent $\beta > 0$ control the strength and shape of frequency weighting. Subtracting 1 makes frequency weighting an increment over the length weight. The caps $w_{\text{freq}}^{\max} \geq 1$ and $w^{\max} > 0$ prevent either the frequency weight or the combined weight from becoming excessively large.

To control the overall penalty strength, we set the regularization coefficient $\lambda_j = \lambda_{\text{reuse}} w_j + \lambda_{\text{ridge}}$. The coefficient λ_{reuse} scales the reuse-based weights; the optional λ_{ridge} adds a uniform penalty independently of length and frequency. Setting $\lambda_{\text{ridge}} = 0$ removes this uniform term. Both coefficients are nonnegative and chosen so that every $\lambda_j > 0$. Under sum aggregation, Appendix A.5.2 relates length and frequency to an upper bound on memory representation changes for unrelated inputs, providing a theoretical basis for these weights.

A.4 Complete Editing Procedure

Algorithm 1 summarizes EngramEdit for one edit batch under sum aggregation. Target computation precedes memory mapping, and the selected n-gram embeddings are updated only after solving the batch-level joint problem. For context-aware gated aggregation, the matching objective and solver are replaced as described in Appendix A.3.2.

Sequential Editing. Sequential editing repeats this procedure on the current conditional memory. At step t, both the target memory representations and the matrices $\mathbf{B}_t$ and $\mathbf{\Lambda}_t$ are computed from the model after steps $1, \ldots, t-1$. The solved update is then added to the current n-gram embedding. This procedure accounts for previous changes whenever a later edit activates an n-gram that has already been updated.

Computational Cost. Let s be the maximum number of configured n-grams activated by one expression. For an edit batch with P expressions, Q unique n-grams, and embedding dimension d, forming the sparse normal equations costs $O(Ps^2 + Psd)$. Solving the resulting dense $Q \times Q$ system costs $O(Q^3 + Q^2 d)$ and uses $O(Q^2 + Qd)$ memory, excluding model storage. If no selected n-gram embedding is shared between groups of expressions, the system can be split into smaller systems. Solving them separately preserves the joint solution and can reduce solver time and memory (see Appendix A.5.6 for the decomposition proof).

A.5 Proofs for Joint Memory Update Allocation

We analyze the joint update in Section 3.3 under sum aggregation (see Appendix A.3.1). The results explain the need for approximate matching, support reuse-based regularization, and characterize the solution's update cost, sensitivity, and computation.

We use the mapping matrix $\mathbf{A} \in \mathbb{R}^{P \times Q}$, matching target matrix $\mathbf{B} \in \mathbb{R}^{P \times d}$, and embedding update matrix $\mathbf{U} \in \mathbb{R}^{Q \times d}$. The jth row of $\mathbf{U}$ is $\mathbf{u}_j^\top$. For the regularized objective, $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \ldots, \lambda_Q)$ with every $\lambda_j > 0$. We write this objective as

$$F(\mathbf{U}) = \|\mathbf{A}\mathbf{U} - \mathbf{B}\|_F^2 + \|\mathbf{\Lambda}^{1/2}\mathbf{U}\|_F^2, \tag{30}$$

and let $\mathbf{U}^\star$ denote a minimizer. The notation $\|\cdot\|_F$ denotes the Frobenius norm; $\|\cdot\|_2$ denotes the Euclidean norm for vectors and the spectral norm for matrices.

Algorithm 1 EngramEdit for One Edit Batch

Require: Current model $\mathcal{M}$ and edit batch $\mathcal{E} = \{e_i\}_{i=1}^{m}$
Require: Number of generated expressions K and configured n-gram lengths $\mathcal{N}$
Require: Regularization coefficients $\lambda_{\text{reuse}}, \lambda_{\text{ridge}} \geq 0$ and clamp factor $\rho > 0$
Require: $\lambda_j = \lambda_{\text{reuse}} w_j + \lambda_{\text{ridge}} > 0$ for each selected embedding
Ensure: Edited model $\mathcal{M}'$ with an updated conditional memory

for all $e_i = (s_i, r_i, o_i \to o_i^\star) \in \mathcal{E}$ **do**
  $x_i \leftarrow$ original edit prompt for (s_i, r_i) ▷ Expression Generation
  $\mathcal{P}_i \leftarrow \text{GenerateExpressions}(\mathcal{M}, e_i, K)$
  $\mathcal{X}_i \leftarrow \{x_i\} \cup \mathcal{P}_i$
  for all $x \in \mathcal{X}_i$ **do** ▷ Joint Target Computation
    $q_i(x) \leftarrow$ last subject-token position in x
    $\mathbf{h}_i(x) \leftarrow \mathbf{h}(x, q_i(x))$
  end for
  Compute the shared perturbation $\boldsymbol{\delta}_i^\star$ using Eq. 4 with clamp factor ρ
  $\mathbf{h}_i^\star(x) \leftarrow \mathbf{h}_i(x) + \boldsymbol{\delta}_i^\star$ for all $x \in \mathcal{X}_i$
end for
$\mathcal{R} \leftarrow \{(i, x) : e_i \in \mathcal{E},\ x \in \mathcal{X}_i\}$
$\mathcal{G}_i(x) \leftarrow \mathcal{G}(x, q_i(x))$ for all $(i, x) \in \mathcal{R}$ ▷ Memory Mapping
Enumerate $\mathcal{R}$ as $\{(i_p, x_p)\}_{p=1}^{P}$, where $P = |\mathcal{R}|$
$\{g_j\}_{j=1}^{Q} \leftarrow \text{Unique}\big(\bigcup_{(i,x)\in\mathcal{R}} \mathcal{G}_i(x)\big)$
$A_{pj} \leftarrow \mathbb{I}[g_j \in \mathcal{G}_{i_p}(x_p)]$ for all p, j ▷ Mapping Matrix
$\mathbf{b}_p \leftarrow \boldsymbol{\delta}_{i_p}^\star$ for all p; $\mathbf{B} \leftarrow [\mathbf{b}_1, \ldots, \mathbf{b}_P]^\top$ ▷ Target Matching
Compute w_j using Eq. 29 and set $\lambda_j \leftarrow \lambda_{\text{reuse}} w_j + \lambda_{\text{ridge}}$
$\boldsymbol{\Lambda} \leftarrow \text{diag}(\lambda_1, \ldots, \lambda_Q)$ ▷ Reuse-Based Regularization
$\mathbf{U}^\star \leftarrow \text{Solve}(\mathbf{A}^\top \mathbf{A} + \boldsymbol{\Lambda}, \mathbf{A}^\top \mathbf{B})$ ▷ Joint Update Solution
for all selected n-grams g_j **do** ▷ Memory Updating
  $\mathbf{e}(g_j) \leftarrow \mathbf{e}(g_j) + \mathbf{u}_j^\star$
end for
return $\mathcal{M}'$

A.5.1 Feasibility of Exact Matching

Shared embeddings can make matching targets incompatible. For example, expressions that activate the same selected n-grams receive the same memory representation change. If their matching targets differ, no embedding update can satisfy both exactly. The following condition identifies when exact matching is possible.

Proposition 1. Exact matching $\mathbf{AU} = \mathbf{B}$ is possible if and only if every column of the matching target matrix $\mathbf{B}$ lies in the column space of the mapping matrix $\mathbf{A}$. Equivalently,

$$\mathbf{z}^\top \mathbf{B} = \mathbf{0}^\top \quad \text{for every } \mathbf{z} \in \text{null}(\mathbf{A}^\top). \tag{31}$$

Proof. Exact matching separates into one equation per embedding dimension, $\mathbf{AU}_{:,r} = \mathbf{B}_{:,r}$ for $r = 1, \ldots, d$. Each equation has a solution precisely when its target column belongs to $\text{col}(\mathbf{A})$. Stacking these column solutions gives an exact update matrix. The identity $\text{col}(\mathbf{A}) = \text{null}(\mathbf{A}^\top)^\perp$ gives the equivalent condition in Eq. 31.

When exact matching is infeasible, the squared matching error in Eq. 9 allows a joint compromise across conflicting targets. The proposition concerns feasibility; positive regularization can also favor an inexact match when an exact match exists.

A.5.2 Reuse-Based Regularization

An embedding update can also change memory representations on unrelated inputs that activate the same n-gram. We relate these changes to activation probabilities to explain why frequently reused embeddings receive stronger update penalties.

Let $\mathcal{D}$ be a fixed distribution over unrelated input–position pairs (x, q). For each selected n-gram g_j, define its activation indicator $a_j(x, q) = \mathbb{I}[g_j \in \mathcal{G}(x, q)]$ and activation probability $p_j = \mathbb{E}_{\mathcal{D}}[a_j(x, q)]$. Let $s \leq |\mathcal{N}|$ be the maximum number of selected n-grams activated at one position under the lookup in Section 2. For fixed embedding updates,

$$\widetilde{\mathbf{h}}(x, q; \mathbf{U}) - \mathbf{h}(x, q) = \sum_{j=1}^{Q} a_j(x, q)\mathbf{u}_j. \tag{32}$$

Proposition 2. For any fixed embedding updates $\{\mathbf{u}_j\}_{j=1}^{Q}$, the expected squared memory representation change satisfies

$$\mathbb{E}_{\mathcal{D}}\left[\|\widetilde{\mathbf{h}}(x, q; \mathbf{U}) - \mathbf{h}(x, q)\|_2^2\right] \leq s \sum_{j=1}^{Q} p_j \|\mathbf{u}_j\|_2^2. \tag{33}$$

No independence assumption between n-gram activations is required.

Proof. Fix an input–position pair (x, q) and let $\mathcal{J}(x, q) = \{j : a_j(x, q) = 1\}$ index its activated selected n-grams. The triangle and Cauchy–Schwarz inequalities give

$$\begin{aligned} \left\| \sum_{j=1}^{Q} a_j(x, q)\mathbf{u}_j \right\|_2^2 &\leq \left(\sum_{j \in \mathcal{J}(x,q)} \|\mathbf{u}_j\|_2 \right)^2 \\ &\leq |\mathcal{J}(x, q)| \sum_{j \in \mathcal{J}(x,q)} \|\mathbf{u}_j\|_2^2 \leq s \sum_{j=1}^{Q} a_j(x, q)\|\mathbf{u}_j\|_2^2. \end{aligned} \tag{34}$$

The bound also holds when no selected n-gram is activated, since both sides are zero. Taking expectations and using $\mathbb{E}_{\mathcal{D}}[a_j(x, q)] = p_j$ proves Eq. 33 by linearity of expectation.

The bound assigns a larger cost to the same embedding update when its n-gram is activated more often on unrelated inputs. This provides a theoretical basis for penalizing large updates to frequently reused embeddings.

Bounds for the Solved Update. The activation bound applies to arbitrary updates. For the updates returned by the joint objective, regularization also limits the total weighted update cost, yielding the following bounds.

Corollary 1. The minimizer $\mathbf{U}^\star$ of Eq. 30 satisfies

$$\|\mathbf{u}_j^\star\|_2 \leq \frac{\|\mathbf{B}\|_F}{\sqrt{\lambda_j}} \quad \text{for every } j, \tag{35}$$

and its expected squared memory representation change on unrelated inputs satisfies

$$\mathbb{E}_{\mathcal{D}}\left[\|\widetilde{\mathbf{h}}(x, q; \mathbf{U}^\star) - \mathbf{h}(x, q)\|_2^2\right] \leq s \left(\max_j \frac{p_j}{\lambda_j} \right) \|\mathbf{B}\|_F^2. \tag{36}$$

Proof. The zero update is feasible, so optimality and nonnegativity of the matching error give

$$\sum_{j=1}^{Q} \lambda_j \|\mathbf{u}_j^\star\|_2^2 \leq F(\mathbf{U}^\star) \leq F(\mathbf{0}) = \|\mathbf{B}\|_F^2. \tag{37}$$

Each summand is nonnegative. Bounding the jth term by $\|\mathbf{B}\|_F^2$, dividing by $\lambda_j > 0$, and taking square roots gives Eq. 35. For the second bound, apply Proposition 2 and then use

$$\sum_{j=1}^{Q} p_j \|\mathbf{u}_j^\star\|_2^2 \leq \left(\max_j \frac{p_j}{\lambda_j} \right) \sum_{j=1}^{Q} \lambda_j \|\mathbf{u}_j^\star\|_2^2 \leq \left(\max_j \frac{p_j}{\lambda_j} \right) \|\mathbf{B}\|_F^2. \tag{38}$$

For fixed matching targets, increasing λ_j tightens the bound on its embedding update. With the unrelated-input distribution also fixed, the bound on memory representation changes decreases when the largest ratio p_j/λ_j decreases. Thus, the bounds connect stronger penalties on frequently activated embeddings to control of memory representation changes on unrelated inputs.

Length and Frequency. The activation probabilities on unrelated inputs are not directly available, so EngramEdit uses n-gram length and corpus frequency as indicators. If the selected n-gram g_k is a shorter suffix of the selected n-gram g_j, every activation of g_j also activates g_k. Hence,

$$a_j(x, q) \leq a_k(x, q) \quad \Longrightarrow \quad p_j \leq p_k. \tag{39}$$

This ordering holds for suffix-related n-grams, not every pair of different lengths.

For a corpus of N_{corpus} input–position pairs, let c_j count the positions activating g_j. Its activation probability under the uniform empirical distribution is $\widehat{p}_j = c_j/N_{\text{corpus}}$. Counts and empirical activation probabilities therefore have the same ordering. The within-length frequency ranks in Appendix A.3.3 retain this ordering within each length, but do not estimate p_j on unrelated inputs. These relations motivate the length and frequency factors without establishing optimality of the particular weight formula. The bounds concern memory representations, not downstream predictions.

A.5.3 Closed-Form Solution and Uniqueness

Overlapping activations can make columns of the mapping matrix linearly dependent. Positive regularization ensures a unique joint update without requiring this matrix to have full column rank.

Proposition 3. If every regularization coefficient satisfies $\lambda_j > 0$, the objective in Eq. 9 has the unique solution in Eq. 10.

Proof. Differentiating Eq. 30 gives $\nabla_{\mathbf{U}} F = 2\mathbf{A}^\top(\mathbf{AU} - \mathbf{B}) + 2\mathbf{\Lambda U}$. Setting the gradient to zero yields

$$(\mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda})\mathbf{U} = \mathbf{A}^\top\mathbf{B}. \tag{40}$$

For any nonzero vector $\mathbf{z} \in \mathbb{R}^Q$,

$$\mathbf{z}^\top(\mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda})\mathbf{z} = \|\mathbf{Az}\|_2^2 + \sum_{j=1}^{Q} \lambda_j z_j^2 > 0. \tag{41}$$

The coefficient matrix is therefore positive definite, making the objective strictly convex. Its unique stationary point is the global minimizer,

$$\mathbf{U}^\star = (\mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda})^{-1}\mathbf{A}^\top\mathbf{B}. \tag{42}$$

A.5.4 Minimum-Cost Update Allocation

Different embedding updates can produce the same memory representation changes on the edit expressions. Extra update components that leave these changes unchanged do not improve target matching, but can affect unrelated inputs that activate different subsets of embeddings. We show that the joint solution avoids such extra components by minimizing the weighted update cost for its achieved memory representation changes.

Proposition 4. Let $\mathbf{U}^\star$ minimize Eq. 30 with every $\lambda_j > 0$. Any alternative embedding update matrix $\mathbf{V} \in \mathbb{R}^{Q\times d}$ satisfying $\mathbf{AV} = \mathbf{AU}^\star$ obeys

$$\|\mathbf{\Lambda}^{1/2}\mathbf{V}\|_F^2 = \|\mathbf{\Lambda}^{1/2}\mathbf{U}^\star\|_F^2 + \|\mathbf{\Lambda}^{1/2}(\mathbf{V} - \mathbf{U}^\star)\|_F^2. \tag{43}$$

Thus, $\mathbf{U}^\star$ uniquely minimizes the weighted update cost among all updates producing the same achieved memory representation changes $\mathbf{AU}^\star$.

Proof. Let $\mathbf{D} = \mathbf{V} - \mathbf{U}^\star$. The shared matching result gives $\mathbf{AD} = \mathbf{0}$, and the normal equation gives $\mathbf{\Lambda U}^\star = \mathbf{A}^\top(\mathbf{B} - \mathbf{AU}^\star)$. Together, these identities imply

$$\operatorname{tr}(\mathbf{D}^\top \mathbf{\Lambda U}^\star) = \operatorname{tr}\big((\mathbf{AD})^\top(\mathbf{B} - \mathbf{AU}^\star)\big) = 0, \tag{44}$$

where tr denotes the matrix trace. Expanding the weighted cost of $\mathbf{V} = \mathbf{U}^\star + \mathbf{D}$ then gives

$$\begin{aligned} \|\mathbf{\Lambda}^{1/2}\mathbf{V}\|_F^2 &= \|\mathbf{\Lambda}^{1/2}\mathbf{U}^\star\|_F^2 + 2\operatorname{tr}(\mathbf{D}^\top \mathbf{\Lambda U}^\star) + \|\mathbf{\Lambda}^{1/2}\mathbf{D}\|_F^2 \\ &= \|\mathbf{\Lambda}^{1/2}\mathbf{U}^\star\|_F^2 + \|\mathbf{\Lambda}^{1/2}\mathbf{D}\|_F^2. \end{aligned} \tag{45}$$

Since every $\lambda_j > 0$, the additional cost is strictly positive unless $\mathbf{D} = \mathbf{0}$, proving minimality and uniqueness under the matching constraint.

The comparison fixes the achieved changes $\mathbf{AU}^\star$, so it remains valid when exact matching to $\mathbf{B}$ is infeasible. Any additional update with columns in $\operatorname{null}(\mathbf{A})$ increases the weighted cost without improving the matched representations. This is a guarantee on weighted update cost, not on downstream prediction changes.

A.5.5 Sensitivity to Target Changes

Target memory representations are computed through optimization and may vary slightly. We therefore examine how these variations affect the solved embedding updates when the mapping and regularization remain fixed. The following bound shows how positive regularization controls this sensitivity.

Proposition 5. For fixed mapping matrix $\mathbf{A}$ and positive definite regularization matrix $\mathbf{\Lambda}$, let $\Delta\mathbf{B}$ be a change in the matching target matrix. The corresponding solution change satisfies

$$\begin{aligned} \Delta\mathbf{U}^\star &= (\mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda})^{-1}\mathbf{A}^\top\Delta\mathbf{B}, \\ \|\Delta\mathbf{U}^\star\|_F &\le \frac{\|\mathbf{A}\|_2}{\lambda_{\min}(\mathbf{\Lambda})}\|\Delta\mathbf{B}\|_F. \end{aligned} \tag{46}$$

Proof. Write $\mathbf{H} = \mathbf{A}^\top\mathbf{A} + \mathbf{\Lambda}$ and let $\mathbf{U}^\star(\mathbf{B})$ denote the solution for matching targets $\mathbf{B}$. Subtracting the normal equations for $\mathbf{B} + \Delta\mathbf{B}$ and $\mathbf{B}$ gives

$$\mathbf{H}\Delta\mathbf{U}^\star = \mathbf{A}^\top\Delta\mathbf{B}, \qquad \Delta\mathbf{U}^\star = \mathbf{U}^\star(\mathbf{B} + \Delta\mathbf{B}) - \mathbf{U}^\star(\mathbf{B}). \tag{47}$$

Multiplying by $\mathbf{H}^{-1}$ proves the exact relation. For every unit vector $\mathbf{z}$, $\mathbf{z}^\top\mathbf{Hz} \ge \mathbf{z}^\top\mathbf{\Lambda z} \ge \lambda_{\min}(\mathbf{\Lambda})$. Thus, $\|\mathbf{H}^{-1}\|_2 \le 1/\lambda_{\min}(\mathbf{\Lambda})$, and

$$\|\Delta\mathbf{U}^\star\|_F \le \|\mathbf{H}^{-1}\|_2\|\mathbf{A}^\top\|_2\|\Delta\mathbf{B}\|_F \le \frac{\|\mathbf{A}\|_2}{\lambda_{\min}(\mathbf{\Lambda})}\|\Delta\mathbf{B}\|_F. \tag{48}$$

The final step uses $\|\mathbf{A}^\top\|_2 = \|\mathbf{A}\|_2$.

For a fixed mapping, the smallest regularization coefficient controls the sensitivity bound even when the mapping matrix is rank deficient. The bound quantifies possible amplification of target changes; it does not require this amplification to be below one.

A.5.6 Component Decomposition

Joint solving is needed only among expressions connected through shared embeddings. When groups share no selected n-gram embeddings, solving them separately can reduce computation without changing the joint solution.

Proposition 6. The joint update decomposes exactly over disconnected components of the bipartite graph between expression rows and selected n-gram columns, with an edge wherever $A_{pj} \neq 0$.

Proof. Ordering rows and columns by connected component puts the mapping matrix into block-diagonal form,

$$\mathbf{A} = \mathrm{diag}(\mathbf{A}_1, \ldots, \mathbf{A}_C). \tag{49}$$

For component c, let $\mathbf{B}_c$ contain its expression targets, and let $\mathbf{U}_c$ and $\mathbf{\Lambda}_c$ contain the updates and regularization coefficients for its embeddings. Since $\mathbf{\Lambda}$ is diagonal, the objective separates as

$$F(\mathbf{U}) = \sum_{c=1}^{C} \left(\|\mathbf{A}_c \mathbf{U}_c - \mathbf{B}_c\|_F^2 + \|\mathbf{\Lambda}_c^{1/2} \mathbf{U}_c\|_F^2 \right). \tag{50}$$

Each summand depends only on its own embedding updates. Applying Proposition 3 to each component gives

$$\mathbf{U}_c^\star = (\mathbf{A}_c^\top \mathbf{A}_c + \mathbf{\Lambda}_c)^{-1} \mathbf{A}_c^\top \mathbf{B}_c. \tag{51}$$

Stacking these solutions and restoring the original embedding order recovers the joint solution. An isolated expression contributes only a constant matching loss; an isolated embedding receives the zero update. Neither affects other components.

B Experimental Setup

This section describes the datasets, evaluation, baselines, and implementation details for the experiments in Section 4.

B.1 Datasets

The editing benchmarks test factual recall and the use of revised facts in reasoning. A separate set of tasks measures general capabilities after editing.

CounterFact. CounterFact (Meng et al., 2022) tests counterfactual changes to factual associations. We use the MultiCounterFact release provided with MEMIT for batched editing (Meng et al., 2023). Each case contains an edit request e_i and its prompt (see Section 2), held-out paraphrases, neighborhood prompts about other subjects whose facts should remain unchanged, and generation prompts. Held-out paraphrases are used only for evaluation; generated expressions are used during editing. Table 3 illustrates an edit request and its evaluation prompts.

ZsRE. ZsRE (Levy et al., 2017) represents factual associations as questions and answers. We use the editing evaluation split distributed with MEMIT (Meng et al., 2023), taking the first reference answer as the editing target. Each case pairs the edit question with a held-out paraphrase and an unrelated question–answer pair from Natural Questions. The edit therefore requests the supplied factual answer, not a counterfactual replacement. An example question, its target answer, and the evaluation questions are shown in Table 3.

MQuAKE. MQuAKE (Zhong et al., 2023) tests whether revised facts can be combined to answer multi-hop questions. We use 3,000 counterfactual cases covering two- to four-hop reasoning. Each case contains one or more fact-level edit requests, variants of a multi-hop question, and the answer implied by the revised facts, with accepted aliases. Table 3 shows how multiple fact edits change the answer to one multi-hop question.

General-Capability Tasks. Following AlphaEdit (Fang et al., 2025), we evaluate SST (SST-2) (Socher et al., 2013) for sentiment classification, MRPC (Dolan & Brockett, 2005) for paraphrase recognition, and CoLA (Warstadt et al., 2019) for linguistic acceptability. RTE (Bentivogli et al., 2009) and NLI (Wang et al., 2018) test textual entailment, and MMLU (Hendrycks et al., 2021) tests knowledge across multiple subjects. We use 100 examples per task; none are included in the edit requests.

B.2 Evaluation

On CounterFact and ZsRE, Efficacy measures success on edit prompts, Generalization on held-out paraphrases, and Specificity on unrelated queries. Utility is the arithmetic mean of these three metrics. Evaluation intervals are given in Appendix B.4.1.

Table 3: Editing examples from CounterFact, ZsRE, and MQuAKE, with simplified formatting. The ellipsis omits an unrelated prefix from the CounterFact paraphrase.

Field	Example
CounterFact	
Subject and relation	Danielle Darrieux; mother tongue
Requested change	French → English
Edit prompt	The mother tongue of Danielle Darrieux is
Held-out paraphrase	… Danielle Darrieux spoke the language
Unrelated prompt	The native language of Montesquieu is
Unchanged answer	French
ZsRE	
Subject	Watts Humphrey
Edit question	What university did Watts Humphrey attend?
Desired answer	Illinois Institute of Technology
Held-out paraphrase	What university did Watts Humphrey take part in?
Unrelated question	Who played Desmond Doss's father in Hacksaw Ridge?
Unchanged answer	Hugo Weaving
MQuAKE	
Edit request 1	Fernando Santos; country of citizenship: Portugal → United Kingdom
Edit request 2	United Kingdom; head of state: Elizabeth II → Emmerson Mnangagwa
Multi-hop question	Who is the head of state of the country where Fernando Santos hold a citizenship?
Original answer	Marcelo Rebelo de Sousa
Desired answer	Emmerson Mnangagwa

CounterFact. We compare the new and original answers using length-normalized log-likelihood. For prompt x, answer tokens $y = (y_1, \ldots, y_{|y|})$, and edited model $\mathcal{M}'$, the answer score is

$$\text{score}(x, y) = \frac{1}{|y|} \sum_{t=1}^{|y|} \log p_{\mathcal{M}'}(y_t \mid x, y_{<t}), \tag{52}$$

where $y_{<t}$ contains the preceding reference-answer tokens. For edit case i, the edit prompt x_i and held-out paraphrases succeed when the new answer $o_i^\star$ scores higher than the original answer o_i. Neighborhood prompts should retain the opposite preference. With $\mathbb{I}[\cdot]$ denoting the indicator function, the prompt-level scores are

$$\begin{aligned} s_i^{\text{edit}}(x) &= \mathbb{I}[\text{score}(x, o_i^\star) > \text{score}(x, o_i)], \\ s_i^{\text{nbr}}(x) &= \mathbb{I}[\text{score}(x, o_i) > \text{score}(x, o_i^\star)]. \end{aligned} \tag{53}$$

Efficacy averages $s_i^{\text{edit}}(x_i)$ over edit cases. Generalization first averages $s_i^{\text{edit}}(x)$ over each case's held-out paraphrases, then across cases. Specificity applies the same two-stage average to $s_i^{\text{nbr}}(x)$ on neighborhood prompts. Scores are multiplied by 100, with each case weighted equally.

For generated continuations, Fluency averages $\frac{1}{3}H_2 + \frac{2}{3}H_3$, where H_2 and H_3 are empirical word bigram and trigram entropies in bits. Consistency measures cosine similarity between TF-IDF vectors of the concatenated generated text and target-related reference text. We average scores across cases and multiply both metrics by 100 for reporting. Fluency is a scaled entropy, not a percentage accuracy.

ZsRE. Given the preceding reference-answer tokens, we check whether the model's most likely next token matches the next reference token. For prompt x and reference answer y, this gives

$$\text{TokenAcc}(x, y) = \frac{1}{|y|} \sum_{t=1}^{|y|} \mathbb{I}\left[\arg\max_v p_{\mathcal{M}'}(v \mid x, y_{<t}) = y_t\right], \tag{54}$$

where v ranges over the token vocabulary. Efficacy evaluates the edit question and its target answer; Generalization uses the held-out paraphrase and the same answer. Specificity evaluates the

unrelated question and its reference answer. Token accuracies are averaged within each case, then across cases, and multiplied by 100. Thus, Specificity measures post-edit answer accuracy, not agreement with pre-edit predictions. Newly correct and newly incorrect answers can offset each other; Appendix C.1.3 separates these transitions.

MQuAKE. Standard prompting asks for a direct answer, while CoT prompting asks for reasoning before the final answer. Both use the benchmark's demonstrations and greedy decoding, with maximum generation lengths of 32 and 128 tokens, respectively. We extract the first nonempty answer line for standard prompting, removing an optional answer label, and the final `Answer:` line for CoT. Predictions and reference answers are normalized for capitalization, whitespace, and boundary punctuation. A prediction is correct if it exactly matches the revised answer or an accepted alias. A case is correct if at least one of its question variants is answered correctly. We report the percentage of correct cases.

General Capabilities. For each task, we select the candidate answer with the highest score in Eq. 52 and compute class-frequency-weighted F1. For task d, let $\mathcal{C}_d$ be its label set, $n_{d,c}$ the number of examples with label c, and n_d the total number of examples. The task score and the mean across six tasks are

$$F_{1,d} = \sum_{c \in \mathcal{C}_d} \frac{n_{d,c}}{n_d} \frac{2\mathrm{TP}_{d,c}}{2\mathrm{TP}_{d,c} + \mathrm{FP}_{d,c} + \mathrm{FN}_{d,c}}, \qquad \overline{F}_1 = \frac{1}{6} \sum_{d=1}^{6} F_{1,d}. \tag{55}$$

Here, $\mathrm{TP}_{d,c}$, $\mathrm{FP}_{d,c}$, and $\mathrm{FN}_{d,c}$ count true positives, false positives, and false negatives for class c; a zero denominator contributes zero. Classes are weighted by frequency within each task, and tasks are weighted equally. We measure changes relative to each method's own pre-edit performance. These F1 scores follow our evaluation protocol, not each benchmark's default leaderboard metric.

Confidence Intervals. Across the editing benchmarks, reported intervals quantify uncertainty across cases, not across random seeds. Following ROME (Meng et al., 2022), we use normal-approximation 95% confidence intervals $\bar{x} \pm 1.96 s/\sqrt{N}$, where $\bar{x}$ and s are the mean and standard deviation of case-level scores. Scores from multiple prompts within a case are averaged first. The sample size N is the number of cases evaluated at each editing stage; the generated-expression analysis in Figure 5(b) uses $N = 2000$. MQuAKE instead uses Wilson 95% confidence intervals over cases. Utility is reported as a point estimate.

B.3 Baselines

We compare gradient-based expert editing, MoE-specific locate-then-edit, and direct conditional memory fine-tuning. AlphaEdit's original update rule applies to a dense FFN projection (Fang et al., 2025). We exclude it because applying that rule to routed experts requires an additional choice of experts and update allocation (Gu et al., 2026). The gradient-based baselines use the model's existing routed computation, with editable parameters specified below.

FT and FT-L. FT uses gradient descent to update selected model parameters so that edit prompts produce the desired answers. FT-L additionally bounds each parameter change to limit deviation from the model entering the edit batch (Zhu et al., 2020). In our MoE adaptation, both methods update expert FFN down-projections at layer 7. They use up to 25 epochs per edit batch, a learning rate of 5×10^{-4}, and optimization minibatches of one example. FT-L bounds each parameter change by 10^{-4}.

AdaLoRA. AdaLoRA (Zhang et al., 2023) learns low-rank updates to weight matrices while keeping the pretrained weights fixed. It allocates more of the rank budget to important update components and prunes less important ones. We apply it to expert down-projections at layer 7, optimizing the prediction loss on edit prompts with initial rank 12, target rank 8, and scaling parameter 32. It uses 25 epochs per edit batch, a learning rate of 5×10^{-4}, and optimization minibatches of one example.

UnKE. UnKE (Deng et al., 2025) edits knowledge by first computing target Transformer-block representations that lead to the desired answers. It then updates block parameters to match these

targets while retaining the outputs on preservation examples. In our MoE adaptation, we fit the layer-7 block outputs by updating the experts' gate, up-, and down-projections; attention and router parameters remain fixed. Target computation uses up to 25 steps at learning rate 0.5, and parameter fitting uses up to 50 steps at learning rate 2×10^{-4}.

MoEEdit. MoEEdit (Gu et al., 2026) jointly updates expert FFN down-projections so that their router-weighted outputs match the editing targets. To preserve outputs on unrelated inputs and limit downstream routing shifts, it constrains each expert's update using a null-space projection built from preservation inputs. The updates are solved one expert at a time through block coordinate descent. We edit layer 7, using up to 25 target-computation steps at learning rate 0.1 and four block coordinate descent passes. The per-expert null-space projections use statistics from 100,000 Wikipedia samples.

Memory-FT (Subject). This baseline directly fine-tunes conditional memory to predict the desired answers, restricting updates to embeddings activated at the last subject-token position of each original edit prompt. Since this selects a limited set of n-grams, we store one cumulative update vector per exact n-gram, as in EngramEdit. We optimize these vectors by backpropagating the prediction loss and add them whenever their n-grams are activated, including outside subject positions. The pretrained hashed tables, token embeddings, memory projection layers, and backbone remain fixed.

Memory-FT (All Tokens). This variant expands direct memory fine-tuning to embeddings activated at every token position in the original edit prompts, using the same prediction loss. Storing a separate update vector for each n-gram in this larger set would increase memory requirements. We therefore directly fine-tune the pretrained hashed tables, allowing rows activated at any token position to receive gradient updates while keeping all other parameters fixed. Both Memory-FT variants use n-gram lengths 2, 3, and 4, up to 25 epochs per edit batch, a learning rate of 3×10^{-3}, and optimization minibatches of one example.

B.4 Implementation Details

We describe the shared model and editing setup, followed by EngramEdit's target computation, memory mapping, and memory updating settings.

B.4.1 Model and Editing Setup

All experiments use LongCat-Flash-Lite, a 68.5B-parameter MoE model with 31.4B parameters in n-gram embeddings (Liu et al., 2026), and run on two NVIDIA RTX PRO 6000 GPUs. Its conditional memory uses suffix n-grams of lengths 2, 3, and 4, with four hashed sub-tables per length. Each sub-table returns a 256-dimensional vector, which is linearly projected to the 3,072-dimensional token embedding space. At each position, the model averages the twelve projected vectors and the original token embedding before passing the result to the backbone. EngramEdit keeps the token embeddings and these projection matrices fixed.

Sequential Editing. An edit batch contains the requests applied together; an optimization minibatch contains the examples used in one gradient step. The main CounterFact and ZsRE experiments apply 2,000 edits in batches of 100, with each batch starting from the model state produced by preceding batches. The longer runs extend editing to 5,000 facts. For MQuAKE, we shuffle the 3,000 cases once with seed 0 and use the same case IDs and hop labels for all methods. Each batch contains 100 cases, with all their fact-level edit requests applied together, so a batch can contain more than 100 fact edits.

Evaluation Schedule. CounterFact trajectories are evaluated every 100 edits, and ZsRE trajectories are compared at common intervals of 200 edits. General capabilities are evaluated before editing and every 500 edits through 5,000 sequential CounterFact edits, using 100 examples per task (see Appendix B.1).

B.4.2 Target Computation and Memory Mapping

For each edit request, we ask the model to generate four semantically equivalent expressions before editing, using the prompt and filtering rules in Appendix A.2.1. The retained expressions and original edit prompt form the expression set used for target computation and memory mapping.

Prediction Loss. During target computation, we evaluate each expression both alone and after each of five model-generated prefix contexts, which vary the surrounding text without rephrasing the edit request. The shared perturbation is added to the conditional memory representation before the first decoder block, while all decoder parameters remain fixed. Following the target readout in MEMIT (Meng et al., 2023), we compute the NLL from the last decoder block's hidden states using the model's fixed final normalization and language-model head. This yields the final predictive distribution in Eq. 20; the KL preservation term also uses final-output probabilities.

Optimization Settings. We optimize the shared perturbation with Adam for at most 25 steps at learning rate 0.5, with $\lambda_{\mathrm{KL}} = 0.0625$, $\lambda_{\mathrm{norm}} = 0.001$, and $\rho = 32$. Optimization stops when the total target-computation loss falls below 0.05, checked every two steps.

Memory Mapping. We use expressions without prefix contexts for memory mapping. At each expression's last subject-token position, we extract suffix n-grams of lengths 2, 3, and 4, skipping incomplete windows and windows crossing an end-of-sequence token.

B.4.3 Memory Updating

Conditional memory uses hashed tables to control the storage cost of a large n-gram space. Editing only requires storing updates for the selected n-grams. We therefore maintain a separate cumulative update vector for each edited n-gram, indexed by its exact token sequence. Additional storage scales with the number of distinct edited n-grams, and the exact indexing prevents hash collisions from coupling their updates. Each vector is added to the embedding output whenever its n-gram is activated, while the pretrained tables remain fixed.

Regularization and Solver Settings. We use the weight construction in Eq. 29, setting $\lambda_{\mathrm{ridge}} = 0.01$ and $\lambda_{\mathrm{reuse}} = 0.05$. The length weights are $\eta_2 = 8$, $\eta_3 = 2$, and $\eta_4 = 1$. For frequency-based regularization, we use $\gamma = 9$, $\beta = 1$, $w_{\mathrm{freq}}^{\max} = 10$, and $w^{\max} = 64$. We estimate the corpus frequency of the selected n-grams from up to 3,000,000 documents in the November 1, 2023 English Wikipedia snapshot, using at most 4,096 tokens from each document. Within each n-gram length, the percentile π_j is the normalized rank of the observed count among the distinct count values. Missing n-grams receive a frequency weight of 1. We form and solve the joint linear system in FP32 using a direct linear solver.

C Additional Experimental Results

This section provides detailed evaluation results and analyses of editing choices and memory use to complement Section 4.

C.1 Detailed Evaluation Results

We extend the sequential editing results, examine multi-hop reasoning by depth, and analyze ZsRE Specificity to complement Section 4.2. We then report task-level general-capability trajectories to complement Section 4.3.

C.1.1 Sequential Editing with 5,000 Facts

To test whether knowledge updates through conditional memory remain effective as edits accumulate, we extend the CounterFact and ZsRE experiments in Section 4.2 to 5,000 sequential edits in batches of 100. We compare EngramEdit with FT, UnKE, MoEEdit, and MFT-S. Each evaluation covers all edits applied so far. Figure 7 reports Efficacy, Generalization, and Specificity every 100 edits on CounterFact and every 200 edits on ZsRE. We observe that:

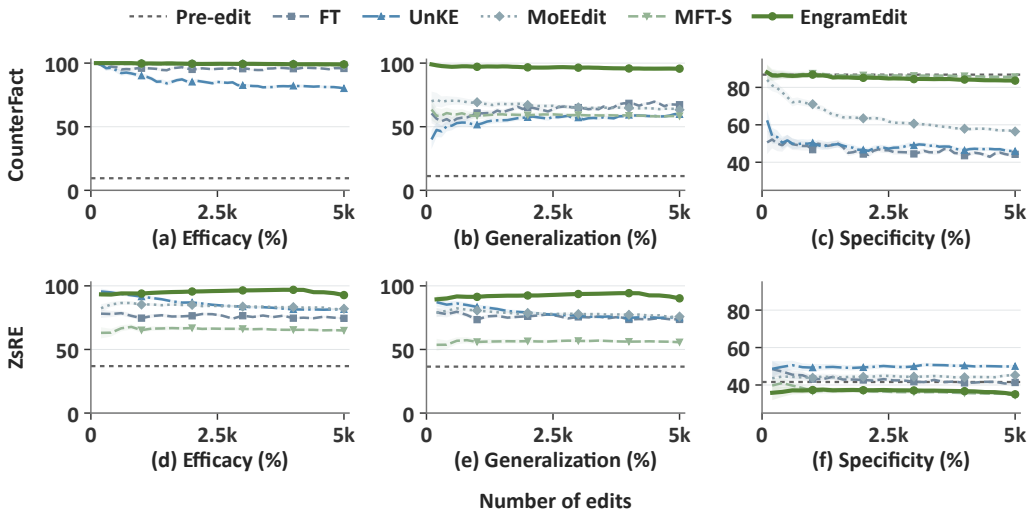


Figure 7: Sequential editing on CounterFact (a–c) and ZsRE (d–f) up to 5,000 edits. Shading shows pointwise case-level 95% confidence intervals. MFT-S denotes Memory-FT (Subject).

- EngramEdit maintains high Efficacy and a clear Generalization lead on both datasets through 5,000 edits. Because each evaluation includes all earlier edits, these results show that revised knowledge remains accessible across expressions as further updates accumulate. EngramEdit supports this access by jointly updating embeddings across expressions, allowing different wordings to activate memory updated for the same fact.
- On CounterFact, EngramEdit and MFT-S maintain much higher Specificity than FT, UnKE, and MoEEdit throughout the sequence, but EngramEdit achieves substantially higher Generalization at similar Efficacy. EngramEdit therefore combines cross-expression access with largely preserved unrelated knowledge as edits accumulate. Reuse-based regularization can help explain this preservation because it penalizes large updates to embeddings that unrelated inputs are more likely to activate.

C.1.2 Multi-Hop Reasoning

To test whether knowledge updated through conditional memory supports reasoning at different depths, we break down the MQuAKE evaluation in Section 4.2 by hop count. All eight methods use the same 3,000 cases: 1,135 two-hop, 1,136 three-hop, and 729 four-hop cases. We compare standard prompting, which asks for a direct answer, with CoT prompting, which asks for step-by-step reasoning before answering. Table 4 reports accuracy within each group. We observe that:

- EngramEdit gains substantially from CoT at all three depths, whereas FT-L, AdaLoRA, and MoEEdit lose accuracy in every group. This contrast shows that the benefit of CoT depends on the editing method, not just the reasoning prompt. For EngramEdit, CoT can help because intermediate steps explicitly express facts whose n-grams activate updated embeddings, allowing revised information to guide subsequent reasoning.
- The best method varies across hop groups under standard prompting, but EngramEdit leads every group with CoT. Its advantage persists through four-hop questions, despite lower accuracy and smaller margins at greater depths. This supports the use of revised knowledge in longer reasoning chains, which require more facts to be accessed and combined.

C.1.3 Analysis of ZsRE Specificity

To understand whether higher ZsRE Specificity reflects better preservation, we compare correctness on unrelated queries before and after 2,000 edits. All seven methods are evaluated on the same 11,379 neighborhood prompts from the first 2,000 edit cases. We compare their recorded post-edit correctness labels against one shared evaluation of unedited LongCat. EngramEdit's results come from the first 2,000 edits of the 5,000-edit run, separate from the main-table run.

Table 4: MQuAKE accuracy (%) by hop with standard and CoT prompts. Brackets give Wilson 95% confidence intervals. Bold and underlining mark the best and second-best scores.

Method	Standard ↑			CoT ↑		
	2-hop	3-hop	4-hop	2-hop	3-hop	4-hop
FT	<u>7.93</u> [6.50, 9.65]	2.11 [1.42, 3.12]	**7.82** [6.08, 10.00]	<u>12.60</u> [10.79, 14.66]	3.61 [2.67, 4.86]	<u>9.74</u> [7.79, 12.11]
FT-L	1.59 [1.01, 2.49]	1.67 [1.07, 2.60]	4.53 [3.24, 6.29]	1.06 [0.61, 1.84]	0.97 [0.54, 1.73]	1.78 [1.05, 3.03]
AdaLoRA	1.50 [0.94, 2.39]	2.64 [1.86, 3.74]	3.84 [2.67, 5.50]	1.15 [0.67, 1.95]	1.23 [0.74, 2.06]	2.33 [1.46, 3.70]
UnKE	4.14 [3.13, 5.46]	**4.23** [3.20, 5.56]	<u>6.17</u> [4.65, 8.16]	5.46 [4.28, 6.94]	<u>4.58</u> [3.51, 5.95]	9.60 [7.67, 11.96]
MoEEdit	3.00 [2.15, 4.16]	2.64 [1.86, 3.74]	3.29 [2.22, 4.85]	0.62 [0.30, 1.27]	0.26 [0.09, 0.77]	0.27 [0.08, 0.99]
MFT-S	6.61 [5.30, 8.20]	2.55 [1.78, 3.64]	5.49 [4.06, 7.39]	8.28 [6.82, 10.03]	3.61 [2.67, 4.86]	3.98 [2.78, 5.65]
MFT-A	5.02 [3.90, 6.45]	1.32 [0.80, 2.17]	5.62 [4.17, 7.54]	4.32 [3.28, 5.66]	2.02 [1.35, 3.02]	3.57 [2.45, 5.17]
EngramEdit	**9.52** [7.94, 11.36]	<u>2.82</u> [2.00, 3.95]	5.49 [4.06, 7.39]	**36.92** [34.16, 39.76]	**20.07** [17.84, 22.50]	**15.09** [12.67, 17.87]

Table 5: ZsRE correctness transitions after 2,000 edits. C and W denote correct and wrong predictions; Post–Pre is the Specificity change in percentage points. Stable Corr. measures unchanged correctness states. Values are case-averaged percentages with 95% CI half-widths in subscripts.

Method	C → W	W → C	Post–Pre	Post Spec. ↑	Stable Corr. ↑
FT-L	$0.62_{\pm 0.19}$	$4.84_{\pm 0.57}$	+4.23	$45.76_{\pm 1.24}$	$94.54_{\pm 0.60}$
AdaLoRA	$2.25_{\pm 0.33}$	$7.51_{\pm 0.71}$	+5.26	$46.79_{\pm 1.27}$	$90.24_{\pm 0.79}$
UnKE	$2.95_{\pm 0.36}$	$11.26_{\pm 0.86}$	+8.31	$49.85_{\pm 1.32}$	$85.79_{\pm 0.93}$
MoEEdit	$2.30_{\pm 0.35}$	$5.14_{\pm 0.56}$	+2.84	$44.37_{\pm 1.26}$	$92.56_{\pm 0.67}$
MFT-S	$5.06_{\pm 0.51}$	$0.40_{\pm 0.12}$	-4.67	$36.87_{\pm 1.17}$	$94.54_{\pm 0.54}$
MFT-A	$9.67_{\pm 0.72}$	$1.84_{\pm 0.30}$	-7.83	$33.71_{\pm 1.13}$	$88.48_{\pm 0.80}$
EngramEdit	$5.25_{\pm 0.54}$	$0.92_{\pm 0.25}$	-4.33	$37.21_{\pm 1.18}$	$93.83_{\pm 0.60}$

Let C and W denote correct and wrong predictions. For each transition $a \to b$, we compute its proportion over all prompts within each case, then average equally across cases to obtain $P(a \to b)$. Post-edit Specificity counts both retained correct predictions ($\mathrm{C} \to \mathrm{C}$) and newly correct predictions ($\mathrm{W} \to \mathrm{C}$). We report the change in Specificity and Stable Correctness, the proportion of unchanged correctness states:

$$\begin{aligned} \Delta\text{Specificity} &= P(\mathrm{W} \to \mathrm{C}) - P(\mathrm{C} \to \mathrm{W}), \\ \text{Stable Correctness} &= 1 - P(\mathrm{C} \to \mathrm{W}) - P(\mathrm{W} \to \mathrm{C}). \end{aligned} \tag{56}$$

Stable Correctness includes predictions that remain wrong, so it differs from retention of previously correct predictions. It also does not imply identical outputs, since one wrong token can change to another. We report 95% confidence-interval half-widths of $1.95996 s/\sqrt{2000}$, where s is the sample standard deviation of the case-level proportions. Table 5 summarizes the results. We observe that:

- FT-L, AdaLoRA, UnKE, and MoEEdit exceed pre-edit Specificity because their $\mathrm{W} \to \mathrm{C}$ rates exceed their $\mathrm{C} \to \mathrm{W}$ rates. Their higher post-edit accuracy than EngramEdit reflects both better retention of initially correct predictions and more corrections of initially wrong predictions.
- EngramEdit preserves 93.83% of correctness states and about 87.4% of initially correct predictions under case-weighted aggregation. Its $\mathrm{W} \to \mathrm{C}$ rate of only 0.92% shows that its post-edit Specificity comes predominantly from retained correct predictions. Together with the high Efficacy and Generalization reported in Section 4.2, these results show that EngramEdit supports effective, generalizable knowledge updates while retaining most previously correct predictions on unrelated queries.

C.1.4 Task-Level General Capabilities

To assess capability preservation across individual tasks as edits accumulate, we track six task-level F1 scores during 5,000 sequential CounterFact edits in batches of 100. We compare EngramEdit

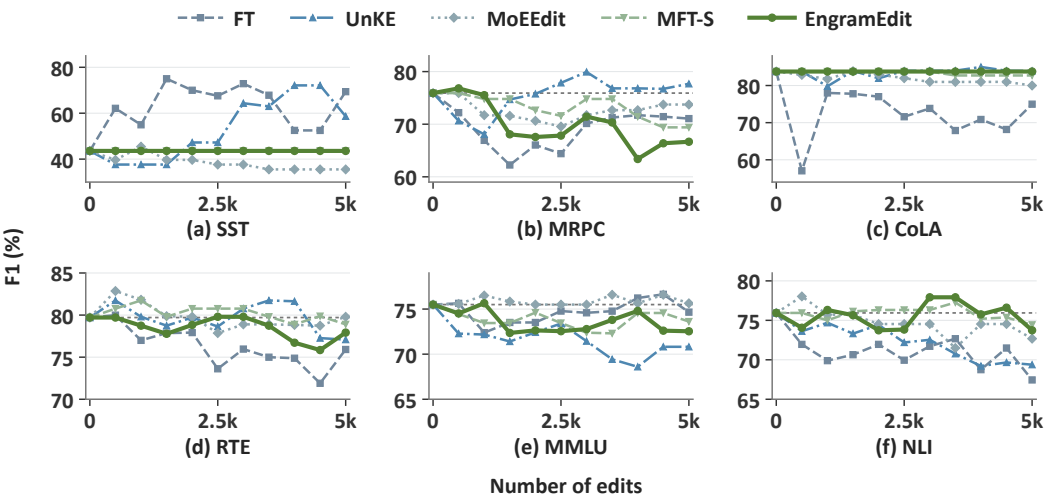


Figure 8: Task-level general capabilities during 5,000 sequential CounterFact edits. F1 is measured on 100 examples per task. Dotted lines mark pre-edit scores.

with FT, UnKE, MoEEdit, and MFT-S on 100 examples per task, evaluating before editing and every 500 edits. Figure 8 shows each method's class-frequency-weighted F1 trajectory, with pre-edit scores as references. We observe that:

- EngramEdit preserves pre-edit F1 on SST and CoLA throughout editing, while RTE, MMLU, and NLI stay within four F1 points of their pre-edit scores. By comparison, FT and UnKE show large SST gains and NLI losses. These trajectories show that EngramEdit largely preserves performance across five tasks throughout sequential editing. This stability can be partly explained by restricting edits to selected embeddings and applying stronger penalties to frequently reused ones, which limits changes to memory shared with unrelated inputs.
- EngramEdit's larger changes are concentrated on MRPC, and MFT-S shows a similar pattern. This suggests that MRPC is more sensitive to memory updates than the other evaluated tasks. MRPC requires comparing the meanings of two sentences, so updates that affect their representations differently may change the equivalence judgment. Such differences can arise when the sentences activate different n-grams.

C.2 Analysis of Editing Choices

To examine how editing choices affect knowledge updating and preservation, we first compare editors with and without generated expressions. We then vary EngramEdit's expression count, editable n-gram lengths, regularization coefficients, and clamp factor. These studies complement the component analyses in Section 4.4.

C.2.1 Multi-Expression Editing

To examine how generated expressions affect different editors, we compare MoEEdit, MFT-S, MFT-A, and EngramEdit with zero or four generated expressions per fact on CounterFact and ZsRE. Each setting uses 2,000 sequential edits in batches of 100 and retains the original edit prompt. All methods use the same generated expressions in the four-expression setting. From Table 6, we observe that:

- Adding four generated expressions improves Generalization for MFT-S, MFT-A, and EngramEdit on both datasets, whereas MoEEdit's Efficacy and Generalization decrease. The benefit therefore depends on how each editor uses the expressions. Generated expressions can help memory-only editors because their different wordings activate additional n-grams, allowing held-out paraphrases to access revised facts through a larger set of updated embeddings.
- EngramEdit leads in Efficacy, Generalization, and Utility on both datasets with either expression count. On CounterFact, it generalizes better than MFT-S at similar Efficacy and Specificity when both use the same expressions and memory update interface. On ZsRE, its stronger Efficacy and Generalization yield higher Utility than MoEEdit despite lower Specificity. EngramEdit thus uses the same expressions more effectively for knowledge updating. By computing targets across

Table 6: Editing with and without generated expressions on CounterFact and ZsRE over 2,000 sequential edits. Subscripts give 95% CI half-widths. Bold and underlining mark the best and second-best scores within each block.

Method	CounterFact				ZsRE			
	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑
Original edit prompt only								
MoEEdit	99.1$_{\pm 0.41}$	67.1$_{\pm 1.69}$	63.5$_{\pm 1.28}$	76.6	85.1$_{\pm 0.91}$	78.7$_{\pm 1.21}$	**44.4**$_{\pm 1.26}$	69.4
MFT-S	99.0$_{\pm 0.45}$	59.1$_{\pm 1.92}$	85.2$_{\pm 0.90}$	81.1	66.9$_{\pm 1.35}$	56.5$_{\pm 1.53}$	36.9$_{\pm 1.17}$	53.4
MFT-A	93.9$_{\pm 1.04}$	40.7$_{\pm 1.75}$	66.6$_{\pm 1.12}$	67.1	62.8$_{\pm 1.49}$	51.3$_{\pm 1.57}$	33.7$_{\pm 1.13}$	49.3
EngramEdit	**99.5**$_{\pm 0.32}$	**75.0**$_{\pm 1.74}$	**85.6**$_{\pm 0.91}$	**86.7**	**95.1**$_{\pm 0.62}$	**83.4**$_{\pm 1.29}$	38.5$_{\pm 1.18}$	**72.3**
Original edit prompt + 4 generated expressions								
MoEEdit	80.7$_{\pm 1.73}$	59.8$_{\pm 1.71}$	66.0$_{\pm 1.21}$	68.8	79.8$_{\pm 1.12}$	72.9$_{\pm 1.35}$	**42.8**$_{\pm 1.23}$	65.2
MFT-S	99.0$_{\pm 0.45}$	90.7$_{\pm 1.09}$	85.0$_{\pm 0.91}$	91.5	67.2$_{\pm 1.37}$	64.2$_{\pm 1.43}$	37.7$_{\pm 1.17}$	56.4
MFT-A	93.8$_{\pm 1.06}$	63.2$_{\pm 1.72}$	63.6$_{\pm 1.20}$	73.5	69.8$_{\pm 1.31}$	65.9$_{\pm 1.43}$	31.9$_{\pm 1.11}$	55.9
EngramEdit	**99.5**$_{\pm 0.31}$	**97.0**$_{\pm 0.69}$	**85.2**$_{\pm 0.92}$	**93.9**	**97.3**$_{\pm 0.40}$	**93.7**$_{\pm 0.79}$	38.3$_{\pm 1.18}$	**76.4**

Table 7: Results with different numbers of generated expressions after 2,000 edits. # Expr. excludes the original prompt. Subscripts give reported CI half-widths; bold and underlining mark best and second-best scores.

# Expr.	CounterFact						ZsRE			
	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑	Flu. ↑	Cons. ↑	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑
0	**99.5**$_{\pm 0.32}$	75.0$_{\pm 1.74}$	**85.6**$_{\pm 0.91}$	86.7	562.6$_{\pm 1.19}$	15.7$_{\pm 0.36}$	95.1$_{\pm 0.62}$	83.4$_{\pm 1.29}$	**38.5**$_{\pm 1.18}$	72.3
2	99.4$_{\pm 0.34}$	95.2$_{\pm 0.83}$	85.3$_{\pm 0.92}$	93.3	566.2$_{\pm 1.11}$	**16.2**$_{\pm 0.36}$	96.2$_{\pm 0.50}$	91.8$_{\pm 0.89}$	37.7$_{\pm 1.10}$	75.2
4	**99.5**$_{\pm 0.31}$	97.0$_{\pm 0.69}$	85.2$_{\pm 0.92}$	93.9	565.2$_{\pm 1.10}$	16.1$_{\pm 0.36}$	**97.3**$_{\pm 0.40}$	93.7$_{\pm 0.79}$	38.3$_{\pm 1.18}$	**76.4**
6	99.4$_{\pm 0.34}$	**97.3**$_{\pm 0.64}$	85.2$_{\pm 0.92}$	**94.0**	**567.5**$_{\pm 1.03}$	**16.2**$_{\pm 0.36}$	96.5$_{\pm 0.47}$	93.7$_{\pm 0.77}$	36.8$_{\pm 1.17}$	75.7
8	99.1$_{\pm 0.41}$	97.2$_{\pm 0.67}$	84.9$_{\pm 0.93}$	93.7	566.0$_{\pm 1.07}$	16.0$_{\pm 0.36}$	96.8$_{\pm 0.44}$	**94.2**$_{\pm 0.74}$	38.0$_{\pm 1.17}$	76.3

expressions and matching them jointly, it accounts for the requirements of all expressions sharing each embedding. Reuse-based regularization further limits large changes to embeddings also used by unrelated inputs.

C.2.2 Number of Generated Expressions

To assess the benefit of generating more expressions per edit, we compare $K \in \{0, 2, 4, 6, 8\}$ generated expressions on CounterFact and ZsRE. Each setting uses 2,000 sequential edits in batches of 100. The original edit prompt is always retained, and all other settings are fixed. From Table 7, we observe that:

- Two generated expressions provide most of the Generalization gain on both datasets, with further improvements at four. Efficacy remains high, while Specificity on both datasets and Fluency and Consistency on CounterFact vary little. A small expression set thus supports broader access to revised knowledge with largely preserved unrelated-query performance. Generated expressions enable this access by adding the n-grams activated by different wordings to memory mapping, so held-out paraphrases have more opportunities to activate updated embeddings.
- Beyond four expressions, Generalization improves only slightly and Utility does not consistently increase. Four expressions achieve the highest Utility on ZsRE and nearly the highest on CounterFact. This supports $K = 4$ as the default, since larger sets require more expression processing without comparable performance gains.

C.2.3 Editable n-gram Lengths

To identify which n-gram lengths support both effective updates and preservation, we compare eight editable-length configurations on CounterFact using 2,000 sequential edits in batches of 100. We

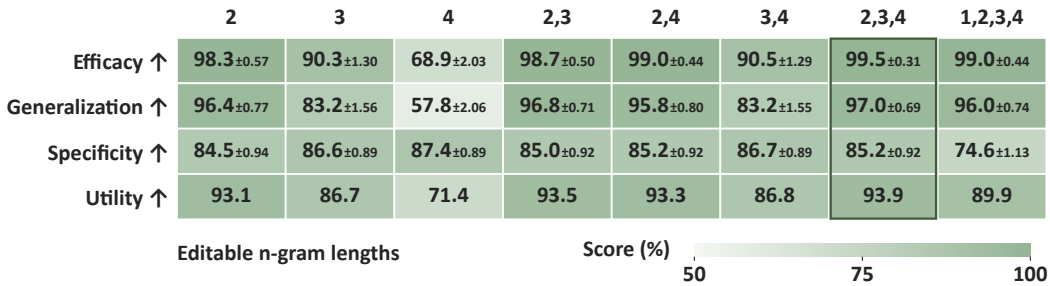


Figure 9: Editable n-gram lengths on CounterFact. The frame marks the default lengths 2, 3, and 4; $n = 1$ denotes single-token embedding updates. Small $\pm$ values give CI half-widths.

Table 8: Sensitivity to regularization and clamp factor after 2,000 CounterFact edits. Only the listed parameter changes. Scores are percentages; subscripts give reported CI half-widths. Bold parameter values mark the defaults.

Parameter	Value	Eff. ↑	Gen. ↑	Spec. ↑	Util. ↑
Reuse-based regularization λ_{reuse}	0	$98.8_{\pm 0.49}$	$96.5_{\pm 0.76}$	$84.7_{\pm 0.93}$	93.3
	0.03	$99.3_{\pm 0.37}$	$96.2_{\pm 0.76}$	$85.2_{\pm 0.92}$	93.6
	0.05	$99.5_{\pm 0.31}$	$97.0_{\pm 0.69}$	$85.2_{\pm 0.92}$	93.9
	0.075	$99.6_{\pm 0.29}$	$96.4_{\pm 0.73}$	$85.3_{\pm 0.91}$	93.8
	0.1	$99.6_{\pm 0.28}$	$96.6_{\pm 0.70}$	$85.2_{\pm 0.92}$	93.8
Ridge regularization λ_{ridge}	0	$99.5_{\pm 0.31}$	$96.6_{\pm 0.71}$	$85.1_{\pm 0.92}$	93.7
	0.001	$99.6_{\pm 0.29}$	$96.6_{\pm 0.72}$	$85.3_{\pm 0.92}$	93.8
	0.01	$99.5_{\pm 0.31}$	$97.0_{\pm 0.69}$	$85.2_{\pm 0.92}$	93.9
	0.03	$99.4_{\pm 0.35}$	$96.8_{\pm 0.71}$	$85.3_{\pm 0.92}$	93.8
	0.1	$99.1_{\pm 0.41}$	$96.6_{\pm 0.73}$	$84.8_{\pm 0.93}$	93.5
Clamp factor ρ	8	$99.1_{\pm 0.41}$	$97.0_{\pm 0.69}$	$85.2_{\pm 0.92}$	93.8
	16	$99.4_{\pm 0.35}$	$96.7_{\pm 0.71}$	$85.3_{\pm 0.92}$	93.8
	32	$99.5_{\pm 0.31}$	$97.0_{\pm 0.69}$	$85.2_{\pm 0.92}$	93.9
	64	$99.6_{\pm 0.28}$	$96.4_{\pm 0.73}$	$85.0_{\pm 0.93}$	93.7

test lengths 2, 3, and 4 individually, in pairs, and together, as well as the full combination with single-token ($n = 1$) embedding updates. All settings use four generated expressions and reuse-based regularization; only the lengths eligible for embedding updates change. From Figure 9, we observe that:

- Among single-length settings, 2-grams achieve the highest Efficacy and Generalization, while longer n-grams give higher Specificity. Shorter n-grams therefore favor access across expressions, whereas longer ones favor preservation. This is because shorter token sequences can activate the same updated embedding across both paraphrases and unrelated inputs.
- Combining lengths 2, 3, and 4 achieves the highest Efficacy, Generalization, and Utility, with higher Specificity than using 2-grams alone. This suggests that different lengths can complement each other. Shorter n-grams support access across expressions, while longer ones provide more context-specific embeddings that can carry part of the update, reducing reliance on embeddings shared with unrelated inputs.
- Excluding single-token embedding updates yields markedly higher Specificity than editing lengths 1–4, with no loss in Efficacy or Generalization. This supports restricting the default editable lengths to 2–4. Individual tokens occur in many unrelated inputs, so single-token updates can affect a wider range of predictions without providing additional editing gains here.

C.2.4 Regularization and Clamp Factor

To test whether strong editing performance requires narrowly tuned regularization or perturbation bounds, we vary one parameter at a time on CounterFact using 2,000 sequential edits in batches of 100. The reuse coefficient λ_{reuse} scales the length- and frequency-based penalties, while the

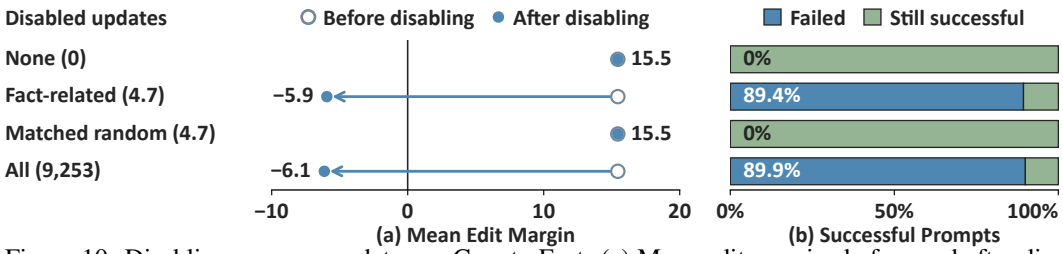


Figure 10: Disabling memory updates on CounterFact. (a) Mean edit margins before and after disabling. Positive values favor revised answers; negative values favor original answers. (b) Outcomes of the 5,850 prompts successful before disabling. Parentheses give the mean number of disabled updates per case.

ridge coefficient λ_{ridge} penalizes all embedding updates equally. The clamp factor ρ bounds the perturbation norm during target computation. Table 8 lists the tested values and marks the defaults. Other settings remain fixed, including four generated expressions and editable lengths 2, 3, and 4. We observe that:

- All tested nonzero reuse coefficients yield higher Efficacy, Specificity, and Utility than disabling the reuse-based term, with little variation among them. The gains therefore do not depend on one precisely tuned strength. Changing the coefficient scales the reuse-based penalties but retains stronger constraints on frequently reused embeddings, which can help preserve unrelated knowledge across these settings.
- All four metrics remain close across ridge coefficients, including zero. This shows that strong performance does not require a precisely tuned uniform penalty. Reuse-based regularization still constrains embedding updates when ridge is removed, so the extra term mainly provides another way to control overall update size.
- Efficacy, Generalization, and Specificity remain stable as the clamp factor increases from 8 to 64, with no consistent Utility gain from larger values. Strong editing performance thus holds across the tested norm bounds. A larger factor permits, but does not force, a larger perturbation, since prediction loss and target regularization still guide its optimization.

C.3 Analysis of Memory Use

To understand how revised facts are accessed through conditional memory, we first test whether successful recall depends on fact-related updates by selectively disabling them, extending Section 4.4. With all updates enabled, we next group inputs by the updated embeddings they activate, then analyze how cross-edit sharing relates to editing outcomes as updates accumulate. These later analyses leave the edited model unchanged and identify associations, while disabling directly tests the effect of removing updates.

C.3.1 Disabling Memory Updates

To test whether revised-fact recall depends on fact-related memory updates, we disable selected updates during inference. We use the model from Figure 5(c), with 9,253 updated embeddings from 2,000 sequential CounterFact edits. Fact-related updates correspond to n-grams selected at the last subject token of each fact's original and four generated expressions. Four conditions disable 1) no updates (*None*); 2) the evaluated fact's updates (*Fact-related*); 3) other updates matched in number and n-gram length (*Matched random*); or 4) all updates (*All*). The random control (seed 0) excludes fact-related updates but does not match update norms or corpus frequencies. Original embeddings remain active, and updates are restored after each evaluation without further editing. Disabling a shared embedding's update removes the combined changes from all associated edits. Differences between All and separately recorded pre-edit outputs may reflect finite-precision variation. For each prompt x and edit e_i, the edit margin measures revised-answer preference:

$$\text{margin}_i(x) = \text{score}(x, o_i^\star) - \text{score}(x, o_i), \tag{57}$$

using length-normalized log-likelihood scores. Positive margins favor revised answers and negative margins original answers. We aggregate margins from edit prompts and held-out paraphrases within

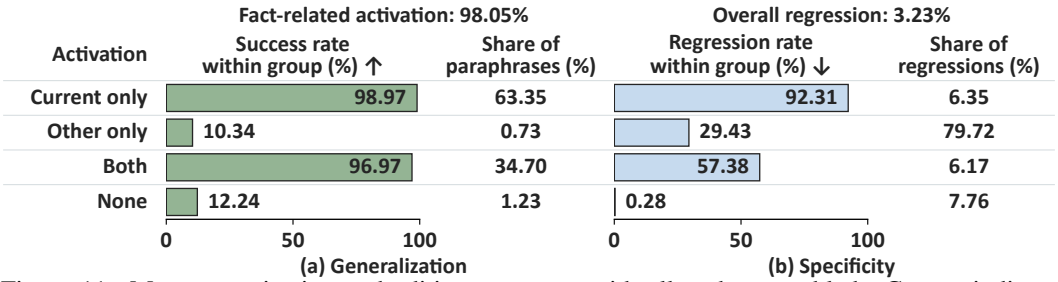


Figure 11: Memory activation and editing outcomes with all updates enabled. Groups indicate activation of embeddings updated for the current edit only, other edits only, both, or neither. (a) Within-group paraphrase success rates and each group's share of all paraphrases. (b) Success-to-failure rates among previously successful neighborhood queries in each group and each group's share of all regressions.

each case, then average cases equally. Of the 2,000 edit prompts and 4,000 held-out paraphrases, 5,850 succeed under None. The success-to-failure rate reports the percentage of these same prompts that no longer favor the revised answer after disabling. From Figure 10, we observe that:

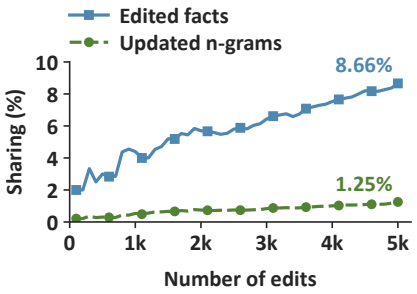
- Fact-related disabling makes the model favor original over revised answers on average and turns 5,228 of the 5,850 successful prompts into failures. These updates therefore support revised-fact recall for most previously successful prompts. Since the original embeddings remain active, the reversal reflects removal of the learned updates, not removal of the underlying memory.
- Matched random disabling changes the mean margin by only -0.0003 and causes no successful prompt to fail. The contrast shows that recall depends on which updates are disabled, not simply how many. Fact-related updates remain available under this control, allowing the model to continue using them to recall revised facts.
- Disabling an average of 4.679 fact-related updates nearly matches disabling all 9,253 updates in both mean margin and success-to-failure rate. This shows that recalling revised facts depends primarily on a small set of memory updates, activated when the input contains the corresponding fact-related n-grams.

C.3.2 Memory Activation and Editing Outcomes

To relate memory activation to generalization and preservation, we evaluate 4,000 held-out paraphrases and 20,000 neighborhood prompts after completing 2,000 sequential CounterFact edits, with all updates enabled. The *current edit* is the edit paired with each prompt in CounterFact; neighborhood prompts ask about unrelated facts. Using the fact-related n-gram sets from Appendix C.3.1, we scan all prompt positions before answer generation and group prompts by the updated embeddings they activate: 1) *Current only*, those associated only with the current edit; 2) *Other only*, only with other edits; 3) *Both*, with both; and 4) *None*, no updated embeddings. An embedding shared across edits counts toward each associated edit. The sets cover all 9,253 updated n-grams, and including teacher-forced answer prefixes leaves the groups unchanged.

Generalization. A held-out paraphrase succeeds when the revised answer scores above the original answer under length-normalized log-likelihood. The held-out paraphrases do not overlap with generated expressions under exact or normalized matching. Figure 11(a) shows the success rate within each group and the group's share of all 4,000 paraphrases. We observe that:

- Current only and Both together cover 98.05% of held-out paraphrases, with success rates of 96.97–98.97%. High success thus extends to most held-out paraphrases and is associated with activation of fact-related updates. Different wordings can share n-grams, so expressions not used during editing can still use embeddings updated for the same fact.
- Other only and None contain just 1.95% of held-out paraphrases but account for 50.36% of failures. This concentration of failures supports broader memory mapping across expressions, as tested in Section 4.4. These paraphrases cannot directly use the updates for their fact because none of the corresponding n-grams are activated.



(a) Cross-Edit Sharing

Group	Facts n (%)	Eff. ↑ (%)	Gen. ↑ (%)	Spec. ↑ (%)	Eff. failures n (%)
No sharing	4,567 (91.34)	99.98	96.7	83.7	1 (2.04)
Sharing	433 (8.66)	88.91	84.1	82.4	48 (97.96)

Facts: % of 5,000; failures: % of 49

(b) Editing Outcomes

Figure 12: Cross-edit sharing on CounterFact. (a) Sharing rates among updated n-grams and edited facts. (b) Final editing results for facts with or without shared n-grams.

Specificity. On neighborhood prompts, success requires the original answer to score above the edit's revised answer. We compare the same prompts before and after editing and count initially successful prompts that become failures as *regressions*. Figure 11(b) shows the regression rate among initially successful prompts in each group and the group's share of all 567 regressions. We observe that:

- Among the 17,534 initially successful prompts, 96.77% remain successful and 3.23% regress; the regression rate in None is only 0.28%. Most previously correct predictions are thus retained, with especially strong preservation in None because these inputs do not directly use embedding updates. The 44 regressions in None may reflect finite-precision differences that reverse close answer rankings across inference runs.
- Other only accounts for 79.72% of all regressions, whereas Current only accounts for 6.35% despite its higher within-group regression rate. Overall, 92.24% of regressions accompany activation of updated embeddings. Regressions are thus concentrated in inputs that activate updates for other edits. Different facts can share n-grams, so an update for one fact can also affect unrelated predictions. This supports using reuse-based regularization to penalize large updates to frequently reused embeddings.

C.3.3 Cross-Edit Sharing

To examine how cross-edit sharing relates to editing outcomes, we analyze the 5,000-edit CounterFact run every 100 edits. We track n-grams selected at the last subject token of each fact's original and four generated expressions. A fact belongs to *Sharing* if any selected n-gram is also selected for another fact, and to *No sharing* otherwise. Figure 12 reports sharing rates and groups the saved final results. We observe that:

- Sharing increases overall but involves only 285 of 22,778 updated n-grams (1.25%) and 433 facts (8.66%) at 5,000 edits. Most embeddings are therefore selected for one fact; the higher fact-level rate reflects that a shared embedding serves multiple facts.
- No sharing covers 91.34% of facts with 99.98% Efficacy and 96.7% Generalization, showing sustained editing success across expressions for most facts. Their selected embeddings are not shared with other edits, so different facts do not compete for updates to the same parameters.
- Sharing contains 48 of the 49 Efficacy failures (97.96%), with lower Generalization but similar Specificity. Failures are thus concentrated in a small group of facts sharing updated embeddings, where different targets may require conflicting changes to the same parameters.