RUR-EDIT: INSTANCE-AWARE RESIDUAL ALLOCATION AND UTILITY-AWARE UPDATE RECONSTRUCTION FOR KNOWLEDGE EDITING

Anonymous authors
Paper under double-blind review

ABSTRACT

Knowledge editing aims to update specific factual knowledge in large language models while preserving unrelated knowledge. Existing multi-layer locate-then-edit methods typically distribute the remaining editing residual uniformly across candidate layers and directly apply each resulting parameter update as a whole. This coarse-grained editing overlooks heterogeneity at two levels: different layers vary in their ability to realize a target edit, while components within a layer-wise update differ in editing utility and interference with preserved knowledge. To address these limitations, we propose **RUR-Edit**, a fine-grained knowledge editing framework for both inter-layer and intra-update heterogeneity. At the inter-layer level, RUR-Edit introduces *instance-aware residual allocation*, which estimates a request-specific layer-importance profile and distributes the remaining target residual accordingly. At the intra-update level, it introduces *utility-aware update reconstruction*, which decomposes each layer-wise update into low-rank components and reconstructs it by balancing target-editing fidelity against interference with preserved knowledge. Experiments on CounterFact and ZsRE across LLaMA3, GPT-J, and Qwen2.5 show that RUR-Edit consistently achieves a stronger balance among editing efficacy, generalization, and specificity. On CounterFact, it achieves the highest overall editing score across all nine model–batch-size settings. On GPT-J, RUR-Edit improves the average overall score by 6.3 points over the strongest baseline while reducing editing time by 56.0%. Further analyses demonstrate the complementary effects of the two components, supporting fine-grained control over residual allocation and update composition for reliable and efficient knowledge editing.

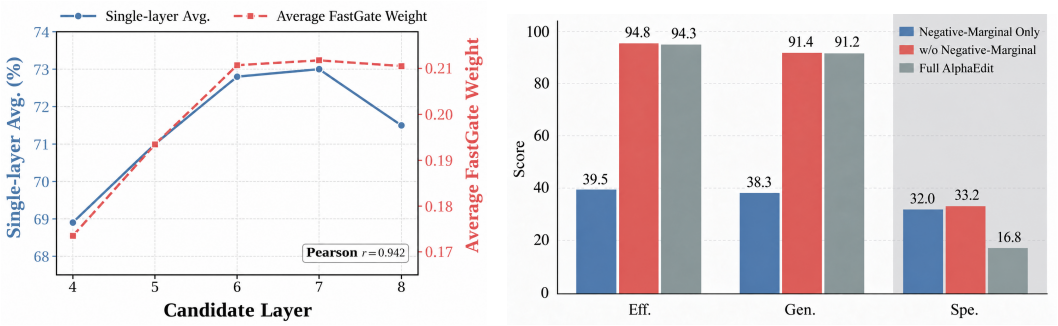
1 INTRODUCTION

Large language models (LLMs) encode extensive factual knowledge in their parameters (Petroni et al., 2019; Roberts et al., 2020; Geva et al., 2021; Dai et al., 2022), yet such knowledge can become outdated or incorrect (Lazaridou et al., 2021; Dai et al., 2022). Knowledge editing provides an efficient way to revise specific model knowledge without costly retraining (Mitchell et al., 2021; 2022; Yao et al., 2023; Wang et al., 2024c).

Among existing approaches, *locate-then-edit* methods are particularly attractive. Motivated by evidence that factual associations can often be localized within and manipulated through MLP modules (Geva et al., 2021; Dai et al., 2022; Meng et al., 2022a), they first *locate* one or more MLP layers associated with the target knowledge and then *edit* the corresponding weight matrices through closed-form parameter updates (Meng et al., 2022a;b; Li et al., 2024a; Fang et al., 2025). In multi-layer editing, the desired knowledge change is realized collaboratively across selected editing layers. Recent studies further improve the stability, locality, and scalability of this paradigm under large-scale or sequential editing (Ma et al., 2025; Li et al., 2026; Park et al., 2026). However, the *editing* stage itself remains coarse-grained at two levels: *how much of the target residual should be assigned to each editing layer* and *how the resulting parameter update should be composed*.

First, existing multi-layer editors commonly distribute the remaining target residual uniformly across the remaining editing layers (Meng et al., 2022b; Li et al., 2024a; Fang et al., 2025; Ma et al.,

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.



(a) Inter-layer heterogeneity **(b) Intra-update heterogeneity**

Figure 1: **Motivation of RUR-Edit.** (a) Editing effectiveness varies across candidate layers, motivating instance-aware residual allocation. (b) Components within a layer-wise update exhibit heterogeneous editing and preservation effects; removing negative-marginal components largely preserves efficacy and generalization while substantially improving specificity.

2025), implicitly treating these layers as equally capable of realizing the target edit. However, their editing effectiveness can differ substantially. Using AlphaEdit as the underlying editor, we independently edit each candidate layer on 500 ZsRE requests with LLaMA3. As shown in Figure 1 (a), single-layer editing performance differs substantially across layers and is strongly correlated with the average layer importance predicted by FastGate ($r = 0.942$). Moreover, FastGate produces request-dependent layer-weight profiles, as analyzed in Appendix B.2. These observations suggest that residual allocation should account for both layer-level heterogeneity and request-specific editing requirements.

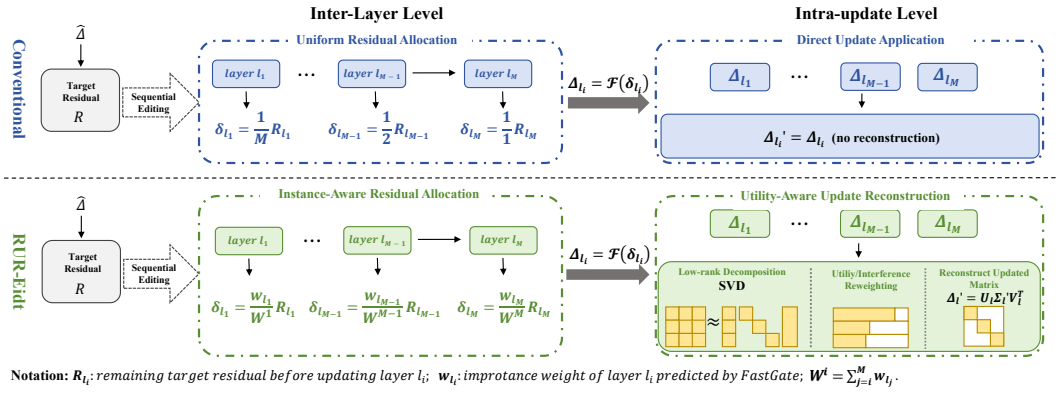
Second, existing methods typically apply each constructed layer-wise parameter update as a whole, implicitly treating its internal components as equally beneficial. We find that this assumption does not always hold. Through a leave-one-component-out analysis of AlphaEdit updates on LLaMA3 with ZsRE, we identify components with negative marginal contributions, whose removal improves the editing objective. As shown in Figure 1 (b), removing these components largely preserves editing efficacy (*Eff.*) and generalization (*Gen.*) while substantially improving specificity (*Spe.*). This reveals significant *intra-update heterogeneity*: a globally effective parameter update may still contain components with limited utility for the target edit but substantial interference with preserved knowledge.

Motivated by these observations, we propose **RUR-Edit**, which refines multi-layer knowledge editing at both levels. At the *inter-layer* level, RUR-Edit introduces **instance-aware residual allocation (IRA)**, which estimates a request-specific layer-importance profile and adaptively distributes the remaining target residual across candidate layers. At the *intra-update* level, it introduces **utility-aware update reconstruction (UAR)**, which decomposes each layer-wise parameter update and reconstructs it by balancing target-editing fidelity against potential interference with preserved knowledge. Together, IRA and UAR determine both *how much* each layer should edit and *how* its resulting parameter update should be composed.

We evaluate RUR-Edit on LLaMA3 (Grattafiori et al., 2024), GPT-J (Wang & Komatsuzaki, 2021), and Qwen2.5 (Qwen et al., 2025) across CounterFact (Meng et al., 2022a) and ZsRE (Levy et al., 2017). On CounterFact, RUR-Edit achieves the highest overall editing score in all nine model–batch-size settings. On GPT-J, it improves the average overall score by 6.3 points over SUIT while reducing editing time by 56.0%. RUR-Edit also consistently improves generalization and specificity on ZsRE. These results demonstrate that explicitly modeling both inter-layer and intra-update heterogeneity provides a more effective balance between knowledge updating, preservation, and editing efficiency.

2 Preliminaries

Multi-Layer Locate-then-Edit. Knowledge editing aims to modify a factual association (s, ρ, o) into (s, ρ, o^*), where s, ρ, and o denote the subject, relation, and original object, respectively, and o^*



Notation: R_{l_i}: *remaining target residual before updating layer* l_i; w_{l_i}: *importance weight of layer* l_i *predicted by FastGate*; $W^i = \sum_{j=i}^{M} w_{l_j}$.

Figure 2: Overview of RUR-Edit. Compared with conventional multi-layer locate-then-edit methods, which uniformly allocate the remaining target residual and directly apply each layer-wise update, RUR-Edit refines the editing process at two levels. At the inter-layer level, instance-aware residual allocation adaptively distributes the remaining residual across candidate layers. At the intra-update level, utility-aware update reconstruction decomposes and reweights each layer-wise update before applying it to the model.

is the desired object (De Cao et al., 2021; Meng et al., 2022a;b; Zhang et al., 2024). Locate-then-edit methods commonly model selected MLP weight matrices as linear associative memories that map key representations to value representations (Bau et al., 2020; Geva et al., 2021):

$$\mathbf{W}_l \mathbf{k}_l = \mathbf{v}_l, \tag{1}$$

where $\mathbf{W}_l$ denotes the edited MLP weight matrix at layer l, $\mathbf{k}_l$ is the editing key associated with the factual query, and $\mathbf{v}_l$ is the corresponding value representation. To inject a target association, these methods construct a localized update $\mathbf{\Delta}_l$ such that

$$\mathbf{\Delta}_l \mathbf{k}_l \approx \mathbf{v}_l^* - \mathbf{v}_l = \mathbf{r}_l, \tag{2}$$

where $\mathbf{v}_l^*$ denotes the desired target value and $\mathbf{r}_l$ is the residual to be realized at layer l. We use AlphaEdit (Fang et al., 2025) as the underlying closed-form editor; its complete update formulation and preservation constraints are provided in Appendix A.1.

Multi-layer editors extend this formulation by realizing the desired target representation through a sequence of updates across several editing layers. Let $\mathcal{L} = \{l_1, \ldots, l_M\}$ denote the ordered set of editing layers and $\mathbf{Z}^*$ the optimized target representation. Before editing layer l_i, the remaining discrepancy to the target is

$$\mathbf{R}_{l_i} = \mathbf{Z}^* - \mathbf{H}_{l_i}, \tag{3}$$

where $\mathbf{H}_{l_i}$ denotes the current representation after incorporating the updates from preceding editing layers. Existing methods commonly divide this remaining residual uniformly across the current and subsequent editing layers:

$$\mathbf{R}_{l_i}^{\text{uni}} = \frac{\mathbf{R}_{l_i}}{M - i + 1}. \tag{4}$$

The allocated residual is then converted by the underlying editor into a layer-wise parameter update, which is directly added to the corresponding weight matrix.

This editing process contains two distinct decisions. At the *inter-layer* level, it determines how much of the remaining target residual should be assigned to each layer. At the *intra-update* level, it determines how the resulting parameter update should modify the model. RUR-Edit refines both decisions by introducing instance-aware residual allocation and utility-aware update reconstruction.

3 METHOD

As illustrated in Figure 2, RUR-Edit refines multi-layer knowledge editing at two complementary granularities. At the *inter-layer* level, it replaces uniform residual allocation with an *instance-aware*

strategy that assigns different editing requests different layer-wise importance profiles. At the *intra-update* level, it decomposes each layer-wise parameter update and reconstructs it according to its editing fidelity and potential interference with preserved knowledge. Together, the two stages determine both *how much* each layer should edit and *how* the corresponding parameter update should be composed.

3.1 Instance-Aware Residual Allocation

Uniform allocation in Equation (4) assigns the same fraction of the remaining residual to each subsequent editing layer. However, as observed in Figure 1 (a), different layers can exhibit substantially different editing effectiveness. Moreover, the most suitable allocation can vary across editing requests. RUR-Edit therefore estimates a request-specific importance profile over the candidate editing layers and uses it to determine how the remaining residual is distributed.

Request-Specific Layer Importance. For an editing request x, we employ FastGate, a lightweight predictor, to estimate the relative importance of each candidate layer:

$$s_l(x) = g_\phi(x, l), \tag{5}$$

where $s_l(x)$ is the importance logit for layer l. The logits are normalized across all candidate editing layers:

$$w_{l_i}(x) = \frac{\exp(s_{l_i}(x))}{\sum_{j=1}^{M} \exp(s_{l_j}(x))}. \tag{6}$$

Thus, each editing request obtains its own layer-importance profile $\mathbf{w}(x)$ rather than sharing a fixed layer weighting.

FastGate is designed only to estimate relative layer importance and therefore introduces little overhead to the editing procedure. It is trained using an analytic teacher signal that approximates the ability of each layer to realize the target residual while accounting for the required parameter change. Its input features, architecture, teacher construction, training objective, and implementation details are provided in Appendix A.2.

Sequential Residual Allocation. Because multi-layer editing proceeds sequentially, the residual available to a layer depends on the updates already applied to preceding layers. Consequently, the original FastGate weights cannot be used directly at every editing step. Before editing layer l_i, we renormalize the predicted importance over only the current and subsequent editing layers:

$$\widetilde{w}_{l_i}(x) = \frac{w_{l_i}(x)}{\sum_{j=i}^{M} w_{l_j}(x)}. \tag{7}$$

The residual assigned to layer l_i is then

$$\boldsymbol{\delta}_{l_i}(x) = \widetilde{w}_{l_i}(x)\mathbf{R}_{l_i}(x). \tag{8}$$

Compared with uniform allocation, Equation (8) allows layers predicted to be more effective for the current request to undertake a larger fraction of the remaining editing burden. Meanwhile, renormalization preserves the sequential structure of multi-layer editing: after each layer is updated, the remaining residual is recomputed before allocation to the next layer. At the final layer, $\widetilde{w}_{l_M}(x) = 1$, ensuring that the remaining residual is fully assigned.

For a batch of editing requests, the allocated residuals at layer l_i are stacked into a residual matrix $\mathbf{D}_{l_i}$. Together with the corresponding editing keys $\mathbf{K}_{l_i}$, $\mathbf{D}_{l_i}$ is passed to the underlying AlphaEdit solver to obtain the original layer-wise parameter update $\boldsymbol{\Delta}_{l_i}$. The complete solver is provided in Appendix A.1.

3.2 Utility-Aware Update Reconstruction

Instance-aware residual allocation determines how much each layer should contribute to the edit, but the resulting layer-wise update $\boldsymbol{\Delta}_l$ is still conventionally applied as a whole. Our motivation analysis in Figure 1 (b) shows that the components within an update can have heterogeneous effects: some are important for realizing the target edit, while others provide limited editing benefit but introduce substantial interference with preserved knowledge. RUR-Edit therefore further refines each update at the component level.

Update Decomposition. We decompose the original layer-wise update using a rank-q singular-value decomposition:

$$\mathbf{\Delta}_l \approx \mathbf{U}_l \mathbf{\Sigma}_l \mathbf{V}_l^\top = \sum_{j=1}^{q} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top. \tag{9}$$

Each rank-one term $\sigma_j \mathbf{u}_j \mathbf{v}_j^\top$ represents a rank-one component of the original parameter update. Rather than retaining all components with their original strengths, RUR-Edit assigns each component an adaptive reconstruction coefficient.

Preservation Interference. For each component, we estimate an interference score d_j that measures its potential effect on knowledge that should remain unchanged. The score incorporates two sources of preservation information: its interaction with previously preserved key directions and its deviation from the protected null space used by the base editor. A larger d_j therefore indicates a component that is more likely to disturb preserved knowledge. The complete formulation of d_j is provided in Appendix A.3.

Utility-Aware Reconstruction. Let c_j denote the reconstruction coefficient of the j-th component. The reconstructed update is

$$\mathbf{\Delta}_l(\mathbf{c}) = \mathbf{U}_l \operatorname{diag}(\boldsymbol{\sigma} \odot \mathbf{c}) \mathbf{V}_l^\top. \tag{10}$$

We determine the coefficients by jointly considering the ability of the reconstructed update to realize the allocated editing residual and its potential interference with preserved knowledge.

$$\begin{aligned} \mathbf{c}^* = \arg\min_{\mathbf{c}} \; & \|\mathbf{\Delta}_l(\mathbf{c})\mathbf{K}_l - \mathbf{D}_l\|_F^2 \\ & + \sum_{j=1}^{q} (d_j + \lambda_{\text{edit}}) c_j^2 + \lambda_{\text{id}} \|\mathbf{c} - \mathbf{1}\|_2^2. \end{aligned} \tag{11}$$

The three terms serve complementary roles. The first measures *editing utility*: components that are necessary for reproducing the allocated residual are encouraged to retain sufficient weight. The second penalizes components with high estimated preservation interference, while λ_{edit} provides additional regularization on the reconstruction magnitude. The final identity regularizer keeps the reconstructed update close to the original solution and prevents unnecessary modification of components that are already compatible with both objectives. Consequently, the optimization does not simply remove individual components; instead, it adaptively reweights them according to their joint contribution to editing and preservation.

Finally, the refined update is reconstructed as

$$\mathbf{\Delta}_l' = \mathbf{U}_l \operatorname{diag}(\boldsymbol{\sigma} \odot \mathbf{c}^*) \mathbf{V}_l^\top, \qquad \mathbf{W}_l' = \mathbf{W}_l + \mathbf{\Delta}_l'. \tag{12}$$

The complete definition of the interference score, coefficient constraints, normalization strategy, and implementation details are provided in Appendix A.3.

4 Experiments

4.1 Experimental Setup

Models, Baselines, and Datasets. We evaluate RUR-Edit on three representative autoregressive language models from different model families and parameter scales: LLaMA3-Instruct (8B) (Grattafiori et al., 2024), GPT-J (6B) (Wang & Komatsuzaki, 2021), and Qwen2.5-Instruct (7B) (Qwen et al., 2025). We compare against representative knowledge editing baselines, including Fine-Tuning (FT-L) (Zhu et al., 2020), MEND (Mitchell et al., 2021), PMET (Li et al., 2024a), MEMIT (Meng et al., 2022b), AlphaEdit (Fang et al., 2025), and SUIT (Park et al., 2026). Detailed descriptions of the baselines are provided in Appendix C.2. All methods are evaluated under the same sequential editing protocol for fair comparison.

We conduct experiments on two widely used knowledge editing benchmarks, **CounterFact** (Meng et al., 2022a) and **ZsRE** (Levy et al., 2017), which are commonly adopted in unified knowledge editing evaluations (Wang et al., 2024b; Zhang et al., 2024).

Table 1: **CounterFact** results for sequentially editing 1,000 facts with different batch sizes (1, 10, and 100). Best results are highlighted in **bold**, and second-best results are underlined.

	Method	Batch size = 1				Batch size = 10				Batch size = 100				
		S ↑	*Eff.* ↑	*Gen.* ↑	*Spe.* ↑	*S* ↑	*Eff.* ↑	*Gen.* ↑	*Spe.* ↑	*S* ↑	*Eff.* ↑	*Gen.* ↑	*Spe.* ↑	*Flu.* ↑
LLaMA3	Pre-edit	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0	634.9
	FT-L	0.5	3.9	2.1	0.2	0.0	9.4	3.8	0.0	1.9	9.9	1.4	1.3	438.5
	MEND	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.3	519.5
	MEMIT	25.9	71.2	57.2	11.9	27.3	85.6	75.3	11.8	41.9	94.5	69.9	21.4	624.4
	PMET	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	37.6	56.6	56.0	22.6	608.8
	AlphaEdit	36.6	96.5	90.2	16.5	37.4	96.5	90.4	17.0	55.8	97.3	88.7	31.0	**633.6**
	SUIT	85.4	**99.6**	89.5	71.9	85.6	**99.9**	89.8	71.9	86.8	**99.7**	90.3	74.2	631.2
	RUR-Edit	**88.0**	**99.6**	**92.1**	**76.0**	**88.4**	99.6	**92.3**	**76.5**	**88.5**	99.5	**92.8**	**76.6**	627.7
GPT-J	Pre-edit	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0	621.1
	FT-L	12.5	31.9	23.4	6.0	9.3	47.8	32.8	3.7	13.3	64.9	46.7	5.3	334.2
	MEND	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	515.0
	MEMIT	14.8	84.2	86.1	5.6	31.5	96.1	93.1	13.5	60.2	92.0	90.4	35.8	617.4
	PMET	0.0	0.0	0.0	0.1	0.0	0.0	0.0	0.0	57.4	84.6	84.5	35.0	618.5
	AlphaEdit	72.8	98.9	94.8	48.7	76.3	99.2	96.0	53.1	73.0	98.3	95.0	49.0	**621.8**
	SUIT	82.3	98.4	92.7	64.5	84.3	99.1	94.3	67.2	84.7	99.2	94.6	67.7	619.4
	RUR-Edit	**90.1**	**100.0**	**96.7**	**77.1**	**90.0**	**100.0**	**96.6**	**77.0**	**90.1**	**99.9**	**96.6**	**77.3**	620.3
Qwen2.5	Pre-edit	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0	0.0	0.0	0.0	100.0	625.5
	FT-L	11.6	21.6	19.5	6.2	12.0	28.1	22.5	5.9	10.3	47.9	31.7	4.2	476.7
	MEND	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	466.3
	MEMIT	50.2	51.6	51.4	47.8	50.6	52.0	52.1	47.8	49.8	49.7	50.8	49.0	323.4
	PMET	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	32.6	67.5	65.9	16.1	545.2
	AlphaEdit	65.0	96.2	92.4	40.1	66.6	96.0	93.2	41.9	67.8	97.1	91.6	43.4	**626.2**
	SUIT	85.6	99.3	90.3	72.0	85.7	99.3	90.8	**71.9**	86.1	99.2	91.6	**72.3**	**626.2**
	RUR-Edit	**87.4**	**99.8**	**94.8**	**72.6**	**86.5**	**99.8**	**95.9**	70.3	**86.8**	**99.8**	**96.8**	71.4	623.6

Table 2: **ZsRE** results for sequential editing of 1000 facts with a batch size of 100.Best results are highlighted in **bold**, and second-best results are underlined. "-" indicates that the corresponding metric is unavailable.

Method	LLaMA3			GPT-J			Qwen2.5		
	Eff. ↑	*Gen.* ↑	*Spe.* ↑	*Eff.* ↑	*Gen.* ↑	*Spe.* ↑	*Eff.* ↑	*Gen.* ↑	*Spe.* ↑
Pre-edit	35.9	34.8	100.0	27.2	26.3	100.0	34.4	33.8	100.0
FT-L	34.8	34.0	0.0	69.3	60.6	0.0	12.5	11.7	0.0
MEND	0.0	0.0	0.0	0.4	0.4	0.0	0.0	0.0	0.0
MEMIT	93.5	90.8	18.0	96.8	90.8	26.3	96.2	91.4	41.5
PMET	89.2	83.9	-	95.1	89.2	-	82.1	76.3	-
AlphaEdit	94.3	91.2	16.8	99.2	95.0	28.2	92.8	90.3	40.8
SUIT	**95.0**	88.1	30.0	**99.8**	94.5	27.5	99.3	90.0	40.1
RUR-Edit	94.8	**91.4**	**33.2**	99.6	**97.5**	**28.6**	**99.5**	**93.2**	**44.5**

Sequential Editing Settings and Evaluation Metrics. For **CounterFact**, each experiment sequentially edits 1,000 facts with batch sizes of 1, 10, and 100. This setting evaluates robustness under repeated parameter updates, where editing performance and previously acquired knowledge may deteriorate as edits accumulate (Hartvigsen et al., 2023; Gupta et al., 2024; Wang et al., 2024a). Following standard knowledge editing evaluation (Meng et al., 2022a;b; Fang et al., 2025; Park et al., 2026), we report efficacy (*Eff.*), generalization (*Gen.*), specificity (*Spe.*), their harmonic mean S, and fluency for the batch-size-100 setting. For **ZsRE**, we report *Eff.* and *Gen.* on the original and paraphrased questions, respectively. Unlike prior work (Park et al., 2026), which evaluates preservation using only the first few generated tokens, we compute *Spe.* over the complete generated response, providing a stricter measure of knowledge preservation.

4.2 Main Results

Tables 1 and 2 summarize the main results. On CounterFact, RUR-Edit achieves the highest overall score S in all nine model–batch-size settings, demonstrating a consistently stronger balance among

efficacy, generalization, and specificity. The improvement is particularly pronounced on GPT-J, where RUR-Edit improves the average S by 6.3 points over SUIT across the three batch sizes. Meanwhile, fluency remains comparable to the strongest baselines.

On ZsRE, RUR-Edit achieves the best generalization and specificity on all three backbone models while maintaining competitive efficacy. In particular, the gains in specificity indicate that the improved editing performance is not obtained by sacrificing preservation of unrelated knowledge. Together, the results across CounterFact and ZsRE demonstrate that RUR-Edit provides a more favorable trade-off between successful knowledge updating and knowledge preservation.

4.3 Ablation Study

To isolate the contribution of the two components of RUR-Edit, we construct three variants based on the same AlphaEdit backbone. **IRA Only** replaces AlphaEdit's uniform residual allocation with our instance-aware residual allocation, while directly applying the resulting layer-wise update without reconstruction. **UAR Only** retains the original uniform residual allocation but applies utility-aware reconstruction to each resulting update. **RUR-Edit** combines both IRA and UAR.

Table 3: Ablation of RUR-Edit on CounterFact with a batch size of 100.

Method	*S*↑	*Eff.*↑	*Gen.*↑	*Spe.*↑
AlphaEdit	55.8	97.3	88.7	31.0
IRA Only	87.7	97.9	91.3	**76.7**
UAR Only	85.4	97.0	89.5	73.6
RUR-Edit	**88.5**	**99.5**	**92.8**	76.6

As shown in Table 3, both components independently improve over AlphaEdit. IRA Only increases S from 55.8 to 87.7, while UAR Only achieves 85.4. Combining them further improves S to 88.5 and yields the highest efficacy and generalization, demonstrating that the two components are complementary. Additional results with smaller batch sizes are provided in Appendix B.7.

4.4 Validation of Instance-Aware Residual Allocation

To further examine how layer importance should be incorporated into multi-layer editing, we compare RUR-Edit with four alternative strategies.

Global Mean assigns all editing requests a fixed layer-weight vector obtained by averaging the FastGate predictions over the first 1,000 CounterFact requests.

Uniform-Weight removes request-specific FastGate predictions and assigns equal weights to all candidate layers, reducing IRA to uniform residual allocation.

Random-Weight samples a normalized layer-weight vector before editing and applies the same fixed vector to all editing requests.

Post-scale first constructs the original layer-wise updates and then scales them using the predicted layer importance, instead of incorporating importance during residual allocation.

Table 4: Validation of instance-aware residual allocation on CounterFact. Comparison with alternative layer-importance integration strategies.

Method	*S*↑	*Eff.*↑	*Gen.*↑	*Spe.*↑
Global Mean	87.4	98.5	91.8	75.3
Uniform-Weight	85.4	97.0	89.5	73.6
Random-Weight	86.3	98.3	88.6	75.2
Post-scale	84.9	98.9	**95.3**	67.9
Top-3 Selection	87.3	99.1	89.7	76.3
No-Nullspace	87.7	99.0	92.8	75.1
RUR-Edit	**88.5**	**99.5**	92.8	**76.6**

Top-3 Selection retains only the three layers with the highest predicted importance scores, replacing soft collaboration across all candidate layers with hard layer selection (Hu et al., 2024; Wang et al., 2026).

No-Nullspace preserves instance-aware residual allocation but removes the null-space preservation constraint from the underlying editor.

As shown in Table 4, the full strategy achieves the highest overall score. Global Mean outperforms Uniform-Weight and Random-Weight, indicating that layer importance contains a useful global pattern; however, its lower performance than RUR-Edit supports the benefit of request-specific allocation. Post-scale underperforms our method, indicating that layer importance is more effective when incorporated before update construction. The lower performance of Top-3 Selection further suggests that soft multi-layer collaboration is preferable to hard layer selection. Finally, the competitive but

weaker performance of No-Nullspace indicates that the gains of IRA are not solely attributable to the null-space constraint.

4.5 Long-Term Sequential Editing

To evaluate robustness under long-term knowledge updates, we conduct large-scale sequential editing on GPT-J with **CounterFact**. We continuously edit 3,500 facts with a batch size of 100 and evaluate general capability (*GC*) after every 100 edits. This setting examines whether repeated parameter updates progressively interfere with previously acquired knowledge and general model capability.

As shown in Figure 3, RUR-Edit maintains more stable performance than SUIT as editing proceeds. Although the two methods are comparable in the early stages, SUIT gradually degrades as more edits accumulate, whereas RUR-Edit retains consistently higher GC in the later stages. These results suggest that RUR-Edit is more robust to cumulative interference during long-term sequential editing.

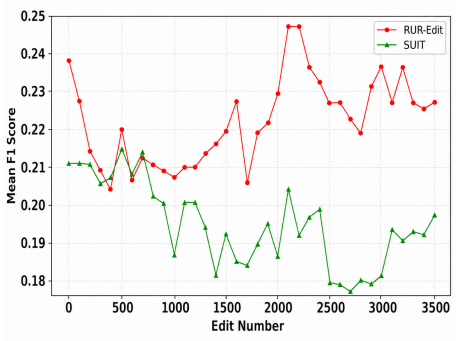


Figure 3: Long-term sequential editing on GPT-J.

4.6 Runtime Evaluation

We further evaluate the end-to-end editing efficiency of AlphaEdit, SUIT, and RUR-Edit across LLaMA3, GPT-J, and Qwen2.5. For each backbone, we measure the editing time per batch with a batch size of 1 under the same evaluation setting.

As shown in Figure 4, RUR-Edit consistently requires less runtime than AlphaEdit and SUIT. The reduction is particularly pronounced on GPT-J, where RUR-Edit requires only 61.36 seconds per batch, compared with 124.17 seconds for AlphaEdit and 139.41 seconds for SUIT, corresponding to a 55.99% reduction relative to SUIT. These results show that the additional FastGate prediction and utility-aware reconstruction do not offset the overall efficiency gains of RUR-Edit.

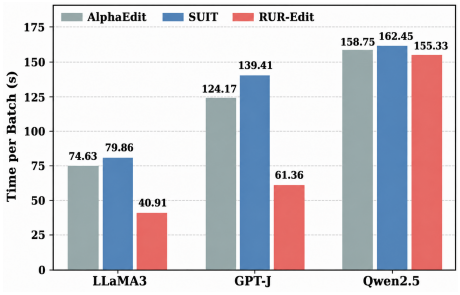


Figure 4: Runtime comparison across backbone models. Editing time per batch with a batch size of 1.

4.7 Hyperparameter Analysis

Figure 5 examines the sensitivity of RUR-Edit to β_{pre} and β_{null}. The former controls the penalty on update components that interact with preserved-key directions, while the latter penalizes deviations from the protected null space. RUR-Edit achieves the best overall score at $\beta_{\text{pre}} = 0.1$ ($S = 88.35$) and $\beta_{\text{null}} = 0.2$ ($S = 88.46$), indicating that moderate regularization provides a favorable balance between fitting the current edit and preserving existing knowledge.

Across both sweeps, S remains within a narrow range of approximately 88.07–88.46, showing that RUR-Edit is not highly sensitive to either coefficient around the selected values. We therefore use $\beta_{\text{pre}} = 0.1$ and $\beta_{\text{null}} = 0.2$ in all experiments. Metric-wise results for efficacy, generalization, and specificity are provided in Appendix B.6.

5 Related Work

Existing knowledge editing methods can be broadly grouped into three categories: parameter modification-based editing, external memory-based editing, and parameter-free editing.

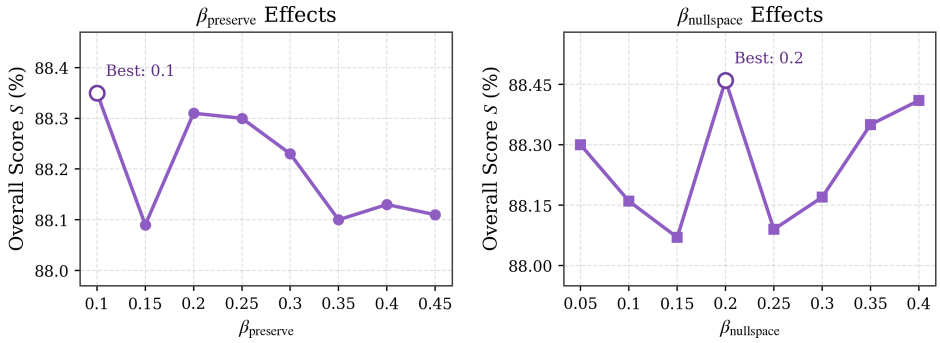


Figure 5: Hyperparameter sensitivity to β_{pre} and β_{null}.

Parameter Modification-based Editing. Parameter modification methods directly alter model parameters to update specific knowledge. ROME (Meng et al., 2022a) identifies knowledge-relevant MLP layers and performs a rank-one update to modify a factual association, while MEMIT (Meng et al., 2022b) extends this formulation to large-scale editing by distributing updates across multiple layers. Subsequent methods improve the efficiency, scalability, and preservation of parameter updates. MEND (Mitchell et al., 2021) learns an editor that transforms gradients into parameter updates, while PMET (Li et al., 2024a) and AlphaEdit (Fang et al., 2025) further refine multi-layer editing through improved optimization and subspace constraints. More recently, SUIT (Park et al., 2026) restricts key–value mappings to edit-relevant subspaces to reduce interference with preserved knowledge. Despite these advances, existing multi-layer methods generally focus on constructing the parameter update itself, while paying less attention to how the editing residual should be allocated across layers and how heterogeneous components within each update should be treated.

External Memory-based Editing. External memory-based methods preserve the original model parameters and store edited knowledge in an auxiliary memory. SERAC (Mitchell et al., 2022) retrieves previously stored edits and uses a separate counterfactual model to produce edited predictions, while MELO (Yu et al., 2024) organizes edits through lightweight adapter-based memory for efficient retrieval and reuse. These approaches provide reversible updates and reduce direct interference with model parameters, but require additional storage and retrieval mechanisms at inference time and may introduce inconsistencies between external edits and the model's internal knowledge.

Parameter-free Editing. A third line of work modifies model behavior without permanently updating its parameters. Knowledge Neurons (Li et al., 2024b) identifies neurons associated with specific factual knowledge and intervenes on their activations, while IKE (Zheng et al., 2023) uses in-context demonstrations to induce the desired knowledge without parameter modification. Such approaches avoid persistent weight updates, but typically rely on inference-time interventions or additional context, making them less suited to settings that require persistent and large-scale sequential knowledge updates.

6 Conclusion

We present RUR-Edit, a multi-layer knowledge editing framework that addresses heterogeneity at both the inter-layer and intra-update levels. RUR-Edit combines instance-aware residual allocation, which adaptively assigns the remaining target residual across candidate layers for each editing request, with utility-aware update reconstruction, which refines each layer-wise parameter update by balancing editing fidelity against potential interference with preserved knowledge. Across CounterFact and ZsRE and multiple backbone models, RUR-Edit achieves a stronger overall balance among efficacy, generalization, and specificity. It also exhibits improved robustness under long-term sequential editing and favorable end-to-end efficiency. These results suggest that finer-grained control over both residual allocation and update composition is a promising direction for reliable and scalable knowledge editing.

AI USE STATEMENT

Generative AI tools were used only to assist with language refinement, manuscript organization, and presentation improvement. All scientific ideas, methodological designs, experimental implementations, results, analyses, and conclusions were produced and verified by the authors. The authors take full responsibility for the content of this work.

ETHICS STATEMENT

This work focuses on improving the reliability and preservation of knowledge editing in large language models. All experiments are conducted on publicly available models and benchmarks and do not involve human subjects or private data. While knowledge editing can be beneficial for correcting outdated or incorrect model knowledge, it may also be misused to introduce false or misleading information. We therefore emphasize that edited models should be carefully evaluated for factual correctness, unintended side effects, and potential misuse before deployment.

REPRODUCIBILITY STATEMENT

We provide the complete formulation of RUR-Edit, including the underlying editing solver, FastGate training procedure, instance-aware residual allocation, utility-aware update reconstruction, and implementation details in the appendix. We also report the model configurations, datasets, data splits, editing layers, batch sizes, evaluation metrics, hyperparameters, and sequential editing protocols used in our experiments. Additional ablations and analysis are included to facilitate verification of the reported results. Code and configuration files will be released to support reproducibility.

REFERENCES

David Bau, Steven Liu, Tongzhou Wang, Jun-Yan Zhu, and Antonio Torralba. Rewriting a deep generative model. In *European conference on computer vision*, pp. 351–369. Springer, 2020.

Damai Dai, Li Dong, Yaru Hao, Zhifang Sui, Baobao Chang, and Furu Wei. Knowledge neurons in pretrained transformers. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8493–8502, 2022.

Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 6491–6506, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.522. URL https://aclanthology.org/2021.emnlp-main.522/.

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. Alphaedit: Null-space constrained model editing for language models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=HvSytvg3Jh.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 5484–5495, 2021.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Akshat Gupta, Anurag Rao, and Gopala Anumanchipalli. Model editing at scale leads to gradual and catastrophic forgetting. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15202–15232, 2024.

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

Chenhui Hu, Pengfei Cao, Yubo Chen, Kang Liu, and Jun Zhao. Wilke: Wise-layer knowledge editor for lifelong knowledge editing. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 3476–3503, 2024.

Angeliki Lazaridou, Adhi Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Tomas Kocisky, Sebastian Ruder, et al. Mind the gap: Assessing temporal generalization in neural language models. *Advances in Neural Information Processing Systems*, 34:29348–29363, 2021.

Omer Levy, Minjoon Seo, Eunsol Choi, and Luke Zettlemoyer. Zero-shot relation extraction via reading comprehension. In Roger Levy and Lucia Specia (eds.), *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pp. 333–342, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/K17-1034. URL https://aclanthology.org/K17-1034/.

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

Xiaopeng Li, Shangwen Wang, Shasha Li, Shezheng Song, Bin Ji, Ma Jun, and Jie Yu. Rethinking residual distribution in locate-then-edit model editing. *Advances in Neural Information Processing Systems*, 38:78348–78374, 2026.

Yongchang Li, Yujin Zhu, Tao Yan, Shijian Fan, Gang Wu, and Liang Xu. Knowledge editing for large language model with knowledge neuronal ensemble. *arXiv preprint arXiv:2412.20637*, 2024b.

Jun-Yu Ma, Hong Wang, Hao-Xiang Xu, Zhen-Hua Ling, and Jia-Chen Gu. Perturbation-restrained sequential model editing. In *International Conference on Learning Representations*, volume 2025, pp. 17302–17329, 2025.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt. *Advances in neural information processing systems*, 35:17359–17372, 2022a.

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

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

Eric Mitchell, Charles Lin, Antoine Bosselut, Christopher D Manning, and Chelsea Finn. Memory-based model editing at scale. In *International conference on machine learning*, pp. 15817–15831. PMLR, 2022.

Haewon Park, Sangwoo Kim, and Yohan Jo. Suit: Knowledge editing with subspace-aware key-value mappings. In *International Conference on Learning Representations*, volume 2026, pp. 149392–149422, 2026.

Fabio Petroni, Tim Rocktäschel, Sebastian Riedel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, and Alexander Miller. Language models as knowledge bases? In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*, pp. 2463–2473, 2019.

Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report, 2025. URL https://arxiv.org/abs/2412.15115.

Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 5418–5426, 2020.

Ben Wang and Aran Komatsuzaki. Gpt-j-6b: A 6 billion parameter autoregressive language model, 2021.

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

Peng Wang, Ningyu Zhang, Bozhong Tian, Zekun Xi, Yunzhi Yao, Ziwen Xu, Mengru Wang, Shengyu Mao, Xiaohan Wang, Siyuan Cheng, et al. Easyedit: An easy-to-use knowledge editing framework for large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 82–93, 2024b.

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

Yangfan Wang, Tianyang Sun, Chen Tang, Jie Liu, Wei Cai, and Jingchi Jiang. Hiedit: Lifelong model editing with hierarchical reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39924–39942, 2026.

Yunzhi Yao, Peng Wang, Bozhong Tian, Siyuan Cheng, Zhoubo Li, Shumin Deng, Huajun Chen, and Ningyu Zhang. Editing large language models: Problems, methods, and opportunities. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10222–10240, 2023.

Lang Yu, Qin Chen, Jie Zhou, and Liang He. Melo: Enhancing model editing with neuron-indexed dynamic lora. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19449–19457, 2024.

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

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*, pp. 4862–4876, 2023.

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 Additional Method Details

A.1 Base Editing Solver

RUR-Edit builds upon the null-space-constrained closed-form editor of AlphaEdit (Fang et al., 2025). This section provides the batch formulation, preservation constraints, and sequential update procedure used as the base editor.

Batch formulation. For a batch of n editing requests, the editing keys at layer l are organized as

$$\mathbf{K}_l = \left[\mathbf{k}_l^1 \mid \cdots \mid \mathbf{k}_l^n\right], \tag{13}$$

and the residuals assigned to this layer are

$$\mathbf{D}_l = \left[\boldsymbol{\delta}_l^1 \mid \cdots \mid \boldsymbol{\delta}_l^n\right]. \tag{14}$$

The desired parameter update satisfies

$$\boldsymbol{\Delta}_l \mathbf{K}_l \approx \mathbf{D}_l, \tag{15}$$

while minimizing interference with knowledge that should remain unchanged.

Null-space projection. Let $\mathbf{K}_{0,l}$ contain key representations sampled from a large collection of text and approximate the knowledge to be preserved at layer l. We construct its non-central covariance matrix

$$\mathbf{C}_{0,l} = \mathbf{K}_{0,l}\mathbf{K}_{0,l}^\top, \tag{16}$$

and perform eigendecomposition

$$\mathbf{C}_{0,l} = \mathbf{U}_{0,l}\boldsymbol{\Lambda}_l\mathbf{U}_{0,l}^\top. \tag{17}$$

Let $\widetilde{\mathbf{U}}_l$ contain eigenvectors whose corresponding eigenvalues are below a threshold τ. The projection matrix is then

$$\mathbf{P}_l = \widetilde{\mathbf{U}}_l\widetilde{\mathbf{U}}_l^\top, \tag{18}$$

which approximately satisfies

$$\boldsymbol{\Delta}_l\mathbf{P}_l\mathbf{K}_{0,l} \approx \mathbf{0}. \tag{19}$$

Since $\mathbf{P}_l$ depends only on the preserved-key statistics, it is computed once for each editing layer and reused across editing batches.

Prior edits and preservation cache. During sequential editing, updates should also preserve knowledge introduced by previous editing batches. Suppose that $t-1$ batches have already been edited at layer l. Their keys form

$$\mathbf{K}_{p,l}^{(t)} = \left[\mathbf{K}_l^{(1)} \mid \cdots \mid \mathbf{K}_l^{(t-1)}\right]. \tag{20}$$

Rather than retaining all historical keys, we maintain their Gram matrix

$$\mathbf{C}_l^{(t)} = \mathbf{K}_{p,l}^{(t)}\left(\mathbf{K}_{p,l}^{(t)}\right)^\top = \sum_{j=1}^{t-1}\mathbf{K}_l^{(j)}\left(\mathbf{K}_l^{(j)}\right)^\top, \tag{21}$$

which can be updated recursively as

$$\mathbf{C}_l^{(t+1)} = \mathbf{C}_l^{(t)} + \mathbf{K}_l^{(t)}\left(\mathbf{K}_l^{(t)}\right)^\top. \tag{22}$$

Closed-form update. Following AlphaEdit, the base layer update can be written as

$$\begin{aligned}\boldsymbol{\Delta}_l = \arg\min_{\widetilde{\boldsymbol{\Delta}}_l} &\left\|\widetilde{\boldsymbol{\Delta}}_l\mathbf{P}_l\mathbf{K}_l - \mathbf{D}_l\right\|_F^2 + \lambda\left\|\widetilde{\boldsymbol{\Delta}}_l\mathbf{P}_l\right\|_F^2 \\ &+ \left\|\widetilde{\boldsymbol{\Delta}}_l\mathbf{P}_l\mathbf{K}_{p,l}\right\|_F^2.\end{aligned} \tag{23}$$

Its closed-form solution is

$$\mathbf{\Delta}_l = \mathbf{D}_l \mathbf{K}_l^\top \mathbf{P}_l \left(\mathbf{K}_{p,l} \mathbf{K}_{p,l}^\top \mathbf{P}_l + \mathbf{K}_l \mathbf{K}_l^\top \mathbf{P}_l + \lambda \mathbf{I} \right)^{-1}. \tag{24}$$

In implementation, the corresponding linear system is solved directly rather than explicitly computing the matrix inverse. Throughout the paper, we use the orientation $\mathbf{W}_l \mathbf{k}_l = \mathbf{v}_l$; implementations storing the edited projection matrix in transposed form use the corresponding transposed system.

RUR-Edit modifies this base procedure in two places. Instance-aware residual allocation determines $\mathbf{D}_l$ before the closed-form update is constructed, while utility-aware update reconstruction refines the resulting $\mathbf{\Delta}_l$ before it is applied.

Sequential multi-layer editing. For the ordered editing layers $\mathcal{L} = \{l_1, \ldots, l_M\}$, updates are applied sequentially. After editing layer l_i, the model is forwarded to the next layer and the remaining target residual is recomputed:

$$\mathbf{R}_{l_{i+1}} = \mathbf{Z}^* - \mathbf{H}_{l_{i+1}}. \tag{25}$$

Thus, an update applied at one layer changes the residual subsequently handled by the remaining layers.

A.2 FastGate Architecture and Training

FastGate estimates a request-dependent importance profile over the candidate editing layers. It is used only for residual allocation and does not directly modify model parameters.

Input representation. For an editing request $x = (s, \rho, o, o^*)$ and candidate layer l, FastGate constructs

$$\mathbf{z}_l(x) = [\mathbf{f}_l(x); \mathbf{e}_l; \mathbf{e}_\rho], \tag{26}$$

where $\mathbf{e}_l$ is a learnable layer embedding, $\mathbf{e}_\rho$ is a relation embedding, and $\mathbf{f}_l(x)$ contains compact request- and layer-dependent statistics. These include summaries of the hidden representation and editing key together with the model likelihoods of the original and desired target objects.

FastGate maps this representation to a scalar importance logit:

$$s_l(x) = g_\phi(\mathbf{z}_l(x)), \tag{27}$$

where g_ϕ is a lightweight multilayer perceptron shared across candidate layers. The logits are normalized as

$$w_l(x) = \frac{\exp(s_l(x))}{\sum_{l' \in \mathcal{L}} \exp(s_{l'}(x))}. \tag{28}$$

This yields a request-specific importance profile

$$\mathbf{w}(x) = [w_{l_1}(x), \ldots, w_{l_M}(x)]. \tag{29}$$

Analytic teacher signal. Directly performing complete editing with every candidate layer for every training request would be expensive. We therefore construct an analytic teacher that estimates how effectively a candidate layer can realize the target residual.

Let $\mathbf{t}$ denote the target residual and

$$\bar{\mathbf{k}}_l = \mathbf{P}_l \mathbf{k}_l \tag{30}$$

the projected editing key. We approximate the update at layer l as

$$\widehat{\mathbf{\Delta}}_l = \frac{\mathbf{t} \bar{\mathbf{k}}_l^\top}{\lambda + \mathbf{k}_l^\top \bar{\mathbf{k}}_l}, \tag{31}$$

whose induced residual change is

$$\widehat{\mathbf{t}}_l = \widehat{\mathbf{\Delta}}_l \mathbf{k}_l. \tag{32}$$

The teacher jointly considers residual reduction and update magnitude:

$$\mathrm{Gain}_l = \frac{\|\mathbf{t}\|_2 - \|\mathbf{t} - \widehat{\mathbf{t}}_l\|_2}{\|\mathbf{t}\|_2 + \epsilon}, \tag{33}$$

$$\mathrm{Risk}_l = \frac{\|\widehat{\mathbf{\Delta}}_l\|_F}{\|\mathbf{t}\|_2 + \epsilon}, \tag{34}$$

and

$$y_l = \mathrm{Gain}_l - \log\left(1 + \mathrm{Risk}_l\right). \tag{35}$$

Thus, a layer receives a higher teacher score when it is predicted to realize more of the desired residual with a smaller parameter perturbation.

Training objective. FastGate is supervised using the teacher scores computed for all candidate layers. When trained by direct score regression, the objective is

$$\mathcal{L}_{\mathrm{gate}} = \frac{1}{M} \sum_{l \in \mathcal{L}} \left(s_l(x) - y_l(x)\right)^2. \tag{36}$$

A separate FastGate is trained for each backbone–dataset combination. Its detailed data partition is reported in Appendix C.3.

A.3 Utility-Aware Update Reconstruction

Component decomposition. Given the original layer-wise update $\mathbf{\Delta}_l$, we compute a truncated singular-value decomposition:

$$\mathbf{\Delta}_l \approx \mathbf{U}_l \mathbf{\Sigma}_l \mathbf{V}_l^\top = \sum_{j=1}^{q} \sigma_j \mathbf{u}_j \mathbf{v}_j^\top. \tag{37}$$

We denote its j-th rank-one component by

$$\mathbf{C}_j = \sigma_j \mathbf{u}_j \mathbf{v}_j^\top. \tag{38}$$

Preservation interference. For each component, RUR-Edit estimates potential interference using two preservation signals. The first measures its interaction with the preservation cache:

$$d_j^{\mathrm{pre}} = \mathrm{Norm}\left(\sigma_j \sqrt{\mathbf{v}_j^\top \mathbf{C}_l \mathbf{v}_j}\right). \tag{39}$$

The second measures deviation from the protected null space:

$$d_j^{\mathrm{null}} = \|\mathbf{v}_j - \mathbf{P}_l \mathbf{v}_j\|_2^2. \tag{40}$$

The combined interference score is

$$d_j = \beta_{\mathrm{pre}} d_j^{\mathrm{pre}} + \beta_{\mathrm{null}} d_j^{\mathrm{null}}. \tag{41}$$

We use maximum normalization

$$\mathrm{Norm}(a_j) = \frac{a_j}{\max_k a_k + \epsilon}. \tag{42}$$

Editing utility and reconstruction. Rather than introducing a separate heuristic utility score, RUR-Edit measures editing utility through the ability of the reconstructed update to reproduce the allocated residual.

Let $\mathbf{c} = [c_1, \ldots, c_q]^\top$ denote the component coefficients. The reconstructed update is

$$\mathbf{\Delta}_l(\mathbf{c}) = \mathbf{U}_l \,\mathrm{diag}(\boldsymbol{\sigma} \odot \mathbf{c}) \mathbf{V}_l^\top. \tag{43}$$

The coefficients are obtained by

$$\begin{aligned} \mathbf{c}^* = \arg\min_{\mathbf{c}} \; & \|\mathbf{\Delta}_l(\mathbf{c}) \mathbf{K}_l - \mathbf{D}_l\|_F^2 \\ & + \sum_{j=1}^{q} \left(d_j + \lambda_{\mathrm{edit}}\right) c_j^2 + \lambda_{\mathrm{id}} \|\mathbf{c} - \mathbf{1}\|_2^2. \end{aligned} \tag{44}$$

Table 5: **Layer-wise editing heterogeneity.** Each candidate layer is independently edited using AlphaEdit on 500 ZsRE requests with LLaMA3. Weight denotes the FastGate weight averaged over these requests and is used only for aggregate analysis.

Layer	*Eff.*	*Gen.*	*Spe.*	*Avg.*	Weight
4	90.1	83.5	33.0	68.9	0.1735
5	93.4	87.3	32.4	71.0	0.1935
6	94.9	90.7	32.7	72.8	0.2107
7	**95.5**	**90.8**	32.6	**73.0**	**0.2117**
8	93.9	87.5	33.0	71.5	0.2105

The first term preserves the editing utility of the reconstructed update, the second penalizes components with high preservation interference and excessive magnitude, and the final term discourages unnecessary deviation from the original update.

After optimization, we constrain

$$c_j \leftarrow \mathrm{clip}\left(c_j, c_{\min}, c_{\max}\right), \tag{45}$$

and construct

$$\boldsymbol{\Delta}_l' = \mathbf{U}_l \,\mathrm{diag}(\boldsymbol{\sigma} \odot \mathbf{c}^*)\mathbf{V}_l^\top. \tag{46}$$

The edited weight matrix is finally

$$\mathbf{W}_l' = \mathbf{W}_l + \boldsymbol{\Delta}_l'. \tag{47}$$

B Additional Analyses

B.1 Inter-Layer Editing Heterogeneity

We provide detailed results for the layer-wise motivation experiment introduced in Section 1. Using LLaMA3, we independently edit each candidate layer with AlphaEdit on 500 ZsRE requests while keeping all other settings unchanged. For each layer, we summarize the editing performance as

$$\mathrm{Avg.} = \frac{\mathrm{Eff.} + \mathrm{Gen.} + \mathrm{Spe.}}{3}. \tag{48}$$

Editing performance varies noticeably across the candidate layers. The average score increases from 68.9 at layer 4 to 73.0 at layer 7, a 4.1-point difference, and subsequently decreases at layer 8. This non-monotonic behavior indicates that deeper layers are not necessarily more effective editing locations.

Across the five candidate layers, the average editing score is strongly correlated with the average FastGate weight ($r = 0.942$), consistent with Figure 1 (a). This aggregate result indicates that FastGate captures meaningful differences in editing capability across layers. Because averaging alone cannot establish request-specific behavior, we next analyze the predicted weights at the individual-request level.

B.2 Instance-Specific Layer-Importance Profiles

Although Figure 1 (a) reports weights averaged across requests, RUR-Edit does not use a fixed dataset-level weighting. For each request x_i, FastGate independently predicts

$$\mathbf{w}(x_i) = [w_{i,4}, w_{i,5}, w_{i,6}, w_{i,7}, w_{i,8}], \qquad \sum_{l \in \mathcal{L}} w_{i,l} = 1. \tag{49}$$

The relative layer ranking changes across editing requests; for example, layer 5 receives the largest weight for Case 1, whereas layers 7 or 8 dominate several other requests. These results confirm that FastGate produces request-dependent layer profiles rather than directly applying the averaged profile shown in Figure 1 (a).

Table 6: **Instance-specific FastGate weights for ten ZsRE requests.** Each row is independently predicted and normalized across the five candidate layers. Bold denotes the largest weight for each request.

Case	w_4	w_5	w_6	w_7	w_8
0	0.1693	0.1873	0.2009	**0.2260**	0.2166
1	0.1716	**0.2243**	0.2140	0.1943	0.1959
2	0.1731	0.2000	0.2037	0.2105	**0.2127**
3	0.1673	0.1856	0.2152	0.2157	**0.2163**
4	0.1722	0.1998	0.2074	0.2057	**0.2149**
5	0.1731	0.2072	0.2035	0.2064	**0.2098**
6	0.1736	0.2018	0.2065	0.2054	**0.2127**
7	0.1698	0.2033	0.2035	0.2104	**0.2130**
8	0.1719	0.1982	0.2033	**0.2136**	0.2130
9	0.1707	0.1986	0.2038	0.2075	**0.2193**

Method	*Eff.*	*Gen.*	*Spe.*
Negative-Marginal Only	39.5	38.3	32.0
w/o Negative-Marginal	**94.8**	**91.4**	**33.2**
Full AlphaEdit	94.3	91.2	16.8

Table 7: Effect of negative-marginal update components on LLaMA3 with ZsRE. Removing these components largely preserves efficacy and generalization while substantially improving specificity.

B.3 Intra-Update Heterogeneity

To examine whether all components of a successful layer-wise update contribute positively to the target edit, we perform a leave-one-component-out analysis. Given

$$\mathbf{\Delta}_l = \sum_{j=1}^{q} \mathbf{C}_j, \tag{50}$$

we remove one component at a time:

$$\mathbf{\Delta}_l^{(-j)} = \mathbf{\Delta}_l - \mathbf{C}_j. \tag{51}$$

Let $\mathcal{R}(\mathbf{\Delta})$ denote the discrepancy between the edited representation and its optimized target. We define the marginal contribution of component j as

$$m_j = \mathcal{R}\left(\mathbf{\Delta}_l^{(-j)}\right) - \mathcal{R}\left(\mathbf{\Delta}_l\right). \tag{52}$$

A component with $m_j < 0$ is termed a *negative-marginal component*, because removing it reduces the target discrepancy.

Removing negative-marginal components changes efficacy from 94.3 to 94.8 and generalization from 91.2 to 91.4, while specificity increases from 16.8 to 33.2. Thus, a globally successful update can still contain components with limited utility for the desired edit and substantial interference with preserved knowledge, motivating component-level reconstruction.

B.4 Teacher Signal Analysis

We analyze the two components of the FastGate teacher signal. Figure 6 illustrates the layer-wise teacher weights for a CounterFact editing example on LLaMA3. Removing Gain produces an almost uniform distribution, indicating that residual reduction provides the primary signal for distinguishing layer importance. Removing Risk retains the non-uniform pattern, suggesting that this term mainly regularizes update magnitude.

Table 8 further reports the corresponding editing performance. Removing either component degrades the overall editing behavior: w/o Gain causes a pronounced drop in efficacy, while w/o Risk

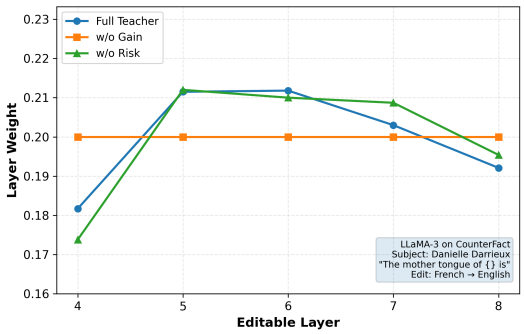


Figure 6: Effect of the Gain and Risk terms on teacher-generated layer weights for a CounterFact editing instance.

Table 8: Ablation of the Gain and Risk terms in the FastGate teacher.

Method	*Eff.*	*Gen.*	*Spe.*
w/o Gain	97.6	91.4	74.5
w/o Risk	99.4	91.6	75.3
Full Teacher	**99.5**	**92.8**	**76.6**

mainly reduces generalization and specificity. The Full Teacher achieves the best efficacy, generalization, and specificity, confirming that Gain and Risk play complementary roles in residual reduction and update-cost regularization.

B.5 Joint Analysis of Editing Quality and Efficiency

To complement the runtime results in the main paper, we further provide a joint comparison of editing quality and computational efficiency on GPT-J. Figure 7 summarizes the overall score S, editing efficacy, generalization, specificity, and normalized time efficiency for AlphaEdit, SUIT, and RUR-Edit.

For time efficiency, we normalize the runtime of each method as $\min(t)/t \times 100$, such that a higher value indicates lower editing cost. RUR-Edit achieves the highest time-efficiency score because it requires only 61.36 seconds per batch, compared with 124.17 seconds for AlphaEdit and 139.41 seconds for SUIT. At the same time, RUR-Edit maintains strong efficacy and generalization while substantially improving specificity and the overall editing score. This comparison shows that the performance gains of RUR-Edit are not obtained at the expense of editing efficiency; instead, it provides a more favorable trade-off between editing quality and computational cost.

B.6 Hyperparameter Sensitivity

Figure 8 reports the metric-wise sensitivity to β_{pre} and β_{null}. Efficacy remains approximately 99.5–99.6 across both sweeps, indicating that these coefficients primarily regulate the trade-off between generalization and specificity rather than whether the target fact can be successfully edited.

For β_{pre}, both generalization and specificity vary mildly, with $\beta_{\text{pre}} = 0.1$ providing a favorable balance. For β_{null}, larger penalties increasingly change the balance between the two preservation-related metrics, and $\beta_{\text{null}} = 0.2$ achieves the best overall score in our validation setting.

B.7 Ablation across Batch Sizes

Table 9 reports additional ablations for batch sizes of 1 and 10. Both instance-aware residual allocation (IRA) and utility-aware update reconstruction (UAR) improve substantially over AlphaEdit, while their combination achieves the strongest overall performance.

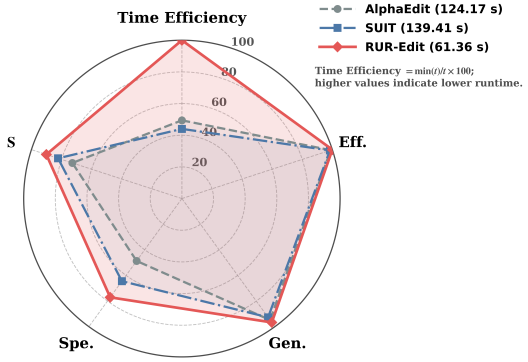


Figure 7: **Joint comparison of editing quality and time efficiency on GPT-J.** Time efficiency is normalized as $\min(t)/t \times 100$, where higher values indicate lower runtime. RUR-Edit achieves a stronger overall balance across editing quality and computational efficiency.

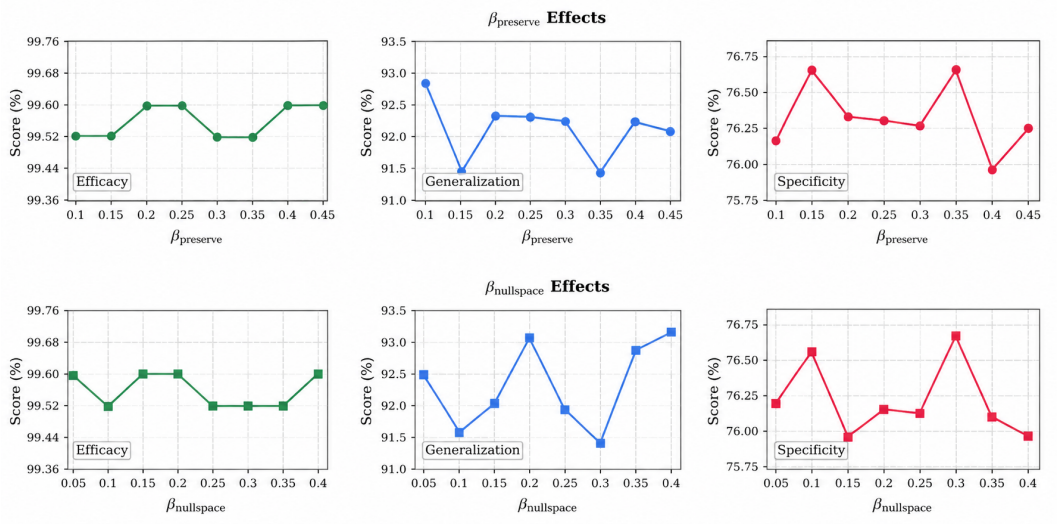


Figure 8: Metric-wise sensitivity to β_{pre} and β_{null}.

C Additional Experimental Details

C.1 Models and Datasets

Models. We evaluate RUR-Edit on three autoregressive language models from different model families: LLaMA3-Instruct (8B) (Grattafiori et al., 2024), GPT-J (6B) (Wang & Komatsuzaki, 2021), and Qwen2.5-Instruct (7B) (Qwen et al., 2025). GPT-J is widely used in prior knowledge-editing studies, while LLaMA3 and Qwen2.5 provide more recent instruction-tuned architectures for assessing cross-model generalization.

Datasets. We use CounterFact (Meng et al., 2022a) and ZsRE (Levy et al., 2017). CounterFact evaluates efficacy on rewriting prompts, generalization on paraphrases, and specificity using neighborhood prompts. ZsRE formulates factual editing as question–answer mappings and provides alternative query formulations for evaluating generalization.

C.2 Baseline Methods

We compare RUR-Edit with six representative baselines. FT-L (Zhu et al., 2020) directly fine-tunes selected model parameters under regularization. MEND (Mitchell et al., 2021) learns auxiliary

Table 9: Additional ablation results on CounterFact with batch sizes of 1 and 10. IRA denotes instance-aware residual allocation and UAR denotes utility-aware update reconstruction.

	Batch Size = 1				Batch Size = 10			
Method	*S*	*Eff.*	*Gen.*	*Spe.*	*S*	*Eff.*	*Gen.*	*Spe.*
AlphaEdit	36.6	96.5	90.2	16.5	37.4	96.5	90.4	17.0
IRA Only	87.4	97.6	90.7	76.5	87.7	97.7	91.4	76.7
UAR Only	84.8	96.5	88.1	73.2	85.0	96.7	88.6	73.1
RUR-Edit	**88.0**	**99.6**	**92.1**	76.0	**88.4**	**99.6**	**92.3**	76.5

Table 12: Data partition used for FastGate training and knowledge-editing evaluation. The same non-overlapping partition is applied independently to each model–dataset combination.

Purpose	Example indices	Number
Standard editing evaluation	0–999	1,000
FastGate training	1,000–4,999	4,000
FastGate validation	5,000–5,499	500
FastGate testing	5,500–5,999	500
Large-scale editing evaluation	6,000–9,499	3,500

networks that transform gradients into localized parameter updates. MEMIT (Meng et al., 2022b) extends closed-form associative-memory editing to batches of facts and multiple layers. PMET (Li et al., 2024a) jointly optimizes internal target representations before updating feed-forward weights. AlphaEdit (Fang et al., 2025) constrains parameter updates using a null-space projection to reduce interference with preserved knowledge. SUIT (Park et al., 2026) constructs subspace-aware key–value mappings and restricts updates to edit-relevant subspaces.

Table 10: Main hyperparameters of utility-aware update reconstruction.

Symbol	Description	Value
q	Retained SVD components	96
β_{pre}	Preservation-cache penalty	0.10
β_{null}	Null-space deviation penalty	0.20
λ_{edit}	Component magnitude regularization	0.05
λ_{id}	Identity regularization	0.45
$c_{\min}$	Minimum reconstruction coefficient	0.60
$c_{\max}$	Maximum reconstruction coefficient	1.10

Table 11: Architecture and optimization hyperparameters of FastGate.

Hyperparameter	Value
Input feature dimension	16
Layer embedding dimension	16
Relation embedding dimension	32
Hidden dimension	$128, 64$
Number of predictor layers	3
Activation	$GELU$
Optimizer	$AdamW$
Learning rate	1×10^{-3}
Batch size	128
Training epochs	80

C.3 FastGate Data Partition

FastGate is trained independently for each backbone–dataset combination. The examples used for predictor training are strictly separated from all knowledge-editing evaluation examples.

Thus, neither the standard evaluation nor the 3,500-edit sequential evaluation overlaps with the examples used to train, validate, or test FastGate.

C.4 Implementation and Hyperparameters

The main hyperparameters introduced by RUR-Edit are summarized in Table 10.

The FastGate architecture and optimization hyperparameters are summarized in Table 11.

C.5 COMPUTATIONAL OVERHEAD

FastGate operates only on compact request-level features and predicts one scalar score per candidate layer, so its inference overhead is small relative to the matrix operations required by the underlying closed-form editor. Utility-aware reconstruction additionally requires a truncated SVD and optimization over q component coefficients. The end-to-end runtime comparison reported in the main experiments includes all of these additional operations.