TamEdit: Trajectory-Aware Meta-Learning for Specificity-Preserving Continual Knowledge Editing

Shiqiang Tian[1]*, **Cheng Ding**[1]*, **Qin Chen**[1]†, **Jie Zhou**[1], **Liang He**[1]
[1] School of Computer Science and Technology, East China Normal University
{sqtian, cding}@stu.ecnu.edu.cn, {qchen, jzhou, lhe}@cs.ecnu.edu.cn

Abstract

Knowledge editing is a promising method for updating Large Language Models efficiently. However, previous studies often suffer from poor specificity in continual editing, as they typically focus on single edits or preventing knowledge forgetting. To address this, we propose TamEdit, a trajectory-aware meta-learning method that preserves specificity for continual knowledge editing. TamEdit unifies three levels: Inner Optimization performs multi-step fast fine-tuning on the single edit; Trajectory-based Editing unifies continual edits with a growing memory; and Outer Optimization leverages meta-learning to distill cross-task strategies for preserving specificity. By capturing the relationships between different single edits within the trajectory, our method learns how to effectively avoid specificity drift. Experiments across multiple LLMs show TamEdit significantly outperforms baselines in continual editing, improving specificity by 14.81% with sub-second inference speed (0.55s per edit), while preserving general capabilities.

1 Introduction

Large language models (LLMs) learn from large text data during pre-training (Roberts et al., 2020; Wang et al., 2024a; Zhuang et al., 2025), and they store many kinds of knowledge in their parameters (Mann et al., 2020; Chen et al., 2025). However, as real-world facts evolve (Li and Chu, 2024), updating LLMs becomes essential. Since full fine-tuning is expensive and cannot be done frequently (Xu et al., 2025; Feng et al., 2025), knowledge editing has emerged as an efficient alternative (Wang et al., 2024b; Hong and Lipani, 2024). While methods like ROME (Meng et al., 2022), MEMIT (Meng et al., 2023), and AlphaEdit (Fang et al., 2024) have been applied to continual edits, they often suffer from specificity drift and catastrophic forgetting over long editing sequences as they treat updates independently.

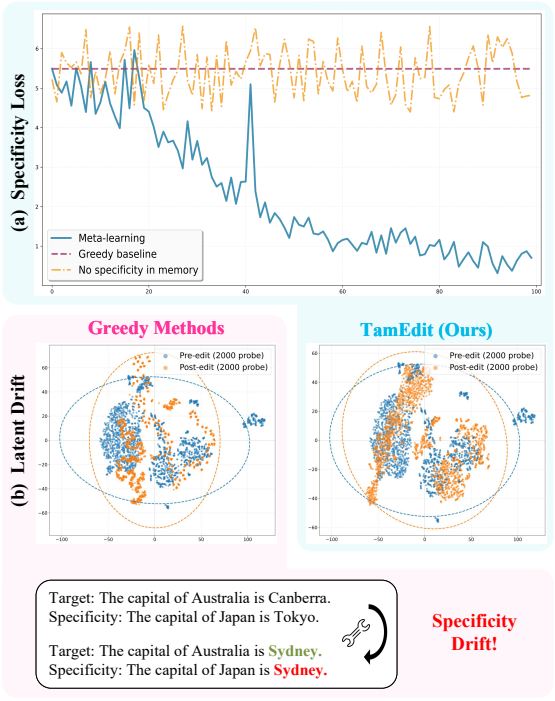

Figure 1: (a) Specificity loss for our meta-learning method vs. the greedy baseline and the variant without specificity in memory. (b) Latent space shifts across continual edits (Ours vs. Greedy Methods). Bottom is a specificity drift example.

Specifically, current continual editing methods typically calculate an optimal update ΔW for each piece of knowledge K in isolation, adding it directly to the LLM parameters W (Jiang et al., 2025; Li et al., 2025; Qiao et al., 2025). This greedy optimization neglects relationships between different pieces of knowledge, leading to specificity drift and error accumulation (Hu et al., 2024; Wang et al., 2025) as shown in Figure 1. The root cause lies in that existing methods are confined to the local optima of single edits, while neglecting the maintenance of global specificity throughout the continual

* Equal contribution.
† Corresponding author.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 21417–21439
July 2-7, 2026

editing process.

To address this challenge, we propose TamEdit, a **t**rajectory-**a**ware **m**eta-learning method that preserves specificity for continual knowledge **edit**ing, which formulates continual editing as a cohesive trajectory rather than isolated steps. Since continual editing requires rapidly learning from multiple few-shot samples while preventing forgetting (Yap et al., 2021), optimization must be conducted at the trajectory level. Furthermore, we employ a hypernetwork (meta-editor) to learn generalizable capabilities across diverse continual editing tasks. Within each trajectory, we introduce a growing memory mechanism that incorporates previously edited knowledge and unrelated facts to mitigate specificity drift and error accumulation. Overall, the optimization is structured into a three-level hierarchy: **Inner Optimization** employs the meta-editor to rapidly adapt to each edit, ensuring high *Efficacy* and *Generalization*; **Trajectory-based Editing** level treats a sequence of edits as a unified trajectory and dynamically updates the memory; and **Outer Optimization** leverages meta-learning to distill generalizable editing strategies across trajectories, equipping the meta-editor with cross-task generalization capability to resist specificity drift and error accumulation. This design separates immediate adaptation from global specificity maintenance, ensuring that rapid learning does not lead to specificity drift and error accumulation.

We evaluate our method on LLaMA-3-8B[1] and Mistral-7B-v0.3 (Jiang et al., 2023) using ZsRE (Levy et al., 2017) and FEVER (Thorne et al., 2018) benchmarks. Following prior work (Li et al., 2025), we simulate a setting with 8,000 items in 400 sequential batches. Results show that greedy baselines such as ROME (Meng et al., 2022), MEMIT (Meng et al., 2023), and AlphaEdit (Fang et al., 2024) suffer from declining *Specificity* as edits accumulate. RLEdit (Li et al., 2025) partially mitigates this issue. In contrast, our editor remains stable. It matches baselines in *Efficacy* and *Generalization* while achieving higher *Specificity*.

To the best of our knowledge, this is the first work to formulate continual editing as a meta-learning problem focused on optimizing editor initialization for resisting specificity drift and error accumulation. By optimizing over entire trajectories rather than isolated edits, we provide a principled framework for continual knowledge editing.

[1] https://www.llama.com/models/llama-3/

In summary, our main contributions are:

- We propose a trajectory-aware meta-learning method that optimizes cross-task capability in continual editing, addressing specificity drift and error accumulation.
- We design a three-level optimization hierarchy with growing memory, balancing rapid adaptation with long-term specificity.
- Experiments on various datasets show significant improvements in Specificity, while still maintaining competitive Efficacy and Generalization.

2 Related Work

2.1 Knowledge Editing in Language Models

Knowledge editing updates stored factual knowledge while preserving unrelated behaviors, broadly categorized into *parameter-modifying* and *parameter-preserving* methods (Yao et al., 2023).

Parameter-Modifying. Fine-tuning (Zhu et al., 2021) updates parameters on edit examples but is difficult to localize, often perturbing unrelated knowledge. Locate-and-edit methods like ROME (Meng et al., 2022) and MEMIT (Meng et al., 2023) identify key layers or locations responsible for facts before updating weights. While MEMIT supports batch editing via a one-shot solve, continuous application causes accumulated interference and failures. Hypernetwork-based methods (e.g., KE (De Cao et al., 2021), MEND (Mitchell et al., 2022a), MALMEN (Tan et al., 2024)) train networks to map edit instances to parameter updates for fast test-time editing. Specifically, MEND transforms fine-tuning gradients, while KE directly predicts updates. Although efficient, they optimize each edit in isolation with a fixed hypernetwork, leading to specificity drift and error accumulation in continual editing scenarios. In contrast, we learn an editor initialization with cross-task generalization capability to rapidly adapt to new editing tasks, while maintaining knowledge and specificity.

Parameter-Preserving. These approaches use external storage. SERAC (Mitchell et al., 2022b) learns a counterfactual model with a classifier to judge response relevance, while T-Patcher (Huang et al., 2023) adds neurons to steer outputs. GRACE (Hartvigsen et al., 2023) and MELO (Yu

et al., 2024) adapt computation via retrieved knowledge; MemPrompt (Madaan et al., 2022) and IKE (Zheng et al., 2023) use in-context learning with retrieved demonstrations. While avoiding parameter updates, they incur growing storage and retrieval overhead as the number of edits increases, limiting scalability to linear $\mathcal{O}(N)$ growth. In contrast, our method directly optimizes model parameters via a meta-editor with cross-task generalization capability, maintaining a strict $\mathcal{O}(1)$ storage requirement without external retrieval at inference time by introducing only a very lightweight parameter hypernetwork.

2.2 Meta-Learning

Meta-learning ("learning to learn") enables rapid adaptation from prior experience. Gradient-based methods like MAML (Finn et al., 2017) learn initializations for fast adaptation, while Meta-SGD (Li et al., 2017) parameterizes adaptation dynamics. These provide foundations for transferable initializations relevant to continual editing.

We extend these ideas to study continual editing as learning to edit over time, beyond single step or task level approaches. We formulate editing at the trajectory level, combining meta-learning with HyperNetworks to train a *continual* editor reasoning over entire trajectories. Our meta-objective optimizes performance over long editing sequences rather than isolated updates.

3 The Challenge of Continual Editing

While effective for single updates, greedy editing suffers from specificity drift and error accumulation when applied sequentially (Zhang et al., 2024; Ma et al., 2024; Cao et al., 2026). Existing methods optimize each edit independently, focusing only on immediate success while ignoring interactions across edits. As a result, they fail to capture cross-edit interference and cumulative effects over time. We argue that continual editing should be optimized over the entire trajectory rather than isolated steps. Compared with greedy approaches, trajectory-level optimization explicitly accounts for dependencies between edits and their long-term impact, enabling more stable updates and better preservation of unrelated knowledge.

3.1 The Trajectory View of Editing

We view continual editing as a trajectory in parameter space. Each edit updates parameters as:

$$W' = W + \Delta W, \tag{1}$$

where $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$, ΔW is the produced increment in a mini-batch. After T continual edits:

$$W_T = W_0 + \sum_{t=1}^{T} \Delta W_t. \tag{2}$$

where W_0 denotes the initial model parameters. Most existing methods generate ΔW_t *greedily* by maximizing success on the current request.

As shown in Figure 1 (b), we use T-SNE to visualize the last-layer hidden representations of 2000 probing samples before and after a sequence of continual edits. This analysis reveals that AlphaEdit (greedy editing) causes significant representation drift, while our method preserves the original representation distribution more effectively, thereby preventing specificity drift. Detailed experimental settings are provided in Appendix A.7.

3.2 Greedy vs. Trajectory-Level Objectives

Given a continual editing task $E = (B_1, \ldots, B_T)$, greedy methods independently optimize each mini-batch $B_t = \{k_1, \ldots, k_n\}$ at step t, where $k_i = (s, r, o^*)$ denotes the i-th editing triplet consisting of a subject, relation, and target object. The vanilla loss function can be expressed as:

$$\min_{\Delta W_t} \mathbb{E}_{k \sim B_t} \big[-\log P_{W_{t-1}+\Delta W_t}(o^* | s, r) \big] \tag{3}$$

This targets immediate success but, as evidenced by the representation drift discussed in Section 3.1, ignores long-term stability. Inspired by meta-learning, we introduce an editor θ that generates parameter updates ΔW to simultaneously achieve rapid task adaptation and preservation of unrelated knowledge. Specifically, a trajectory-level objective is designed to model the relationships across successive batch updates. It evaluates the accumulated parameter updates $\sum_{t=1}^{T} \Delta W_t$ against the accumulated memory $\mathcal{M}_T = \bigcup_{t=1}^{T} B_t$:

$$\min_{\theta} \quad \mathbb{E}_{E \sim \mathcal{D}_{\text{traj}}} \Big[\mathcal{L}\big(\mathcal{M}_T;\ \sum_{t=1}^{T} \Delta W_t\big) \Big] \tag{4}$$

where $\mathcal{L}(\mathcal{M}_T; \sum_{t=1}^{T} \Delta W_t)$ denotes the trajectory-level memory loss evaluated on the accumulated

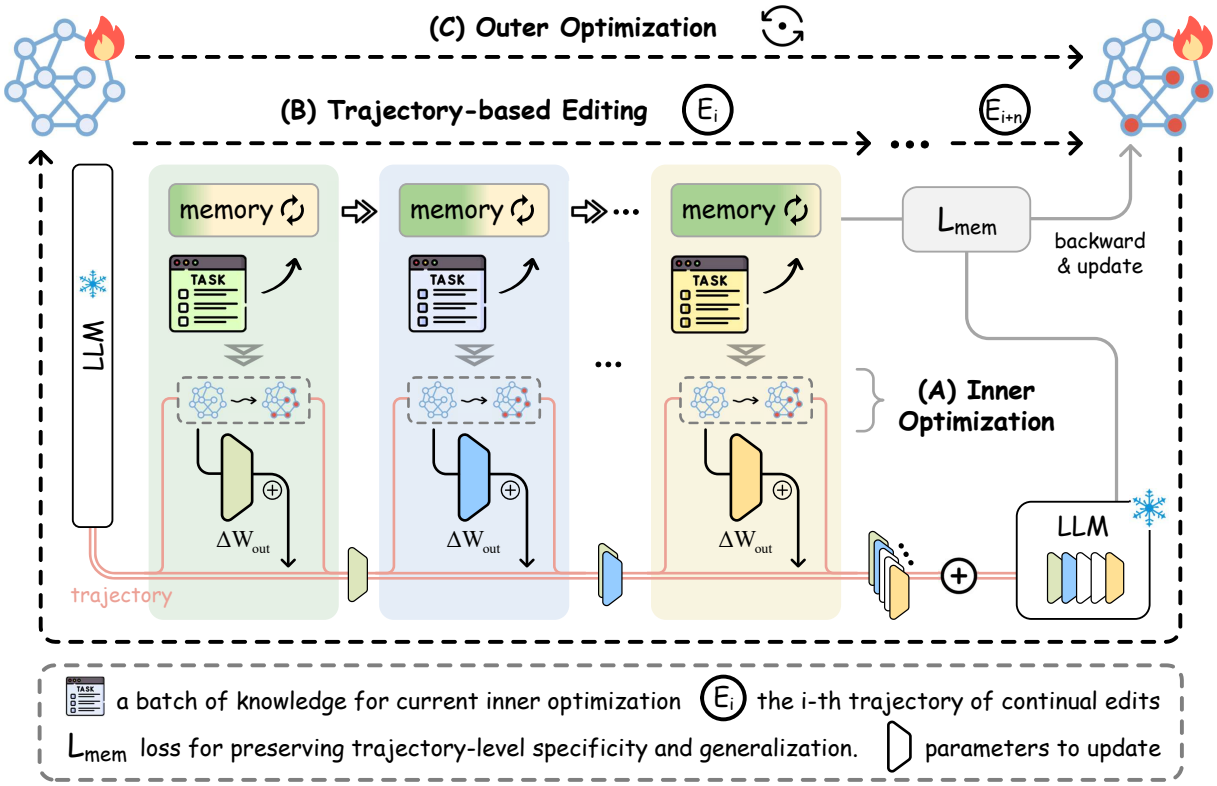

Figure 2: Overview of the trajectory-aware editing framework. (A) Inner Optimization: rapidly adapts the model to each mini-batch of edits. (B) Trajectory-based Editing: models continual editing as a sequence of mini-batch updates with growing memory to avoid specificity drift. (C) Outer Optimization: learns trajectory-level editing strategies across different editing trajectories to preserve specificity over long sequences.

memory $\mathcal{M}_T$, and $E \sim \mathcal{D}_{\text{traj}}$ denotes an editing trajectory sampled from the trajectory distribution.

Figure 1 (a) confirms this: meta-learning with diverse evaluation datasets (paraphrase, neighborhood, and attribute) steadily reduces specificity loss. In contrast, a greedy baseline with a random editor initialization remains unchanged, and removing neighborhood and attribute data from memory prevents the editor from learning cross-trajectory resistance to specificity drift, resulting in non-convergent loss curves.

4 Trajectory-Aware Meta-Learning

Our method separates fast adaptation from long-term specificity in continual editing. Inner Optimization handles the current edit, Trajectory-based Editing models cross-edit interactions over time, and Outer Optimization learns trajectory-level strategies that guide updates toward preserving unrelated knowledge. All these components form a trajectory-aware meta-learning framework that balances immediate success with sustained specificity in continual knowledge editing.

4.1 Problem Formulation

Metric Definition. To evaluate the editor on continual editing tasks, we construct four evaluation datasets to comprehensively assess the updated model across four metrics. Details are as follows:

$$\mathcal{P}_{\text{req}}(k) = \{(s, r, o^*)\} \tag{5}$$

$$\mathcal{P}_{\text{para}}(k) = \{(s, r^*, o^*)\} \tag{6}$$

$$\mathcal{P}_{\text{neigh}}(k) = \{(s', r, o_{s'}) \mid s' \in \mathcal{N}(s)\} \tag{7}$$

$$\mathcal{P}_{\text{attr}}(k) = \{(s', r, o^*) \mid s' \in \mathcal{A}(s)\} \tag{8}$$

where r^* denotes paraphrased relations, $\mathcal{N}(s)$ and $\mathcal{A}(s)$ denote neighborhood and attribute entities, o^* is the target knowledge, and $o_{s'}$ is the original. These datasets assess three key capabilities. **Efficacy** and **Generalization**: $\mathcal{P}_{\text{req}}$ and $\mathcal{P}_{\text{para}}$ verify whether the model successfully updates the knowledge to the target o^*; **Specificity**: $\mathcal{P}_{\text{neigh}}$ and $\mathcal{P}_{\text{attr}}$ evaluate specificity by ensuring the model preserves the original outputs, keeping $o_{s'}$ (old) for neighbors and o^* (new) for attributes. Only $\mathcal{P}_{\text{req}}$ is used for inner optimization, while the remaining sets are used in trajectory-based editing and outer optimization to enforce specificity.

Data Organization. We structure the continual editing process into three hierarchical levels:

- **Mini-batch** $B = \{k_1, \ldots, k_n\}$: A set of knowledge items to be edited simultaneously.
- **Trajectory** $E = (B_1, \ldots, B_T)$: A sequence of mini-batches applied to the base model, representing a complete editing trajectory.
- **Memory** $\mathcal{M}_T = \bigcup_{t=1}^{T} B_t$: The cumulative set of edited items up to step T. We utilize the associated datasets ($\mathcal{P}_{\text{para}}, \mathcal{P}_{\text{neigh}}, \mathcal{P}_{\text{attr}}$) from $\mathcal{M}_T$ for outer optimization evaluation.

Target Modules. To perform knowledge editing, we identify the specific modules in LLMs where factual associations are stored and updated. We focus on the FFN output projections W^l_{out}, which serve as the target modules for editing. At layer l, the FFN output $\mathbf{m}^l$ is computed from the previous hidden state $\mathbf{h}^{l-1}$ and the attention output $\mathbf{a}^l$ through layer normalization $\gamma(\cdot)$, input/output projections $W^l_{\text{in}}, W^l_{\text{out}}$, and activation $\sigma(\cdot)$. Following (Meng et al., 2022), W^l_{out} can be interpreted as a linear associative memory that maps an internal key $\mathbf{k}$ to a value $\mathbf{v}$:

$$\underbrace{\mathbf{m}^l}_{\mathbf{v}} = W^l_{\text{out}} \underbrace{\sigma(W^l_{\text{in}}\, \gamma(\mathbf{h}^{l-1} + \mathbf{a}^l))}_{\mathbf{k}}. \tag{9}$$

This property makes W^l_{out} a suitable editing location, as modifying it directly updates factual associations while limiting interference to unrelated knowledge. Based on this, we edit W^l_{out}, denoted as W, and use the editor to generate the update ΔW_t. Unlike greedy editors that optimize each update independently (Eq. 3), our method learns these updates via trajectory-aware meta-learning (Eq. 4), enabling better preservation of specificity in continual knowledge editing.

4.2 Inner Optimization

At the inner level, the goal of the editor is to rapidly adapt the model to the current mini-batch B_t by minimizing the rewrite loss defined in Eq. 3 on $\mathcal{P}_{\text{req}}$. This step focuses on fast, batch-level adaptation without considering long-term interactions across edits. To realize this process, we employ a hypernetwork to generate parameter updates conditioned on each knowledge item.

Knowledge Embedding. For each knowledge item $k_i \in B_t$, we derive an embedding $\mathbf{z}_i$ by feeding $\mathcal{P}_{\text{req}}(k_i)$ into the backbone f_W and extracting the hidden state of the last token from the final layer:

$$\mathbf{z}_i = \big[f_W(\mathcal{P}_{\text{req}}(k_i))\big]_{\text{last}} \in \mathbb{R}^H. \tag{10}$$

where H represents the dimension of the hidden state. Stacking the embeddings of all m knowledge items in the mini-batch yields the batch input $Z \in \mathbb{R}^{m\times H}$.

Low-Rank Hypernetwork Updates. To ensure efficient and controlled parameter updates, we adopt a low-rank formulation that restricts the update space, reducing interference with unrelated knowledge while maintaining sufficient expressiveness. Given the mini-batch input Z, we map it into a hidden representation and process it with a three-layer feed-forward network:

$$\tilde{H} = \text{MLP}\big(\eta(\psi(ZW_e))\big), \quad \tilde{H} \in \mathbb{R}^{m\times d}, \tag{11}$$

where $W_e \in \mathbb{R}^{H\times d}$, $\psi(\cdot)$ denotes the GELU activation, $\eta(\cdot)$ denotes Layer Normalization, and MLP$(\cdot)$ denotes a feed-forward network.

Two linear projections then generate the low-rank components:

$$\begin{aligned} U &= \text{reshape}(f_u(\tilde{H})) \in \mathbb{R}^{m\times d_{\text{out}}\times\rho}, \\ V &= \text{reshape}(f_v(\tilde{H})) \in \mathbb{R}^{m\times d_{\text{in}}\times\rho}, \end{aligned} \tag{12}$$

where ρ is the shared rank. The functions $f_u(\cdot)$ and $f_v(\cdot)$ denote linear projections applied to $\tilde{H}$, and reshape$(\cdot)$ expands the output dimensions to form matrices for each knowledge item. For each knowledge item i, we have $U_i \in \mathbb{R}^{d_{\text{out}}\times\rho}$ and $V_i \in \mathbb{R}^{d_{\text{in}}\times\rho}$.

Finally, the update is constructed by aggregating the low-rank updates across B_t:

$$\Delta W_t = \frac{1}{m}\sum_{i=1}^{m} U_i V_i^\top, \quad \Delta W_t \in \mathbb{R}^{d_{\text{out}}\times d_{\text{in}}}. \tag{13}$$

Optimization Objective. The inner objective minimizes the rewrite negative log-likelihood on $\mathcal{P}_{\text{req}}$ alongside an L_2 regularization to restrict the update magnitude:

$$\begin{aligned} \min_\theta \mathbb{E}_{(x,y)\sim\mathcal{P}_{\text{req}}(B_t)}\Big[-\log P_{W_{t-1}+\Delta W_t}(y \mid x)\Big] \\ + \lambda_{\text{reg}}\|\Delta W_t\|_2^2, \end{aligned} \tag{14}$$

where $x = (s, r)$ denotes the input and $y = o^*$ denotes the target knowledge. After multi-step gradient descent on $\mathcal{L}_{\text{inner}}$, the final update ΔW_t is applied to W_{t-1}.

4.3 Trajectory-based Editing

At the trajectory level, continual edits are modeled as a sequence of updates over a trajectory. For $t = 1, \ldots, T$, the model parameters evolve as:

$$W_t = W_{t-1} + \Delta W_t, \tag{15}$$

while the memory grows as:

$$\mathcal{M}_t = \mathcal{M}_{t-1} \cup B_t. \tag{16}$$

This formulation captures the cumulative effect of sequential edits.

Since the inner optimization only focuses on $\mathcal{P}_{\text{req}}$, it cannot ensure global specificity across the trajectory. To address this, after the trajectory ends at step T, we evaluate the hypernetwork using the accumulated memory $\mathcal{M}_T$, consisting of generalization and specificity probes ($\mathcal{P}_{\text{para}}, \mathcal{P}_{\text{neigh}}, \mathcal{P}_{\text{attr}}$). We define the memory loss for each prompt category $c \in \{\text{para}, \text{neigh}, \text{attr}\}$ as the expected negative log-likelihood:

$$\mathcal{L}_c(\mathcal{M}_T; W_T) = \mathbb{E}_{(x,y)\sim\mathcal{P}_c(\mathcal{M}_T)} \big[-\log P_{W_T}(y \mid x) \big]. \tag{17}$$

where $P_{W_T}(y \mid x)$ denotes the probability of generating y given x under model parameters W_T, and (x, y) is a knowledge tuple with input $x = (s, r)$ and target $y = o$. The total trajectory-level objective is a weighted sum:

$$\mathcal{L}_{\text{mem}}(\mathcal{M}_T; W_T) = \sum_c \lambda_c \mathcal{L}_c(\mathcal{M}_T; W_T). \tag{18}$$

where λ_c balances generalization and specificity.

This objective evaluates the accumulated parameter updates W_T generated by the hypernetwork, measuring both generalization and specificity. Crucially, $\mathcal{M}_T$ is only used during meta-training, and no memory is required at inference time.

4.4 Outer Optimization

At the outer level, we perform meta-optimization over trajectories to learn the hypernetwork initialization. Unlike inner optimization that focuses on rapid adaptation to the current mini-batch, the outer optimization aggregates over the trajectory level and optimizes the hypernetwork based on the final editing outcome after a sequence of updates.

Based on the trajectory-level memory loss defined above, the objective is:

$$\theta^* = \arg\min_\theta \ \mathbb{E}_{E\sim\mathcal{D}_{\text{traj}}} \Big[\mathcal{L}_{\text{mem}}(\mathcal{M}_T; W_T) \Big] \quad \text{s.t.} \quad W_T = W_0 + \sum_{t=1}^{T} \Delta W_t \tag{19}$$

where E denotes an editing trajectory sampled from $\mathcal{D}_{\text{traj}}$. Here, the inner optimization generates the sequence of updates ΔW_t, trajectory-based editing accumulates them into the final model parameters W_T while also accumulating the memory $\mathcal{M}_T$, and the outer optimization updates θ by minimizing the memory loss on all trajectories $\mathcal{D}_{\text{traj}}$.

This objective evaluates the quality of the entire editing trajectory rather than the success of an individual update. By optimizing the hypernetwork initialization across trajectories, the model learns generalizable editing strategies that improve specificity preservation.

5 Experiments

5.1 Experimental Setup

LLM Backbones. We experiment with two representative 8B-scale models: LLaMA-3-8B[2] and Mistral-7B-v0.3 (Jiang et al., 2023).

Datasets and Metrics. We train our editor using trajectories constructed from CounterFact (Meng et al., 2022), and evaluate continual-editing performance on two standard benchmarks, ZsRE (Levy et al., 2017) and FEVER (Thorne et al., 2018). Following prior work (Mitchell et al., 2022a; Meng et al., 2022, 2023), we report *Efficacy*, *Generalization*, and *Specificity* as evaluation metrics. Dataset details and metric definitions are provided in Appendix A.2 and A.3, respectively.

Baselines. We compare against representative methods (details in Appendix A.1): FT (Zhu et al., 2021), ROME (Meng et al., 2022), MEND (Mitchell et al., 2022a), MALMEN (Tan et al., 2024), DAFNet (Zhang et al., 2024), MEMIT (Meng et al., 2023), PRUNE (Ma et al., 2024), RECT (Gu et al., 2024), AlphaEdit (Fang et al., 2024), and RLEdit (Li et al., 2025).

Table 1: Comparison of TamEdit and baseline methods on continual editing tasks. Avg is computed by averaging all six metrics. The best results are highlighted in boldface with a dark background, while the second best results are marked with a light background. The upper block contains fine-tuning and locate-then-edit methods, and the lower block contains hypernetwork-based methods. MEND* and MALMEN* retrain their hypernetworks for each new knowledge edit. † denotes results of baseline methods taken from RLEdit (Li et al., 2025).

Methods	FEVER							
	LLaMA-3-8B			Mistral-7B-v0.3			Avg↑	Time↓
	Eff.	Gen.	Spe.	Eff.	Gen.	Spe.		
FT†	1.80 ±0.10	6.39 ±0.17	26.33 ±0.13	17.08 ±0.24	21.57 ±0.30	37.47 ±0.32	18.44	0.3358s
ROME†	38.56 ±0.28	44.53 ±0.29	9.29 ±0.17	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	15.40	2.1662s
MEMIT†	0.18 ±0.02	0.01 ±0.01	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.03	6.3423s
PRUNE†	56.64 ±0.24	43.31 ±0.18	0.85 ±0.05	5.27 ±0.34	3.11 ±0.16	5.89 ±0.23	19.18	6.3356s
RECT†	60.95 ±0.27	52.40 ±0.26	1.75 ±0.07	0.55 ±0.04	0.05 ±0.01	0.00 ±0.00	19.28	6.0486s
AlphaEdit†	94.22 ±0.25	**94.14 ±0.18**	25.57 ±0.14	32.74 ±0.42	30.03 ±0.29	8.44 ±0.45	47.52	6.2307s
MEND†	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00	0.9175s
MEND*†	10.37 ±0.19	9.34 ±0.22	4.88 ±0.24	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	4.10	6.1280s
MALMEN†	0.01 ±0.01	0.01 ±0.01	0.14 ±0.03	16.67 ±0.21	16.61 ±0.18	12.16 ±0.16	7.60	1.9858s
MALMEN*†	5.74 ±0.20	5.29 ±0.13	1.26 ±0.18	17.73 ±0.24	13.30 ±0.27	13.85 ±0.21	9.53	9.3358s
DAFNet†	31.27 ±0.47	28.82 ±0.43	66.55 ±0.41	4.86 ±0.14	4.21 ±0.19	41.71 ±0.48	29.57	8.2553s
RLEdit†	95.34 ±0.34	93.58 ±0.38	70.36 ±0.29	88.99 ±0.22	88.25 ±0.25	73.64 ±0.11	85.03	**0.2238s**
TamEdit	**95.45 ±0.23**	93.97 ±0.37	**73.92 ±0.52**	**89.26 ±0.51**	**88.53 ±0.34**	**76.25 ±0.45**	**86.23**	0.5523s

Methods	ZsRE							
	LLaMA-3-8B			Mistral-7B-v0.3			Avg↑	Time↓
	Eff.	Gen.	Spe.	Eff.	Gen.	Spe.		
FT†	17.10 ±0.22	16.73 ±0.22	8.27 ±0.13	32.84 ±0.30	33.78 ±0.30	**42.19 ±0.31**	25.82	0.3366s
ROME†	0.54 ±0.04	0.57 ±0.04	0.40 ±0.02	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.25	2.5621s
MEMIT†	0.00 ±0.00	0.00 ±0.00	0.13 ±0.02	0.00 ±0.00	0.00 ±0.00	0.13 ±0.02	0.04	6.0677s
PRUNE†	12.27 ±0.43	12.01 ±0.23	9.88 ±0.29	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	5.69	6.1588s
RECT†	11.05 ±0.41	8.12 ±0.15	28.12 ±0.13	8.18 ±0.33	8.04 ±0.46	11.32 ±0.49	12.47	6.6558s
AlphaEdit†	86.83 ±0.23	81.48 ±0.28	29.09 ±0.22	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	32.90	6.1831s
MEND†	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00	0.9686s
MEND*†	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	0.00	5.9265s
MALMEN†	9.87 ±0.12	9.00 ±0.09	2.11 ±0.15	0.00 ±0.00	0.00 ±0.00	0.00 ±0.00	3.50	2.2779s
MALMEN*†	12.23 ±0.11	11.08 ±0.22	2.43 ±0.09	0.01 ±0.01	0.02 ±0.01	1.25 ±0.04	4.50	9.4277s
DAFNet†	21.99 ±0.47	11.17 ±0.43	32.21 ±0.39	1.25 ±0.08	2.12 ±0.12	25.27 ±0.54	15.67	8.2383s
RLEdit†	**89.42 ±0.34**	87.32 ±0.23	44.78 ±0.50	71.12 ±0.31	**67.42 ±0.27**	27.43 ±0.44	64.58	**0.2224s**
TamEdit	88.92 ±0.28	**87.59 ±0.19**	**51.41 ±0.47**	**72.56 ±0.21**	65.35 ±0.51	35.26 ±0.25	**66.85**	0.5466s

5.2 Main Results

To evaluate the editor's ability to incorporate new facts while preserving existing knowledge, we sample 8,000 items and divide them into 400 sequential batches of size 20 (400×20 setting). Starting from a base model, we apply these batches sequentially to form continuous update trajectories. We then evaluate *Efficacy*, *Generalization*, and *Specificity* on the final model. Results are summarized in Table 1. The "Time" column reports average editing time per knowledge edit, including costs such as covariance computation or hypernetwork updates.

Table 1 shows that most baselines suffer from the same issue that specificity degrades as edits accumulate. While AlphaEdit maintains strong Efficacy and Generalization, its Specificity drops to 15.78% on average. The strongest baseline, RLEdit, achieves an average Specificity score of 54.05% across all datasets. In contrast, our method TamEdit further improves Specificity by 5.16%. Overall, our method obtains the best performance, achieving a good balance among Efficacy, Generalization, and Specificity. These observations indicate that TamEdit learns how to preserve specificity rather than dataset-specific patterns by meta-learning with three-level optimizations.

[2] https://www.llama.com/models/llama-3/

Table 2: Ablation studies on initialization strategies under the 20×100 setting.

Method	LLaMA-3-8B						Mistral-7B-v0.3					
	ZsRE			FEVER			ZsRE			FEVER		
	Eff.	Gen.	Spe.	Eff.	Gen.	Spe.	Eff.	Gen.	Spe.	Eff.	Gen.	Spe.
Random Init	51.26	40.06	28.18	53.21	41.52	31.37	47.81	40.24	26.67	47.97	46.27	29.90
SFT Init	47.79	40.88	18.13	46.57	35.63	29.75	46.42	41.89	27.52	45.75	39.62	22.76
TamEdit	**88.10**	**83.41**	**53.02**	**93.61**	**91.79**	**72.52**	**73.37**	**68.13**	**46.05**	**94.15**	**89.86**	**79.89**

Table 3: Ablation on memory composition using LLaMA-3-8B on ZsRE under the 20×100 setting. $\mathcal{P}_{\text{spe}}$ denotes $\mathcal{P}_{\text{neigh}} \cup \mathcal{P}_{\text{attr}}$.

Memory	Eff.↑	Gen.↑	Spe.↑
$\mathcal{P}_{\text{para}} + \mathcal{P}_{\text{spe}}$ (Ours)	88.10	83.41	53.02
$\mathcal{P}_{\text{req}} + \mathcal{P}_{\text{spe}}$	87.02 ↓1.08	79.26 ↓4.15	54.91 ↑1.89
$\mathcal{P}_{\text{spe}}$	73.74 ↓14.36	69.97 ↓13.44	51.64 ↓1.38
$\mathcal{P}_{\text{para}}$	56.34 ↓31.76	50.18 ↓33.23	37.71 ↓15.31
$\mathcal{P}_{\text{req}}$	56.67 ↓31.43	47.96 ↓35.45	30.86 ↓22.16

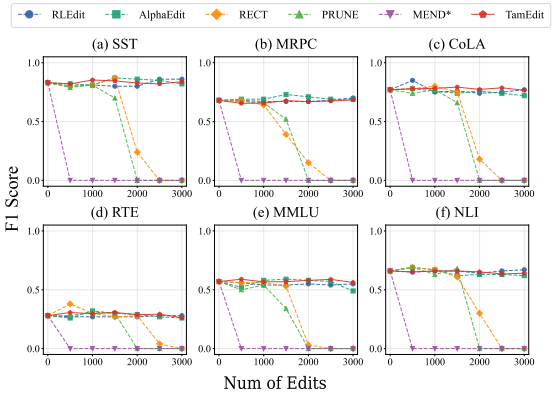

Figure 3: Performance of the post-edited model on downstream tasks used for general capability testing.

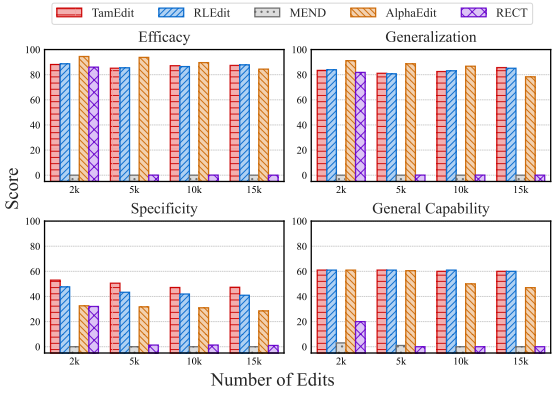

Figure 4: Performance of different methods with various editing scales.

5.3 Further Analyses

Ablation Studies. Table 2 shows that the initialization strategy plays a crucial role in continual editing. Compared with Random Init and SFT Init, TamEdit consistently achieves the best overall results across both backbones and datasets, indicating that our initialization method provides a stronger starting point for continual editing. In particular, TamEdit shows a significant advantage in Specificity, demonstrating effective resistance to specificity drift. Meanwhile, it also maintains strong Efficacy and Generalization, suggesting that the gain in Specificity does not come at the expense of editing effectiveness. Regarding memory composition, we experiment with LLaMA-3-8B on ZsRE, and the results are shown in Table 3. As discussed in Section 4.1, $\mathcal{P}_{\text{neigh}}$ and $\mathcal{P}_{\text{attr}}$ are closely related to specificity. Therefore, after incorporating $\mathcal{P}_{\text{neigh}}$ and $\mathcal{P}_{\text{attr}}$ into the memory, Specificity is substantially improved.

General Capability Preservation. We further evaluate the general capability of the post-edited model on six downstream benchmarks: SST (Socher et al., 2013), MRPC (Dolan and Brockett, 2005), CoLA (Warstadt et al., 2019), RTE (Bentivogli et al., 2009), MMLU (Hendrycks et al., 2021), and NLI (Williams et al., 2018). As Figure 3 shows, TamEdit preserves the general capabilities as AlphaEdit and RLEdit. Details are provided in Appendix C.2.

Effect of Editing Scale. We scale to 15,000 editing items on ZsRE with different continual-editing settings (batches × samples/batch): 20×100, 50×100, 100×100, and 150×100. Figure 4 shows that our method significantly outperforms strong baselines as AlphaEdit and RLEdit on the Specificity metric with the growing editing scale, while maintaining comparable performance in terms of other metrics. More details are in Appendix C.1.

Adaption to Editing Techniques. Since our hypernetwork produces parameter increments without modifying backbone weights, it is inherently compatible with complementary editing techniques. We

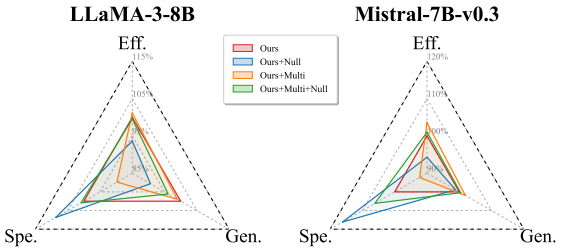

Figure 5: Comparison of editor variants on LLaMA-3-8B and Mistral-7B-v0.3 under the 400×20 setting.

Table 4: Effect of edited-layer choices using LLaMA-3-8B on ZsRE under the 20×100 setting.

Edited Layers	Eff.	Gen.	Spe.
5	88.10	83.41	53.02
6	86.27	82.60	**53.32**
7	86.80	82.33	49.02
6,7	87.11	82.09	49.31
5,6,7	87.85	83.31	50.70
5,6,7,8	**88.40**	**83.97**	48.92

further analyze this extensibility by incorporating two variants: null-space projection (Fang et al., 2024) and multi-layer editing. Null-space projection projects each update ΔW into the null space of preserved knowledge, preventing interference with untouched key-value associations. For multi-layer editing, the hypernetwork applies multiple FFN layers.

Figure 5 evaluates the performance of various editors within the framework of our method. Null-space projection (Ours+Null) improves Specificity, while this gain comes at the expense of Efficacy and Generalization. Multi-layer editing (Ours+Multi) retains Efficacy and Generalization but degrades in Specificity. The combination (Ours+Multi+Null) balances well among all editing metrics, showing the flexibility and potential of our method to incorporate more advanced editors. Details are in Appendix C.3.

Effect of Edited Layers. Table 4 shows the results of various layer selections. Single-layer editing at layer 5 achieves high Specificity of about 53% while keeping competitive Efficacy and Generalization. Conversely, expanding to multiple layers (e.g., 5–8) yields marginal gains in Efficacy and Generalization but degrades Specificity due to increased interference. We thus select layer 5 for editing in experiments.

6 Conclusions and Future Work

We propose TamEdit, a trajectory-aware meta-learning method for continual knowledge editing. By unifying fast inner optimization, trajectory-based editing, and outer optimization, TamEdit distills transferable strategies to mitigate specificity drift over lifelong edits. Future work will explore the impact of sustained specificity on long-chain reasoning, scale to larger models, and extend to broader alignment objectives like safety.

Limitations

While our method demonstrates promising continual editing performance, its evaluation is currently confined to standard benchmarks rather than broader data domains, leaving the behavior of post-edited LLMs on multi-hop reasoning and complex inference chains underexplored. Moreover, although our reliance on a low rank assumption for parameter updates is theoretically sound and practically effective in the current setting, it may eventually encounter capacity bottlenecks under substantially more complex editing tasks. Fully addressing these capacity constraints, alongside specificity drift and error accumulation across diverse knowledge types, remains a critical open challenge for future work.

Acknowledgments

This research is funded by the National Nature Science Foundation of China (No.62477010, No.62577022 and No.62307028), AI for Science Program, Shanghai Municipal Commission of Economy and Informatization under Grant (2025-GZL-RGZN-BTBX-02018), and Shanghai Science and Technology Innovation Action Plan (No.24YF2710100).

References

Armen Aghajanyan, Sonal Gupta, and Luke Zettlemoyer. 2021. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 7319–7328, Online. Association for Computational Linguistics.

Luisa Bentivogli, Peter Clark, Ido Dagan, and Danilo Giampiccolo. 2009. The fifth pascal recognizing textual entailment challenge. *TAC*, 7(8):1.

Ding Cao, Yuchen Cai, Yuqing Huang, Xuesong He, Rongxi Guo, Guiquan Liu, and Guangzhong Sun. 2026. On the superimposed noise accumulation problem in sequential knowledge editing of large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30139–30147.

Kedi Chen, Dezhao Ruan, Yuhao Dan, Yaoting Wang, Siyu Yan, Xuecheng Wu, Yinqi Zhang, Qin Chen, Jie Zhou, Liang He, Biqing Qi, Linyang Li, Qipeng Guo, Xiaoming Shi, and Wei Zhang. 2025. A survey of inductive reasoning for large language models. *Preprint*, arXiv:2510.10182.

Nicola De Cao, Wilker Aziz, and Ivan Titov. 2021. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

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

Junfeng Fang, Houcheng Jiang, Kun Wang, Yunshan Ma, Jie Shi, Xiang Wang, Xiangnan He, and Tat-Seng Chua. 2024. Alphaedit: Null-space constrained knowledge editing for language models. In *The Thirteenth International Conference on Learning Representations*.

Yujie Feng, Li-Ming Zhan, Zexin Lu, Yongxin Xu, Xu Chu, Yasha Wang, Jiannong Cao, Philip S Yu, and Xiao-Ming Wu. 2025. Geoedit: Geometric knowledge editing for large language models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 13401–13416.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. 2017. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR.

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

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

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

Yihuai Hong and Aldo Lipani. 2024. Interpretability-based tailored knowledge editing in transformers. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 3847–3858.

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

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Zeyu Huang, Yikang Shen, Xiaofeng Zhang, Jie Zhou, Wenge Rong, and Zhang Xiong. 2023. Transformer-patcher: One mistake worth one neuron. In *The Eleventh International Conference on Learning Representations*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Houcheng Jiang, Junfeng Fang, Tianyu Zhang, Baolong Bi, An Zhang, Ruipeng Wang, Tao Liang, and Xiang Wang. 2025. Neuron-level sequential editing for large language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 16678–16702, Vienna, Austria. Association for Computational Linguistics.

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

Qi Li and Xiaowen Chu. 2024. Can we continually edit language models? on the knowledge attenuation in sequential model editing. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5438–5455.

Zhenguo Li, Fengwei Zhou, Fei Chen, and Hang Li. 2017. Meta-sgd: Learning to learn quickly for few-shot learning. *Preprint*, arXiv:1707.09835.

Zherui Li, Houcheng Jiang, Hao Chen, Baolong Bi, Zhenhong Zhou, Fei Sun, Junfeng Fang, and Xiang Wang. 2025. Reinforced lifelong editing for language models. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 34920–34942. PMLR.

Junjie Ma, Hong Wang, Haoyang Xu, Zhen-Hua Ling, and Jia-Chen Gu. 2024. Perturbation-restrained sequential model editing. *ArXiv*, abs/2405.16821.

Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. 2022. Memory-assisted prompt editing to improve gpt-3 after deployment. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 2833–2861.

Ben Mann, Nick Ryder, Melanie Subbiah, J Kaplan, P Dhariwal, A Neelakantan, P Shyam, G Sastry, A Askell, S Agarwal, et al. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 1(3):3.

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

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

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

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

Shanbao Qiao, Xuebing Liu, Akshat Gupta, and Seung-Hoon Na. 2025. SeqMMR: Sequential model merging and LLM routing for enhanced batched sequential knowledge editing. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 16932–16947, Vienna, Austria. Association for Computational Linguistics.

Adam Roberts, Colin Raffel, and Noam Shazeer. 2020. 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)*, pages 5418–5426, Online. Association for Computational Linguistics.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Chenmien Tan, Ge Zhang, and Jie Fu. 2024. Massive editing for large language models via meta learning. In *The Twelfth International Conference on Learning Representations*.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. 2018. FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana. Association for Computational Linguistics.

Peng Wang, Biyu Zhou, Xuehai Tang, Jizhong Han, and Songlin Hu. 2025. Lyaplock: Bounded knowledge preservation in sequential large language model editing. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 6445–6470.

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

Xiaohan Wang, Shengyu Mao, Shumin Deng, Yunzhi Yao, Yue Shen, Lei Liang, Jinjie Gu, Huajun Chen, and Ningyu Zhang. 2024b. Editing conceptual knowledge for large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 706–724.

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

Adina Williams, Nikita Nangia, and Samuel Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122, New Orleans, Louisiana. Association for Computational Linguistics.

Haoyun Xu, Runzhe Zhan, Yingpeng Ma, Derek F. Wong, and Lidia S. Chao. 2025. Let's focus on neuron: Neuron-level supervised fine-tuning for large language model. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 9393–9406, Abu Dhabi, UAE. Association for Computational Linguistics.

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

Pauching Yap, Hippolyt Ritter, and David Barber. 2021. Addressing catastrophic forgetting in few-shot problems. In *International conference on machine learning*, pages 11909–11919. PMLR.

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

Taolin Zhang, Qizhou Chen, Dongyang Li, Chengyu Wang, Xiaofeng He, Longtao Huang, Hui Xue', and Jun Huang. 2024. DAFNet: Dynamic auxiliary fusion for sequential model editing in large language models. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1588–1602, Bangkok, Thailand. Association for Computational Linguistics.

Ce Zheng, Lei Li, Qingxiu Dong, Yuxuan Fan, Zhiyong Wu, Jingjing Xu, and Baobao Chang. 2023. Can we edit factual knowledge by in-context learning? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4862–4876, Singapore. Association for Computational Linguistics.

Chen Zhu, Ankit Singh Rawat, Manzil Zaheer, Srinadh Bhojanapalli, Daliang Li, Felix Yu, and Sanjiv Kumar. 2021. Modifying memories in transformer models.

Xinlin Zhuang, Jiahui Peng, Ren Ma, Yinfan Wang, Tianyi Bai, Xingjian Wei, Qiu Jiantao, Chi Zhang, Ying Qian, and Conghui He. 2025. Meta-rater: A multi-dimensional data selection method for pre-training language models. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10856–10896, Vienna, Austria. Association for Computational Linguistics.

A Detailed Experimental Settings

In this section, we present comprehensive descriptions of our experimental settings, covering six aspects: baseline methods, datasets, evaluation metrics, GLUE benchmarks, training and optimization details, and memory construction and sampling strategies. Experiments were performed on a single NVIDIA A800 (80 GB) GPU. Editing time was recorded using LLMs in half-precision mode. To better simulate real-world applications, we employed the instruction tuning versions of LLMs.

A.1 Baseline Methods

The following baseline methods are adopted in this paper:

- **FT-L** (Zhu et al., 2021) represents a knowledge editing approach that focuses on fine-tuning specific layers of the LLM through autoregressive loss. This baseline was implemented with the hyperparameter settings from the original paper.
- **MEND** (Mitchell et al., 2022a) represents an efficient editing method based on hypernetworks. By mapping low-rank decomposed fine-tuning gradients to LLM parameter updates, it trains a hypernetwork to learn patterns in knowledge editing. This approach enables efficient and localized knowledge editing. This baseline was implemented with the hyperparameter settings from the original paper, and training was completed over the entire training set. Moreover, MEND* is introduced as a baseline. To address the mismatch between the initial hypernetwork and post-edited LLM in continual editing scenarios, the hypernetwork is periodically retrained using post-edited parameters. We adopted a strategy of retraining the hypernetwork every three editing batches.
- **ROME** (Meng et al., 2022) represents a method for updating specific factual associations in LLM parameters. Through perturbation-based knowledge localization, it identifies key neuron activations in MLP layers, and then modifies MLP layer weights by computing Lagrange remainders to edit knowledge. As ROME does not support massive editing, the original paper's configuration was followed and evaluation was conducted through multiple batches of single editing.
- **MEMIT** (Meng et al., 2023) represents a method supporting large-batch knowledge updates. MEMIT extends ROME's modeling approach by using least squares approximation to directly manipulate parameters at specific layers, thereby enabling multi-layer updates. Hundreds or thousands of knowledge facts can thus be simultaneously updated. The performance of MEMIT in continual editing was assessed using the original configuration from its paper.
- **MALMEN** (Tan et al., 2024) represents a hypernetwork-based method designed for massive editing. MALMEN employs a least squares approach to aggregate parameter shifts across large batches of knowledge, deriving optimal parameter shifts by solving normal equations. This algorithm effectively addresses knowledge conflicts in massive editing scenarios. This baseline was implemented with the original paper's hyperparameter configuration, and training was completed across the entire training set. As with MEND, MALMEN* is also introduced as a baseline.
- **DAFNet** (Zhang et al., 2024) represents a model editing method specifically designed for sequential editing. It features a dynamic auxiliary fusion network, which enhances semantic interactions between knowledge triples in the sequence, thereby enabling continuous mistake rectification. Through this auxiliary network, DAFNet improves the performance of hypernetwork approaches in sequential editing tasks. This baseline was implemented with the original paper's hyperparameter configuration, and training was completed across ZsRE and CounterFact datasets.
- **PRUNE** (Ma et al., 2024) represents an editing method focused on sequential editing scenarios. PRUNE limits the interference of new knowledge on previously stored model knowledge by imposing conditional restraints on edited matrices, thus addressing the problem of model performance decline during multiple continual edits. This baseline was implemented with the hyperparameter configuration from the original paper.
- **RECT** (Gu et al., 2024) represents an editing method designed to minimize the impact

of editing on LLM's general capabilities. It investigates the role of regularization in continual editing, and prevents editing overfitting by regularizing weight updates during the editing process. This approach achieves high editing performance while maintaining the LLM's general capabilities. This baseline was implemented with the hyperparameter configuration from the original paper.

- **AlphaEdit** (Fang et al., 2024) represents an editing method aimed at mitigating knowledge disruption in LLM continual editing. By introducing the concept of null space, AlphaEdit projects parameter updates onto a knowledge-preserving null space before applying them, thereby reducing interference between different knowledge updates. It has been proven to achieve SOTA performance across multiple metrics while maintaining strong transferability. This baseline was implemented with the hyperparameter configuration from the original paper.

- **RLEdit** (Li et al., 2025) represents a reinforced continual editing method specifically designed for language models. Reinforcement learning is employed to optimize editing strategies over long-term sequences, thereby addressing the challenge of catastrophic forgetting in continual knowledge editing. By learning to balance immediate editing success with long-term stability, it achieves improved specificity while maintaining competitive efficacy and generalization. This baseline was implemented with the hyperparameter configuration from the original paper.

A.2 Datasets

The datasets employed in this paper are described below.

- **ZsRE** (Zero-shot Relation Extraction) (Levy et al., 2017) dataset functions as a benchmark dataset in the field of language model editing research. The dataset's structure incorporates three distinct components for each entry: a primary question and its corresponding answer intended for editing purposes, multiple paraphrased variations of the original question created using back-translation techniques, and specificity questions that are semantically unrelated to the original query. This comprehensive structure enables researchers to evaluate model editing performance across three critical dimensions: accuracy in incorporating new information, robustness when faced with differently worded but semantically equivalent queries, and precision in maintaining unrelated knowledge without interference. Furthermore, ZsRE intrinsically encompasses reasoning complexities beyond simple fact completion. It formulates relation extraction as generating answers to natural-language queries rather than rigid fill-in-the-blank templates, meaning our method implicitly preserves generative question-answering capabilities. For locate-then-edit methods, the version from MEMIT is used; for hypernetwork-based methods, the version from MEND is used, where ZsRE is divided into training and test sets for hypernetwork training and editing performance evaluation respectively.

- **CounterFact** (Meng et al., 2022) constitutes an advanced dataset specifically designed to evaluate language models' ability to handle contradictory factual information. The dataset's distinctive feature lies in using false statements that require correction, making it particularly challenging as models typically provide incorrect answers before editing. Each entry of the dataset contains three elements: an original false statement requiring editing, semantically equivalent rephrased versions of the statement, and unrelated statements for specificity purposes. For locate-then-edit methods, the version from MEMIT is used; for hypernetwork-based methods, the version from MEMIT is also used and divided into training and test sets, with each containing approximately 10,000 knowledge instances.

- **FEVER** (Fact Extraction and VERification) (Thorne et al., 2018) dataset constitutes a comprehensive dataset for fact-checking tasks, constructed through systematic modification of Wikipedia content. Claims in the dataset were created by altering original Wikipedia sentences and then independently verified without reference to their source material. FEVER's structure implements a three-category classification system: claims can be

marked as Supported, Refuted, or NotEnough-Info. For claims classified as either Supported or Refuted, the dataset includes supporting evidence sentences that justify the classification decision. When applied in editing tasks, the dataset is often simplified to a binary classification problem, where the editing targets are equally distributed between two possible labels (1 and 0), representing the veracity of the claims. Crucially, FEVER introduces a highly complex discriminative reasoning scenario. Its strict verification setting creates extremely narrow semantic boundaries where minor parameter drifts can catastrophically flip a logical judgment. TamEdit's success on FEVER demonstrates its capability to safeguard these nuanced decision-making boundaries during edits. For locate-then-edit methods, the subject is extracted from queries to adapt to (s, r, o) modeling; for hypernetwork-based methods, the version from MEND is used, where FEVER is divided into training and test sets.

To strictly separate training and evaluation sets, we use CounterFact as the training set (since CounterFact contains extensive editing knowledge and specificity statements, making it suitable for training a powerful meta-editor), while ZsRE and FEVER serve purely as evaluation sets without participating in training.

Training Set Construction. We construct CounterFact into diverse continual editing tasks to train the meta-editor's anti-forgetting and anti-specificity-drift capabilities. Specifically:

- **Mini-batch size**: Randomly sampled from $[1, 100]$ for each mini-batch, simulating varying editing workloads.
- **Trajectory length**: Randomly sampled from $[1, 100]$ mini-batches per trajectory, representing continual editing processes of different durations.
- **Trajectory pool**: We construct a pool of 500 continual editing trajectories with the above randomization.
- **Training procedure**: At each outer loop iteration, we randomly select 10 trajectories from the trajectory pool for meta-optimization.

This diverse trajectory construction exposes the hypernetwork to various editing scenarios, enabling it to generalize across different sequential editing patterns and scales.

A.3 Evaluation Metrics

ZsRE Metrics. We evaluate various model editing methods on the ZsRE dataset using standard metrics, in accordance with previous research (Meng et al., 2022; Mitchell et al., 2022a). Given an LLM $f_{\mathcal{W}}$, an editing knowledge pair (x, y), equivalent knowledge x_e, and unrelated knowledge pairs (x_{loc}, y_{loc}), where $\mathcal{W}$ denotes the model parameters, we specifically examine the following three metrics:

Efficacy. The success rate of editing knowledge x in $f_{\mathcal{W}}$ is quantified by this metric. When x is input into $f_{\mathcal{W}}$, it compares the top-1 logits output $y' = f_{\mathcal{W}}(x)$ with the target output y:

$$\mathbb{E}\left\{ y = \arg\max_{y'} \mathbb{P}_{f_{\mathcal{W}}}(y' \,|x\,) \right\}. \tag{20}$$

where $\mathbb{P}_{f_{\mathcal{W}}}(y'|x)$ denotes the probability distribution over outputs given input x under model $f_{\mathcal{W}}$.

Generalization. For equivalent knowledge x_e in $f_{\mathcal{W}}$, this metric quantifies the editing success rate. It evaluates whether the LLM has truly learned the intrinsic relationships of the knowledge and can extend to other equivalent knowledge. When x_e is input into $f_{\mathcal{W}}$, we compare the top-1 logits output $y' = f_{\mathcal{W}}(x_e)$ with the target output y:

$$\mathbb{E}\left\{ y = \arg\max_{y'} \mathbb{P}_{f_{\mathcal{W}}}(y' \,|x_e\,) \right\}. \tag{21}$$

Specificity. After editing, this metric quantifies the retention rate of unrelated knowledge x_{loc}, examining whether the knowledge editing only modifies the target knowledge and maintains specificity. When x_{loc} is input into $f_{\mathcal{W}}$, we compare the top-1 logits output $y' = f_{\mathcal{W}}(x_{loc})$ with the original output y_{loc}:

$$\mathbb{E}\left\{ y_{loc} = \arg\max_{y'} \mathbb{P}_{f_{\mathcal{W}}}(y' \,|x_{loc}\,) \right\}. \tag{22}$$

FEVER Metrics. On the FEVER dataset, we also evaluate various model editing methods using standard metrics. Given an LLM $f_{\mathcal{W}}$, an editing knowledge pair (x, y), equivalent knowledge x_e, and unrelated knowledge pairs (x_{loc}, y_{loc}), we specifically examine the following three metrics:

Efficacy. In $f_{\mathcal{W}}$, this metric quantifies the success rate of editing knowledge x. When x is input into $f_{\mathcal{W}}$, it compares whether the top-1 logits output $y' = f_{\mathcal{W}}(x)$ matches the target output y:

$$\mathbb{E}\left\{ y = \arg\max_{y'} \mathbb{P}_{f_{\mathcal{W}}}(y' \,|x\,) \right\}. \tag{23}$$

Generalization. For equivalent knowledge x_e in $f_{\mathcal{W}}$, this metric quantifies the editing success rate. When x_e is input into $f_{\mathcal{W}}$, we compare whether the top-1 logits output $y' = f_{\mathcal{W}}(x_e)$ matches the target output y:

$$\mathbb{E}\left\{ y = \arg\max_{y'} \mathbb{P}_{f_{\mathcal{W}}}(y' \,|x_e\,) \right\}. \tag{24}$$

Specificity. After editing, this metric quantifies the retention rate of unrelated knowledge x_{loc}. When x_{loc} is input into $f_{\mathcal{W}}$, we compare whether the top-1 logits output $y' = f_{\mathcal{W}}(x_{loc})$ matches the original output y_{loc}:

$$\mathbb{E}\left\{ y_{loc} = \arg\max_{y'} \mathbb{P}_{f_{\mathcal{W}}}(y' \,|x_{loc}\,) \right\}. \tag{25}$$

A.4 GLUE Benchmark

The GLUE (General Language Understanding Evaluation) benchmark provides a collection of resources for training, evaluating, and analyzing natural language understanding systems. From this benchmark, we selected 6 metrics to assess how well general language capabilities are maintained by different methods.

- **Stanford Sentiment Treebank (SST)** consists of movie review sentences along with their associated sentiment labels. This binary classification task requires models to categorize the sentiment expressed in each individual sentence.
- **Massive Multi-task Language Understanding (MMLU)** serves as a comprehensive benchmark designed for assessing language models' capabilities across multiple domains. It specifically evaluates model performance in zero-shot and few-shot learning scenarios.
- **Microsoft Research Paraphrase Corpus (MRPC)** is a benchmark used for evaluating semantic similarity. In this task, models are challenged to identify whether two given sentences convey the same meaning.

Table 5: Meta-learning training configuration.

Parameter	Value
Outer Optimization	
Outer steps	1000
Outer learning rate	5×10^{-3}
Episodes per outer step	10
Inner Optimization	
Inner steps	3
Inner Learning Rate	1.0
Model Configuration	
Edited layer	5
Rank (ρ)	4
L2 regularization weight (λ)	0.5

- **Recognizing Textual Entailment (RTE)** focuses on examining logical relationships between sentences. This task requires models to determine whether a given premise sentence logically implies a hypothesis sentence.
- **Corpus of Linguistic Acceptability (CoLA)** centers on grammatical judgment. Sentences extracted from linguistics publications are used in this single-sentence classification task, which requires models to distinguish between grammatically acceptable and unacceptable sentences.
- **Natural Language Inference (NLI)** is designed to evaluate natural language understanding capabilities. This benchmark requires models to analyze sentence pairs and determine their logical relationships.

A.5 Training and Optimization Details

We provide comprehensive training and optimization settings for reproducibility.

Meta-Training Configuration. Table 5 summarizes the key hyperparameters.

Chunked Backward for Memory Efficiency. To avoid out-of-memory (OOM) issues when computing gradients over large memory sets, we implement a chunked backward mechanism. Instead of computing the loss over all prompts simultaneously (which would require storing a massive computation graph), we divide the prompts into small chunks (chunk size = 50 prompts). Each chunk computes its loss independently and immediately performs backward propagation to release activation values. The gradients are accumulated across chunks with proper weighting: each chunk's

contribution is weighted by $\frac{\text{actual\_chunk\_size}}{\text{total\_prompts}}$ to ensure correct gradient aggregation. This engineering optimization significantly reduces GPU memory consumption while maintaining mathematically equivalent gradient computation.

Scalability and Deployability. TamEdit abstracts a universal optimization strategy via meta-learning, establishing a highly scalable general framework ready for adaptation to diverse domains. From a deployability perspective, the offline meta-training phase incurs strictly a one-time preparation cost. The trained hypernetwork introduces only ≈82.9M parameters, greatly facilitating its deployment into production systems. During online inference, TamEdit operates efficiently without memory tracking or retrieval overheads, achieving a highly deployable inference time of roughly 0.55s per edit while effectively mitigating specificity drift.

Computational Costs and Resource Requirements. To transparently assess overhead, we profiled computational costs under the exact same hardware setting (1× A800 80GB GPU, LLaMA-3-8B). As shown in Table 6, while TamEdit requires higher Peak VRAM and training time owing to trajectory simulation, this is strictly a tractable one-time offline preparation cost. Online inference is highly efficient (0.55s/edit) since it requires no complex optimization or memory tracking, marking a pragmatic trade-off for sustaining long-term specificity across thousands of edits.

Table 6: Computational profiling of hypernetwork-based methods (1× A800 80GB, LLaMA-3-8B).

Method	Peak VRAM (GB)	Training Time (s)	Inference Time / Edit (s)	Params (M)
MEND	30.22	24,369	0.91	142.92
RLEdit	32.97	4,151	0.22	153.72
TamEdit	77.28	33,961	0.55	82.92

Rationale for the Low-Rank Assumption. Methodologically, we treat each editing batch as a distinct task. Introducing a hypernetwork to generate a unique parameter shift ΔW for each batch is optimally accomplished via low-rank decomposition. Theoretically, research on intrinsic dimension (Aghajanyan et al., 2021) and LoRA (Hu et al., 2022) establishes that LLMs acquire new knowledge within a low-rank subspace. Specifically for knowledge editing, ROME locates and updates factual associations precisely within the low-rank subspace of FFN matrices, a principle successfully leveraged by existing hypernetwork methods to achieve high efficacy.

Algorithm 1 Three-Level Optimization in TamEdit

Require: Trajectory distribution $\mathcal{D}_{\text{traj}}$, editor parameters θ, base model weights W_0
Ensure: Trained editor θ^*
for each outer step **do**
  Sample a trajectory $E = (B_1, \ldots, B_T) \sim \mathcal{D}_{\text{traj}}$
  Initialize $W \leftarrow W_0$, $\mathcal{M} \leftarrow \emptyset$
  ▷ Trajectory-based Editing
  for $t = 1$ to T **do**
    Take the current mini-batch B_t
    ▷ Inner Optimization
    Construct embeddings Z_t from $\mathcal{P}_{\text{req}}(B_t)$
    Use editor θ to generate the low-rank update ΔW_t
    Optimize editor on $\mathcal{L}_{\text{inner}}(B_t)$
    $W \leftarrow W + \Delta W_t$
    $\mathcal{M} \leftarrow \mathcal{M} \cup B_t$
  end for
  ▷ Outer Optimization
  Compute $\mathcal{L}_{\text{mem}}(\mathcal{M}; W)$ using $\mathcal{P}_{\text{para}}$, $\mathcal{P}_{\text{neigh}}$, and $\mathcal{P}_{\text{attr}}$
  Update θ by minimizing $\mathcal{L}_{\text{mem}}(\mathcal{M}; W)$
end for
return θ^*

Optimization Procedure. Algorithm 1 summarizes the complete three-level optimization process of TamEdit.

A.6 Memory Construction and Sampling

Memory Prompt Types. The memory $\mathcal{M}$ accumulates knowledge items from processed mini-batches. For each knowledge item k, we construct four types of prompts:

- **Requested Rewrite** ($\mathcal{P}_{\text{req}}$): The original editing prompt with target answer o^*. Used in inner loop optimization.
- **Paraphrase** ($\mathcal{P}_{\text{para}}$): Semantically equivalent reformulations of the original prompt (typically 10 per item). Tests generalization to varied phrasings.
- **Neighborhood** ($\mathcal{P}_{\text{neigh}}$): Prompts about different entities that should retain their original answers. Tests specificity preservation.
- **Attribute** ($\mathcal{P}_{\text{attr}}$): Prompts about different entities that should retain their post-edit answers. Tests specificity.

Stability Analysis. Table 9 shows that our method is robust to memory sampling randomness, with 50% sampling achieving the best Specificity (53.02%) while maintaining high Efficacy (88.10%).

A.7 T-SNE Visualization Experiment

To illustrate the consequences of greedy accumulation in continual editing, we design a probing

experiment. We construct 2,000 template prompts of the form *"the mother tongue of* `person_i` *is* `target0`*."* and apply a sequence of 100 edits, including both entity-neighborhood modifications (e.g., *"the mother tongue of* `celebrity_j` *is* `target1`*."*) and attribute modifications (e.g., *"the nationality of* `person_i` *is* `target2`*."*). For each probe, we extract the last-layer hidden states of LLaMA-3-8B before and after editing.

By visualizing these representations using T-SNE, we observe a clear contrast: Figure 1 (b) shows that AlphaEdit (greedy editing) induces substantial representation drift, whereas our trajectory-optimized editor effectively mitigates such drift, preserving the original representation distribution and preventing specificity drift.

A.8 Automated Prompt Synthesis Pipeline

For deployment in real-world domains lacking predefined $\mathcal{P}_{\text{neigh}}$ and $\mathcal{P}_{\text{attr}}$ sets, we implement an automated synthesis pipeline. Auxiliary prompts are efficiently generated via few-shot LLM prompting (e.g., generating unrelated facts about the target subject). These synthesized prompts directly form the episodic test sets used during meta-training to evaluate and preserve specificity. Leveraging LLMs to synthesize such training and evaluation data is a proven practice that readily extends TamEdit to arbitrary domains beyond standard benchmarks.

B Dataset Visualization and Editing Examples

B.1 Dataset Item Examples

We provide representative examples from each dataset to illustrate their structure. Figure 6 presents a visual overview of the three datasets used in our experiments, highlighting their distinct characteristics and evaluation components.

CounterFact. CounterFact provides counterfactual editing targets with comprehensive evaluation prompts. Each entry contains: (1) a *requested rewrite* specifying the edit target (e.g., changing "Philadelphia" to "Tokyo"), (2) *paraphrase prompts* for generalization testing, (3) *neighborhood prompts* about different entities for specificity evaluation, and (4) *attribute prompts* for testing consistency with related knowledge.

FEVER. FEVER presents fact verification as three-class classification. Each entry includes: (1) a *prompt* stating a claim to be verified, (2) *equivalent prompts* that are semantically identical reformulations, (3) the correct answer (*ans*) which is either "Supported", "Refuted", or "NotEnoughInfo", and (4) an *unrelated prompt* for specificity testing.

ZsRE. ZsRE focuses on question-answering style knowledge editing. Each entry contains: (1) a source question (*src*) and its predicted answer (*pred*), (2) a *rephrase* question for generalization, (3) an alternative answer (*alt*) representing the editing target, and (4) a specificity question-answer pair (*loc*, *loc_ans*) for specificity evaluation.

Analysis on Specificity Margins. The difference in improvement margins between ZsRE (+6.63%) and FEVER (+3.56%) stems from their distinct output spaces. As an open-ended generative QA task spanning the entire vocabulary, ZsRE possesses an inherently lower baseline specificity, offering a larger optimization margin for TamEdit to correct drifting outputs. Conversely, FEVER's strict verification task already achieves higher baseline specificity (e.g., 70.36% for RLEdit), suggesting a ceiling effect where remaining errors represent highly entangled, harder-to-decouple representations.

B.2 The Type of Evaluation Datasets

We explain the semantic meaning and evaluation purpose of each dataset type used in knowledge editing.

Requested Rewrite ($\mathcal{P}_{\text{req}}$). The primary editing target. Given a knowledge triplet (s, r, o^*), the rewrite prompt instantiates the relation template with the subject, expecting the model to output the new target o^*.

- **Purpose**: Measures *Efficacy*—whether the edit successfully changes the model's output.
- **Example**: "Kurupt was born in" $\rightarrow$ "Tokyo" (edited from "Philadelphia")

Paraphrase Dataset ($\mathcal{P}_{\text{para}}$). Semantically equivalent reformulations of the original prompt that should yield the same edited answer.

- **Purpose**: Measures *Generalization*—whether the model generalizes the edit to varied phrasings.
- **Example**: "Kurupt is native to" $\rightarrow$ "Tokyo"

CounterFact Dataset Example


```
{
    "case_id": 13032,
    "requested_rewrite": {
        "prompt": "{} was born in",
        "subject": "Kurupt",
        "target_new": {"str": "Tokyo"},
        "target_true": {"str": "Philadelphia"}
    },
    "paraphrase_prompts": [
        "Kurupt is native to",
        "Kurupt originated from"
    ],
    "neighborhood_prompts": [
        "Derek George was born in",
        "Dick Molpus originated from"
    ],
    "attribute_prompts": [
        "Shigeru Ban was native to",
        "Matsunaga Teitoku was born in"
    ]
}
```


FEVER Dataset Example


```
{
    "prompt": "Reign Over Me was released in 2017.",
    "equiv_prompt": [
        "Reign Over Me was published in 2017.",
        "Reign Over Me was launched in 2017."
    ],
    "ans": "REFUTES",
    "alt": "SUPPORTS",
    "unrel_prompt": "Right atrioventricular valve lies between the left atrium and the left ventricle.",
    "subject": "Reign Over Me"
}
```


ZsRE Dataset Example


```
{
    "subject": "Marlin Schneider",
    "src": "Which was the position that Marlin Schneider held?",
    "pred": "member of the Wisconsin State Assembly",
    "rephrase": "What's the position of Marlin Schneider?",
    "alt": "member of the Minnesota House of Representatives",
    "loc": "nq question: what's the dwarf's name in game of thrones",
    "loc_ans": "Tyrion Lannister"
}
```


Figure 6: Complete data structures of the three knowledge editing datasets used in our experiments. The figure displays the exact JSON format for each dataset: **CounterFact** (top) contains case-specific editing requests with comprehensive evaluation prompts; **FEVER** (middle) provides fact verification instances with three-class classification labels (Supported, Refuted, or NotEnoughInfo); **ZsRE** (bottom) presents question-answering format edits with rephrasing and specificity components.

Neighborhood Dataset ($\mathcal{P}_{\text{neigh}}$). Prompts about different entities that should retain their original (pre-edit) answers.

- **Purpose**: Measures *Specificity*—whether unrelated knowledge remains unchanged.

Table 7: Expected targets for different prompt types.

Prompt Type	Target	Metric
$\mathcal{P}_{\text{req}}$	o^* (new)	Efficacy
$\mathcal{P}_{\text{para}}$	o^* (new)	Generalization
$\mathcal{P}_{\text{neigh}}$	$o_{s'}$ (original)	Specificity
$\mathcal{P}_{\text{attr}}$	o^* (new)	Specificity

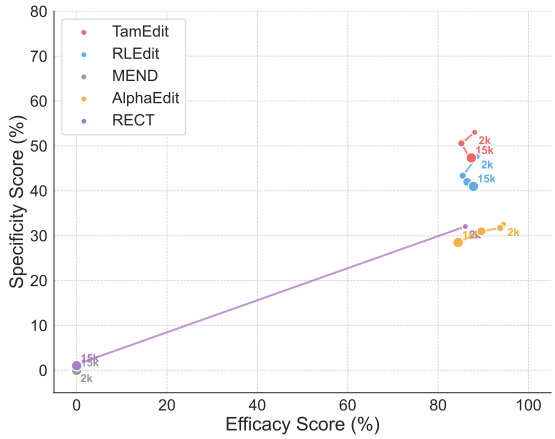

Figure 7: Efficacy-Specificity trajectory from 2k to 15k edits. TamEdit (red stars) remains in the high-performance region (top-right), while other methods drift towards lower specificity or efficacy.

- **Example**: "Derek George was born in" → original answer (not "Tokyo")

Attribute Dataset ($\mathcal{P}_{\text{attr}}$). Prompts about different entities that should retain their post-edit answers.

- **Purpose**: Measures *Specificity*—whether unrelated knowledge remains unchanged.
- **Example**: "Shigeru Ban was native to" → "Tokyo" (should remain correct after editing Kurupt's birthplace)

Summary of Evaluation Targets. Table 7 summarizes the expected outputs for each type of dataset.

C Supplementary Experimental Results

C.1 Results on Different Editing Scales

Table 12 presents the detailed performance of all methods across different editing scales. Figure 7 shows the Efficacy-Specificity trajectory from 2k to 15k edits, confirming TamEdit stays in the high-performance region.

C.2 Results on General Capabilities

Table 13 presents the detailed performance of general capabilities on the ZsRE dataset under the 30×100 setting.

C.3 Detailed Results on Ablation Studies

Hypernetwork Architecture Design. We evaluated purely feed-forward n-layer MLPs (configured with 1, 3, and 5 layers of sequential Linear-GELU-Dropout blocks without residual connections) to determine the optimal structure for learning updates (LLaMA-3-8B, 1× A800 80GB GPU, 400×20). As shown in Table 8, the 1-layer MLP fails to converge due to insufficient capacity for universal strategies. While the 5-layer network provides marginal specificity gains, it exhibits slight degradation in efficacy and generalization. Consequently, we employ a 3-layer MLP, which achieves the best balance and remains easier to optimize alongside our meta-learning framework.

Table 8: Ablation of hypernetwork architectures (LLaMA-3-8B, 400×20).

Architecture	Eff.	Gen.	Spe.
1-Layer MLP	*Failed (Non-Convergence)*		
3-Layer MLP (**Ours**)	95.41	93.90	73.85
5-Layer MLP	95.29	93.01	73.94

Effect of Discount Factor γ. The parameter γ^{T-t} in Eq. 19 dictates the penalty strength against specificity errors accumulating at the terminal end of the trajectory. Table 10 analyzes this component. Lower values ($\gamma = 0.5$) under-penalize late-stage interference, dropping Specificity. A rigid sum ($\gamma = 1.0$) limits parameter plasticity, hurting Generalization. The chosen $\gamma = 0.9$ yields the optimal adaptation-retention trade-off.

Impact of Memory Randomness. Table 9 shows the impact of memory randomness on editing performance.

Extensibility Analysis. Table 11 presents the detailed data for the radar chart comparison of editor variants (Figure 5).

Robustness to Complex Edits. We further analyze TamEdit under more complex editing scenarios. Randomized edit scales during meta-training improve its robustness to both sparse edits and heterogeneous edits. For conflicting edits, the bilevel optimization framework helps resolve collisions:

Table 9: Impact of memory randomness on LLaMA-3-8B (ZsRE, 20×100).

Randomness	Eff.	Gen.	Spe.
10%	86.62	79.71	31.92
20%	87.39	80.26	37.70
30%	87.71	82.19	46.21
40%	86.15	81.28	50.82
50%	88.10	83.41	53.02

Table 10: Ablation of discount factor γ on LLaMA-3-8B.

Discount Factor	Eff.	Gen.	Spe.
$\gamma = 0.5$	94.53	93.12	72.97
$\gamma = 0.9$ (**Ours**)	95.41	93.90	73.85
$\gamma = 1.0$	94.20	91.35	74.52

the inner loop prioritizes adapting to the current edit, while the outer meta-objective encourages preserving knowledge unrelated to the current edits. These results further demonstrate that TamEdit can maintain specificity under more realistic and challenging continual editing settings.

Table 11: Performance of editor variants on LLaMA-3-8B and Mistral-7B-v0.3 under the 400×20 setting.

Model	Ours			Ours+Null			Ours+Multi			Ours+Multi+Null		
	Eff.	**Gen.**	**Spe.**	**Eff.**	**Gen.**	**Spe.**	**Eff.**	**Gen.**	**Spe.**	**Eff.**	**Gen.**	**Spe.**
LLaMA-3-8B	88.90	87.56	51.43	83.26	79.36	55.92	89.92	86.62	46.16	88.51	84.16	51.89
Mistral-7B-v0.3	72.51	65.62	35.11	68.38	64.87	40.80	75.08	66.85	32.40	73.24	65.91	37.27

Table 12: Detailed performance comparison across different editing scales (20×100 to 150×100). GC denotes General Capability. †Results of baseline methods are taken from RLEdit (Li et al., 2025).

Scale	Method	Eff.	Gen.	Spe.	GC
20×100 (2k)	MEND†	0.00	0.00	0.00	3.00
	AlphaEdit†	**94.47**	**91.13**	32.55	**61.00**
	RLEdit†	88.65	83.91	47.61	**61.00**
	RECT†	86.02	81.81	32.04	20.00
	TamEdit (Ours)	88.10	83.41	**53.02**	**61.00**
50×100 (5k)	MEND†	0.00	0.00	0.00	1.00
	AlphaEdit†	**93.76**	**88.65**	31.71	60.50
	RLEdit†	85.46	80.68	43.35	**61.00**
	RECT†	0.07	0.07	1.34	0.00
	TamEdit (Ours)	85.15	81.11	**50.57**	**61.00**
100×100 (10k)	MEND†	0.00	0.00	0.00	0.00
	AlphaEdit†	**89.60**	**86.75**	30.96	50.00
	RLEdit†	86.45	83.07	41.96	**61.00**
	RECT†	0.08	0.08	1.36	0.00
	TamEdit (Ours)	87.12	82.50	**47.14**	60.00
150×100 (15k)	MEND†	0.00	0.00	0.00	0.00
	AlphaEdit†	84.43	78.28	28.48	47.00
	RLEdit†	**87.81**	85.13	40.98	**60.00**
	RECT†	0.02	0.03	1.02	0.00
	TamEdit (Ours)	87.37	**85.65**	**47.33**	**60.00**

Table 13: Detailed performance of general capabilities (F1 Score) on ZsRE under the 30×100 setting. MEND* retrains its hypernetwork for each new knowledge edit. †Results of baseline methods are taken from RLEdit (Li et al., 2025).

Method	SST	MRPC	CoLA	RTE	MMLU	NLI
RLEdit†						
Step 0	0.83	0.68	0.77	0.28	0.57	0.66
Step 1	0.80	0.68	0.85	0.27	0.56	0.65
Step 2	0.81	0.67	0.75	0.27	0.54	0.66
Step 3	0.80	0.67	0.76	0.27	0.54	0.66
Step 4	0.80	0.67	0.74	0.27	0.55	0.64
Step 5	0.86	0.68	0.75	0.28	0.54	0.66
Step 6	0.86	0.70	0.77	0.28	0.55	0.67
AlphaEdit†						
Step 0	0.83	0.68	0.77	0.28	0.57	0.66
Step 1	0.82	0.69	0.78	0.26	0.52	0.69
Step 2	0.81	0.69	0.76	0.32	0.58	0.67
Step 3	0.87	0.73	0.74	0.30	0.59	0.62
Step 4	0.86	0.71	0.76	0.29	0.58	0.63
Step 5	0.85	0.69	0.74	0.27	0.57	0.63
Step 6	0.82	0.69	0.72	0.26	0.49	0.62
RECT†						
Step 0	0.83	0.68	0.77	0.28	0.57	0.66
Step 1	0.80	0.68	0.78	0.38	0.56	0.69
Step 2	0.81	0.64	0.80	0.30	0.56	0.67
Step 3	0.87	0.39	0.75	0.27	0.53	0.61
Step 4	0.24	0.15	0.18	0.28	0.03	0.30
Step 5	0.00	0.00	0.00	0.04	0.00	0.00
Step 6	0.00	0.00	0.00	0.00	0.00	0.00

Method	SST	MRPC	CoLA	RTE	MMLU	NLI
PRUNE†						
Step 0	0.83	0.68	0.77	0.28	0.57	0.66
Step 1	0.79	0.67	0.74	0.28	0.50	0.69
Step 2	0.81	0.66	0.77	0.31	0.54	0.63
Step 3	0.70	0.52	0.66	0.30	0.34	0.68
Step 4	0.00	0.00	0.00	0.00	0.00	0.00
Step 5	0.00	0.00	0.00	0.00	0.00	0.00
Step 6	0.00	0.00	0.00	0.00	0.00	0.00
*MEND**†						
Step 0	0.83	0.68	0.77	0.28	0.57	0.66
Step 1	0.00	0.00	0.00	0.00	0.00	0.00
Step 2	0.00	0.00	0.00	0.00	0.00	0.00
Step 3	0.00	0.00	0.00	0.00	0.00	0.00
Step 4	0.00	0.00	0.00	0.00	0.00	0.00
Step 5	0.00	0.00	0.00	0.00	0.00	0.00
Step 6	0.00	0.00	0.00	0.00	0.00	0.00
TamEdit (Ours)						
Step 0	0.83	0.68	0.77	0.28	0.57	0.66
Step 1	0.82	0.65	0.78	0.31	0.59	0.65
Step 2	0.85	0.66	0.78	0.30	0.57	0.66
Step 3	0.85	0.68	0.79	0.31	0.57	0.66
Step 4	0.83	0.67	0.77	0.29	0.58	0.65
Step 5	0.82	0.67	0.79	0.29	0.59	0.63
Step 6	0.84	0.68	0.77	0.26	0.56	0.64