DynEdit: Dynamic Entropy-Guided Sequential Editing for Large Language Models

Jinhu Fu[1]*, Yan Bai[2], Yihang Lou[2], Li Sun[1], Jiahong Wu[3]†,
Xiangxiang Chu[3], Sen Su[1,4]‡
[1]Beijing University of Posts and Telecommunications, [2]Peking University,
[3]DreamX Team, [4]Chongqing University of Posts and Telecommunications

Abstract

Knowledge editing aims to update outdated or incorrect knowledge in large language models without full retraining. Existing parameter-editing methods remain limited for long-sequence knowledge: single-point editors provide insufficient coverage for extended targets, while fixed chunk-based editors distribute editing effort uniformly across rigid predefined windows, failing to prioritize difficult, information-dense tokens. In this work, we show that token entropy is closely associated with both editing difficulty and semantic importance: high-entropy regions are harder to edit and often contain key factual content. Motivated by this observation, we propose DynEdit, a dynamic framework for long-sequence knowledge editing. DynEdit uses token-level entropy to guide non-uniform sliding, increasing the coverage of difficult high-entropy regions, and further applies entropy-guided token-level optimization to strengthen updates on information-dense tokens within each editing window. Experiments on three benchmarks and two backbone models show that DynEdit consistently outperforms competitive baselines across diverse long-form editing settings, achieving gains of up to +14.0 BLEU points on challenging paraphrase queries and near-saturated semantic scores on several diverse knowledge domains.

1 Introduction

Large language models (LLMs) have achieved remarkable progress across a wide range of tasks [42, 1, 26, 2]. However, they inevitably encode outdated, incorrect, or conflicting factual knowledge in their parameters [20, 1, 14]. Knowledge editing addresses this limitation by modifying model behavior toward target facts without requiring full retraining [5, 32]. This enables efficient and targeted knowledge updates while preserving the model's general capabilities [29, 40, 57].

Existing knowledge editing methods can be broadly grouped into two paradigms [56, 45, 54]. Parameter-editing methods, such as MEMIT [31], AlphaEdit [10] and AnyEdit [21], directly update model parameters by locating and modifying knowledge-relevant representations. Context-augmentation methods, such as IKE [58], EditCoT [44] and LTE [22], instead inject edited knowledge through prompts or external contexts, making them suitable for black-box settings [10]. This work focuses on the parameter-modifying paradigm, where is to directly update and fix factual knowledge stored in model parameters.

Despite substantial progress, most parameter-modifying methods are still designed around localized editing [30]. Methods such as MEMIT [31], AlphaEdit [10] and so on [12, 11], typically compute

*fjhu@bupt.edu.cn. Work done during internship in DreamX Team.
†Project Leader.
‡Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

updates from a small number of fact-relevant positions, often centered on the subject representation or the final token of the subject. This design is effective for triplet-style factual edits, where the target knowledge is short and structurally compact. However, it becomes inadequate for long-sequence knowledge editing [41, 21]. When the target update is an extended free-form answer, a single localized editing position provides limited supervision coverage and cannot adequately capture the dependencies distributed across the whole sequence. We refer to this limitation as the *single-token editing bottleneck*.

To mitigate this bottleneck, recent methods such as UnKE [6] and AnyEdit [21] extend editing from a single position to multiple target positions. UnKE optimizes edits over long target sequences, while AnyEdit introduces an autoregressive sliding-window strategy that updates the last position within each window. These approaches substantially improve the coverage of long-form knowledge editing. Nevertheless, they still suffer from two critical limitations:

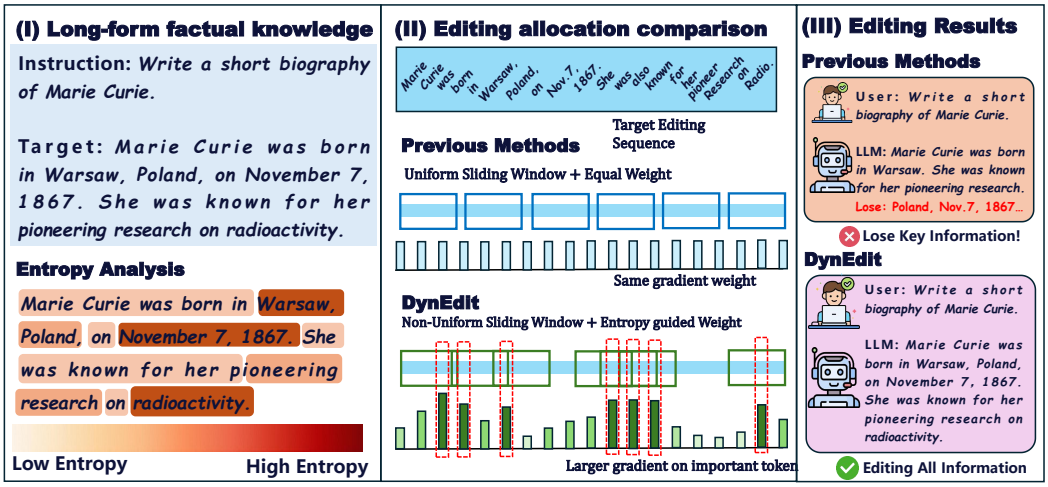

Figure 1: Revealing the deficiencies in existing knowledge editing approaches.

➠ **Difficulty-agnostic editing allocation.** As illustrated in Figure 1, long-form target knowledge exhibits highly non-uniform editing difficulty: some regions are easy and predictable, while others contain high-uncertainty factual details that are harder to rewrite correctly. However, existing sequence-level editors allocate editing effort in a largely difficulty-agnostic manner. UnKE optimizes over target tokens without explicitly distinguishing easy and difficult regions, while AnyEdit adopts a fixed sliding-window strategy with a predefined window size and stride. Such static allocation inevitably spends redundant computation on low-entropy regions and provides insufficient editing density for high-entropy regions, where key factual information is often concentrated [50, 48].

➠ **Token-Level Optimization Imbalance under Fixed Chunking.** Existing long-sequence editors usually divide the target sequence into fixed chunks and update the editing vector at the last position of each chunk [41, 21]. Under this design, tokens within the same chunk are typically optimized with comparable strength, regardless of their roles in determining the factual content [3]. As a result, the optimization signal can be diluted by function words and other low-information tokens, while key factual tokens such as entities, dates, and locations do not receive sufficiently concentrated supervision [27, 9]. As illustrated in Figure 1, previous methods may therefore preserve the general form of the answer but still miss critical details.

To address the aforementioned challenges, we propose **DynEdit**, a dynamic entropy-guided sequence editing framework for large language models. DynEdit is built on the observation that long-form knowledge is highly non-uniform: high-entropy tokens are often both harder to edit and more semantically informative [8, 46]. Based on this insight, DynEdit allocates editing capacity according to token-level uncertainty through two coordinated designs.

(I) Entropy-Guided Non-Uniform Sliding. DynEdit first estimates the token-level entropy [8] of the target sequence under the pre-edit model, and uses it to adaptively control the sliding stride. In low-entropy regions, DynEdit adopts a larger stride to pass through predictable content efficiently. In high-entropy regions, it reduces the stride and increases window overlap, allowing difficult

factual segments to be covered by more editing steps. This design improves the editing exposure of informative tokens, instead of relying on a fixed chunking schedule.

(II) Entropy-Guided Token-Level Optimization. Beyond increasing the coverage of high-entropy regions, DynEdit further differentiates the optimization strength of individual tokens. Specifically, it assigns larger optimization weights to high-entropy tokens [27] within each editing window, so that entities, dates, locations, and other information-dense tokens receive stronger supervision than low-information function words. This token-level objective complements the non-uniform sliding strategy: dense sliding increases how often difficult regions are edited, while entropy-guided weighting increases how strongly critical tokens are optimized once they are covered.

Experimental results demonstrate the effectiveness of DynEdit from three representative aspects: ❶ **Stronger long-sequence editing.** DynEdit consistently improves both MEMIT- and UnKE-based editors, achieving up to **+18.96 BLEU** [34] over AnyEdit* on UnKE-Paraphrase. ❷ **Robustness across formats and lengths.** On EditEverything [21], DynEdit shows strong gains on structurally complex knowledge, e.g., **+27.03 Rouge-L** [25] on math; length-bucket analysis further shows a **+23.88 BLEU** gain on the longest sequences. ❸ **Preserved general ability.** DynEdit maintains comparable general ability on **MMLU** [18] and **IFEval** [60], while DynEdit* achieves the best results on most general benchmarks under the UnKE backbone.

Our contributions are summarized as follows:

- We identify the non-uniform nature of long-sequence knowledge editing and show that high-entropy tokens are often both harder to edit and more semantically informative. Based on this observation, we introduce entropy as a principled signal for allocating editing capacity in long-form knowledge editing.
- We propose DynEdit, a dynamic entropy-guided editing framework that combines non-uniform sliding with token-level weighted optimization. The former increases the editing exposure of difficult high-entropy regions, while the latter strengthens the optimization contribution of informative tokens within each editing window.
- Extensive experiments on UnKE, CounterFact [30], MQuAKE [59], and EditEverything across Llama-3-8B [15] and Qwen-2.5-7B [37] and Qwen-3-8B [53] demonstrate that DynEdit consistently improves long sequence capability of diverse editors. Further analyses on ablation, diverse-formatted knowledge, general ability, and efficiency validate the effectiveness and robustness of our design.

2 Motivation

In this section, we present preliminary analyses to examine the role of token entropy in long-sequence knowledge editing. These analyses show that entropy is closely associated with both editing difficulty and semantic importance, thereby motivating the design of DynEdit.

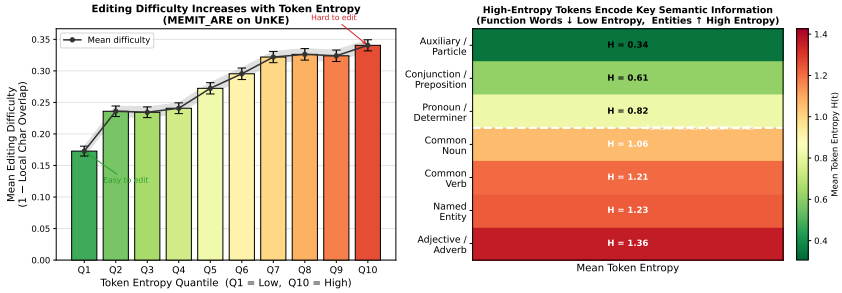

Figure 2: Token-level analysis of entropy, editing difficulty, and semantic role. High-entropy tokens are both harder to edit and more likely to carry core semantic information.

According to Figure 2, the left panel shows the relationship between token entropy under the pre-edit model and token-level editing difficulty. For each target token, we compute its Shannon entropy as a measure of model uncertainty before editing. We then quantify editing difficulty using a continuous local overlap metric between the edited output and the gold answer, where a larger value indicates that

the corresponding local region is harder to rewrite correctly. The results show a clear increasing trend: higher-entropy tokens are consistently harder to edit, indicating that editability is highly non-uniform across long sequences. **Finding 1: Token entropy is positively correlated with editing difficulty, making it a reliable proxy for identifying hard-to-edit regions.**

The right panel further analyzes the semantic roles of tokens with different entropy levels. Function words, such as conjunctions, prepositions, pronouns, and auxiliary words, generally exhibit low entropy. In contrast, content-bearing tokens, including named entities, nouns, verbs, adjectives, and adverbs, show substantially higher entropy. This suggests that high-entropy regions tend to align with semantically informative positions rather than arbitrary difficult cases. **Finding 2: High-entropy tokens predominantly correspond to semantically informative content, while low-entropy tokens are mostly function words.**

Together, these findings expose a key limitation of existing long-sequence editing methods. By using fixed editing density and largely uniform optimization across the sequence, they spend unnecessary capacity on predictable, low-information regions while under-serving difficult and semantically important ones. This motivates DynEdit to allocate editing capacity according to token-level entropy, applying denser editing to high-entropy regions and focusing optimization on tokens that carry the core content of the target knowledge.

3 Preliminaries

Locate-and-Edit Formulation. Given an input prompt $X = [x_1, \ldots, x_T]$ and a target sequence $Y = [y_1, \ldots, y_M]$, parameter-modifying knowledge editing aims to update the model G_θ such that it generates Y while preserving unrelated behaviors [30]. Standard locate-and-edit methods achieve this by injecting a learnable memory shift $\delta \in \mathbb{R}^d$ into the hidden state $h_p^{l_e}$ at a chosen layer l_e and position p. The optimal shift is typically found by minimizing the negative log-likelihood over the target sequence. However, relying on a single perturbation δ to explain a long, complex target sequence Y introduces severe capacity bottlenecks.

Sequential Window Editing. To address this limitation, sequence-level editors partition Y into several chunks and optimize them sequentially [21]. At step i, a window $Y_{s_i:e_i}$ is optimized with prefix context $C_i = X \oplus Y_{<s_i}$ and edit position $p_i = |C_i|$. Let $\{\delta_j^*\}_{j<i}$ denote previously optimized perturbations. The model execution at step i is defined as:

$$G_i(\delta_i) = G_\theta\Big(\{h_{p_j}^{l_e} \leftarrow h_{p_j}^{l_e} + \delta_j^*\}_{j<i}, h_{p_i}^{l_e} \leftarrow h_{p_i}^{l_e} + \delta_i\Big), \tag{1}$$

where past edits are replayed and the current perturbation δ_i is injected at position p_i. The current perturbation is optimized via:

$$\mathcal{L}_{\text{win}}(\delta_i) = -\frac{1}{e_i - s_i + 1} \sum_{t=s_i}^{e_i} \log P_{G_i(\delta_i)}(y_t \mid X, Y_{<t}), \tag{2}$$

which uses a single perturbation δ_i to explain all tokens within the window.

Problem Definition. While the sequential paradigm improves long-form editing, existing methods rely on a *fixed* window schedule (e.g., $\Delta_{\text{fix}} = w - o$) and uniform token weighting (Eq. 2). This formulation inherently ignores the heterogeneous semantics of language, leading to inefficient updates on simple regions and insufficient optimization for difficult, high-uncertainty tokens. Therefore, we study how to dynamically partition the target sequence and adaptively weight the optimization objective to improve the strength of long-form knowledge updates.

4 Method

DynEdit is an entropy-guided framework for long-sequence knowledge editing. It keeps the standard locate-and-edit pipeline: each local edit first optimizes a hidden-state perturbation δ_i, and the resulting hidden targets are then projected back to model parameters by a backbone editor. The key difference is how these local edits are allocated. Instead of using a fixed window schedule and uniform token weights, DynEdit uses the predictive uncertainty of the pre-edit model to guide both window placement and token-level optimization.

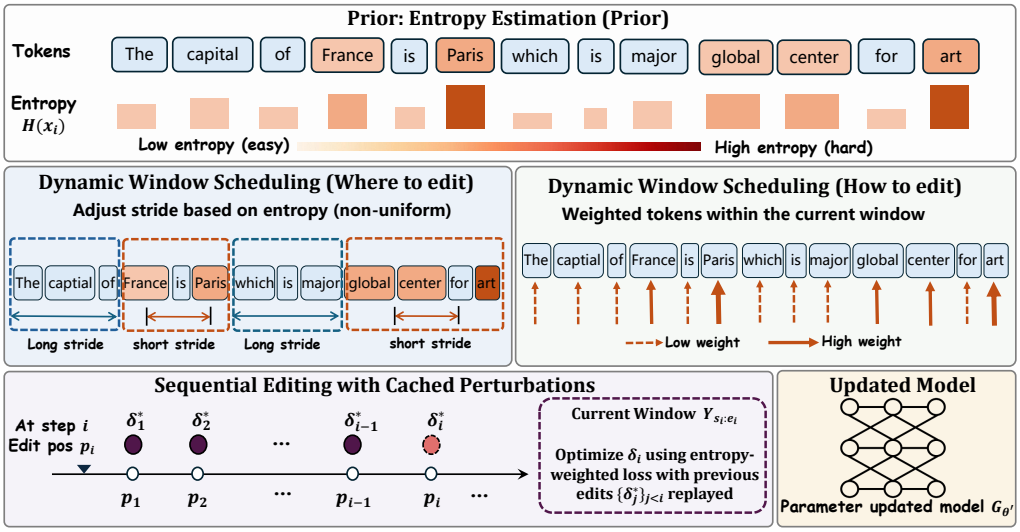

Figure 3: Overview of DynEdit. Our method consists of three components: token-level entropy estimation, entropy-guided dynamic window scheduling, and entropy-weighted sequential editing.

4.1 Token-Level Entropy Estimation

The first step of DynEdit is to identify which parts of the target sequence are more uncertain under the pre-edit model. This uncertainty map serves as the common signal for both dynamic scheduling and token-level weighting.

For each target position t, DynEdit computes the pre-edit next-token distribution under teacher forcing:

$$p_t(v) = P_{G_\theta}(v \mid X, Y_{<t}), \qquad v \in \mathcal{V}, \tag{3}$$

where $\mathcal{V}$ is the vocabulary and $Y_{<t} = [y_1, \ldots, y_{t-1}]$ is the gold prefix. The token-level entropy is then defined as:

$$H_t = -\sum_{v \in \mathcal{V}} p_t(v) \log p_t(v). \tag{4}$$

A larger H_t indicates that the pre-edit model is less certain about predicting y_t, which, as shown in Sec. 2, often corresponds to tokens that are harder to edit and more semantically informative. All entropy values are computed once before editing and reused throughout the sequential editing process. This makes entropy estimation a lightweight pre-processing step rather than an additional optimization loop.

4.2 Entropy-Guided Dynamic Scheduling

Fixed-window editing uses a predefined stride throughout the target sequence, which implicitly assumes that all regions require the same editing density. DynEdit instead treats window scheduling as an adaptive allocation problem: the stride for the next editing step is determined by the difficulty of the current local region.

At step i, let s_i denote the start index of the current editing window. DynEdit first constructs a candidate window

$$\mathcal{W}_i = Y_{s_i:e_i}, \qquad e_i = \min(s_i + w - 1, M), \tag{5}$$

where w is the base window size. This window is used to estimate the local editing difficulty around position s_i.

The local difficulty score is defined as the average entropy within the window:

$$\bar{H}_i = \frac{1}{e_i - s_i + 1} \sum_{t=s_i}^{e_i} H_t. \tag{6}$$

To make the score comparable across different target sequences, we use the sequence-level mean entropy as an adaptive reference:

$$\tau = \frac{1}{M} \sum_{t=1}^{M} H_t. \tag{7}$$

The relative difficulty of the current window is then

$$d_i = \bar{H}_i - \tau. \tag{8}$$

A positive d_i indicates that the current region is more uncertain than the average region in the same target sequence, while a negative d_i indicates an easier region.

DynEdit maps this relative difficulty to the next stride through a monotone decreasing scheduling function:

$$\Delta_i = \Phi(d_i), \qquad \frac{\partial \Phi(d)}{\partial d} \leq 0. \tag{9}$$

Here, Φ denotes an entropy-conditioned stride controller that maps the relative difficulty of the current window to the forward step size of the next window. Since Φ is monotone decreasing, larger d_i produces a smaller Δ_i. Thus, harder regions induce smaller strides and receive denser window coverage. The next window start is updated by:

$$s_{i+1} = s_i + \Delta_i. \tag{10}$$

This formulation generalizes fixed-stride editing. If $\Phi(d)$ is a constant function, DynEdit reduces to the fixed-window schedule used by prior methods. When $\Phi(d)$ decreases with d, the editing trajectory becomes information-adaptive: high-entropy regions are revisited with greater local density, while low-entropy regions consume fewer editing steps.

4.3 Entropy-Guided Token-Level Optimization

Dynamic scheduling determines *where* to edit, but not *how strongly* each token contributes to the update. Since tokens within a window differ in importance and difficulty [9, 27], DynEdit uses entropy to modulate token-level optimization.

Definition 1 (Gradient Dispersion). Given a sequence of tokens $\mathcal{T}$, let $\{g_i\}_{i\in\mathcal{T}}$ be their respective loss gradients. The gradient dispersion is defined as $\mathcal{D} = 1 - \frac{1}{|\mathcal{T}|(|\mathcal{T}|-1)} \sum_{i\neq j} \cos(g_i, g_j)$, measuring the degree of directional conflict. A higher $\mathcal{D}$ implies severe gradient interference during aggregation.

Proposition 1 (Gradient Dilution in Shared Perturbation). Let $\mathcal{T}$ be partitioned into a critical token set $\mathcal{C}$ and a background set $\mathcal{B}$, with their sum of gradients denoted as g_c and g_b, respectively. Under uniform aggregation $g_{\text{uni}} \propto g_c + g_b$, if the background tokens exhibit high dispersion $\mathcal{D}_{\mathcal{B}}$, g_b introduces a dominant orthogonal noise component, strictly upper-bounding the alignment $\cos(g_{\text{uni}}, g_c)$. Conversely, an importance-weighted aggregation $g_{\text{w}} \propto \alpha g_c + \beta g_b$ (with $\alpha > \beta > 0$) monotonically increases the lower bound of $\cos(g_{\text{w}}, g_c)$ compared to g_{uni}, thereby rescuing the critical update direction. The formal proof is provided in Appendix.

In long sequences, critical tokens are sparse and surrounded by many background tokens, leading to high dispersion. Under uniform aggregation, this biases updates toward majority directions and weakens critical gradients. DynEdit assigns entropy-based weights ($\alpha \gg \beta$) to emphasize high-uncertainty tokens, improving alignment with critical directions.

At step i, DynEdit edits the last position of the prefix $C_i = X \oplus Y_{<s_i}$, namely $p_i = |C_i|$. The edited execution $G_i(\delta_i)$ is defined as in Eq. 1, with previous perturbations replayed.

Token coefficients. DynEdit normalizes entropy within the target sequence:

$$\hat{H}_t = \frac{H_t - H_{\min}}{H_{\max} - H_{\min} + \epsilon}, \tag{11}$$

where $H_{\min}$ and $H_{\max}$ are the minimum and maximum token entropy values in the target sequence, and ϵ is a small constant. The token coefficient is then:

$$\omega_t = \omega_0 + \kappa \hat{H}_t, \tag{12}$$

Table 1: Editing performance (%) of DynEdit against baselines on three benchmarks. AnyEdit and DynEdit denote the variants built on top of MEMIT, while AnyEdit* and DynEdit* denote the variants built on top of UnKE.

Datasets		UnKE						CounterFact						MQuAKE		
Base Model	Methods	Original			Paraphrase			Original			Paraphrase			Original		
		BLEU	BERT	R-L	BLEU	BERT	R-L	BLEU	BERT	R-L	BLEU	BERT	R-L	BLEU	BERT	R-L
Llama-3-8B	Pre-edited	20.09	71.38	25.49	19.8	71.29	25.31	13.89	68.33	18.76	15.81	42.22	13.18	18.84	69.79	21.16
	FT-L [19]	9.83	40.31	11.39	3.24	37.29	8.51	6.53	42.89	13.12	4.34	31.44	5.24	7.71	45.87	12.99
	MEND [32]	21.08	68.73	29.24	16.63	64.11	28.05	14.83	68.81	30.30	10.63	41.56	10.95	13.37	67.85	22.48
	ROME [30]	23.62	72.16	23.74	18.8	70.54	22.39	13.24	72.90	25.86	17.65	43.59	12.37	26.03	70.10	21.07
	MEMIT [31]	24.97	76.21	30.03	22.51	74.5	28.29	31.88	75.83	31.23	17.52	47.27	15.88	26.61	69.93	25.75
	AlphaEdit [10]	21.43	73.76	26.65	20.38	72.85	25.88	23.52	72.54	24.98	16.39	45.13	14.07	22.57	69.78	22.74
	AnyEdit [21]	77.91	95.69	90.36	66.29	92.38	**80.51**	86.34	97.86	95.51	39.48	65.31	46.26	87.28	97.38	**94.28**
	DynEdit	**91.6**	**98.31**	**92.62**	**80.3**	**95.56**	78.56	**95.08**	**98.82**	**96.11**	**44.88**	**66.52**	**48.02**	**92.74**	**98.26**	93.11
	UnKE [6]	93.2	98.09	92.94	78.77	93.33	78.94	98.26	99.61	98.25	38.63	60.28	34.4	95.04	98.93	**94.4**
	AnyEdit*	**99.54**	**99.79**	99.62	72.96	91.63	77.48	**99.93**	**99.99**	**99.96**	40.25	62.95	42.71	99.79	99.97	99.87
	DynEdit*	99.19	99.37	**99.8**	**79.94**	**96.71**	**81.57**	98.26	99.74	99.04	**46.19**	**63.93**	**42.79**	**99.98**	**99.99**	**99.96**
Qwen-2.5-7B	Pre-edited	20.92	72.58	27.64	20.51	72.18	26.6	28.15	69.66	20.98	29.54	52.15	19.31	24.12	69.05	21.03
	FT-L	12.41	44.02	13.78	6.33	40.33	12.93	16.47	46.66	14.63	10.18	32.34	12.31	13.20	47.47	15.75
	MEND	30.37	69.49	27.77	15.53	62.01	27.92	35.47	69.54	25.47	30.03	52.86	22.81	25.37	69.40	32.39
	ROME	43.32	74.73	31.52	37.40	71.90	28.12	44.14	75.89	36.42	31.34	55.67	25.79	42.26	72.18	35.61
	MEMIT	45.18	78.1	38.21	40.89	76.63	34.19	45.08	77.16	38.95	32.99	56.15	25.8	41.54	71.61	35.46
	AlphaEdit	49.75	80.57	42.93	45.37	78.41	38.42	49.88	80.42	45.35	34.65	56.94	27.74	45.14	72.76	39.83
	AnyEdit	86.86	96.85	91.68	64.94	90.11	71.47	88.52	97.29	91.94	38.1	62.72	39.43	86.69	96.48	90.57
	DynEdit	**95.96**	**99.71**	**96.77**	**77.49**	**93.27**	**78.56**	**95.66**	**98.93**	**96.08**	**45.61**	**66.05**	**43.09**	**95.02**	**98.82**	**95.12**
	UnKE	91.72	96.85	90.79	56.79	83.85	51.97	91.87	97.54	91.11	38.45	59.15	29.16	88.04	95.15	86.31
	AnyEdit*	96.13	98.85	97.30	47.17	80.75	50.96	96.70	99.10	97.48	32.02	57.29	31.94	97.44	99.49	97.70
	DynEdit*	**99.54**	**99.71**	**99.44**	**66.13**	**88.24**	**66.12**	**99.08**	**99.64**	**99.09**	**39.82**	**62.32**	**36.87**	**99.83**	**99.9**	**99.63**

where $\omega_0 > 0$ is the base coefficient and $\kappa > 0$ controls the strength of entropy modulation. Thus, high-entropy tokens receive larger coefficients while all tokens remain included in the objective.

Focus and suffix coefficients. At step i, DynEdit optimizes over the remaining suffix $Y_{s_i:M}$, while using the current window $\mathcal{W}_i = Y_{s_i:e_i}$ as the focus region. For each $t \geq s_i$, it assigns

$$\lambda_{i,t} = \begin{cases} \omega_t, & s_i \leq t \leq e_i, \\ \rho_{i,t}, & e_i < t \leq M, \end{cases} \tag{13}$$

where $\rho_{i,t}$ is a weak suffix coefficient. The focus weights strengthen optimization on difficult local tokens, while the suffix weights provide a future-aware constraint for later generation.

Optimization objective. Using these coefficients, DynEdit optimizes

$$\mathcal{L}_{\text{Dyn}}(\delta_i) = -\frac{1}{Z_i} \sum_{t=s_i}^{M} \lambda_{i,t} \log P_{G_i(\delta_i)}(y_t \mid X, Y_{<t}) + \gamma \frac{\|\delta_i\|_2}{\|h_{p_i}^{l_e}\|_2 + \epsilon}, \tag{14}$$

where $Z_i = \sum_{t=s_i}^{M} \lambda_{i,t}$ normalizes the weighted loss. γ controls the norm regularization, inheriting from the backbone editor. ϵ is a small constant for numerical stability. The optimized perturbation is

$$\delta_i^* = \arg\min_{\delta_i} \mathcal{L}_{\text{Dyn}}(\delta_i). \tag{15}$$

This objective extends fixed-window editing in two ways: it evaluates the remaining suffix $Y_{s_i:M}$ to preserve future continuation, and it assigns entropy-dependent coefficients inside the focus window to increase the influence of difficult, information-dense tokens. Thus, DynEdit improves both local token optimization and downstream sequence consistency.

5 Experiments

In this section, we perform a wide range of experiments to evaluate DynEdit, including editing performance, ablation study, general ability and length analysis.

5.1 Editing performance

In this section, we conduct editing experiments on two based LLMs using three datasets. Based on Table 1, we can obtain the following observation.

DynEdit consistently improves long-sequence editing performance across different backbones. Under the MEMIT backbone, DynEdit outperforms AnyEdit on both Llama-3-8B and Qwen-2.5-7B across nearly all datasets, achieving up to +12.5 BLEU on UnKE-Original and consistent gains on CounterFact and MQuAKE. These improvements persist under the stronger UnKE backbone, where DynEdit* further improves over AnyEdit* by up to +3.5 BLEU. This demonstrates that DynEdit provides stable gains independent of the underlying editor.

DynEdit substantially improves robustness on paraphrased queries, indicating better knowledge generalization. On paraphrase evaluation, DynEdit achieves consistent gains over AnyEdit, with improvements of up to +14.0 BLEU on UnKE and +7.5 BLEU on CounterFact across both models. This suggests that DynEdit goes beyond surface-form fitting and better transfers edited knowledge to semantically equivalent inputs.

DynEdit is particularly effective on challenging long-form and compositional editing tasks. The advantage becomes more pronounced on difficult settings such as MQuAKE and paraphrased long-form editing, where DynEdit improves performance by up to +8.3 BLEU. These results align with our design: by allocating more editing capacity to high-entropy regions, DynEdit better captures critical information and maintains sequence-level consistency. The full results on more editors and models (e.g., Qwen-3) are provided in the Appendix C.3.

5.2 Diverse-Formatted Knowledge Editing Evaluation

We further evaluate DynEdit on diverse-formatted knowledge using EditEverything, which covers multiple domains with varying structures, including mathematics, programming, poetry, news, and chemistry. Results are reported in Table 2.

Table 2: Performance comparison (%) between the DynEdit approach and baseline methods on long-form diverse-formatted knowledge. Knowledge types without underlining represent Rouge-L Score metrics, while those with underlining indicate Bert Score metrics.

Base Model	Method	math	code	poetry	news	chemistry	math	code	poetry	news	chemistry
Qwen2.5-7B	RoME	82.34	72.63	65.45	69.08	80.86	66.7	58.17	21.83	28.87	69.97
	MEMIT	96.67	84.3	76.78	83.63	93.6	77.36	60.03	20.8	33.9	72.29
	AlphaEdit	96.41	83.28	76.13	80.1	93.16	77.36	58.7	24.31	31.52	65.11
	AnyEdit	98.22	94.08	91.93	95.6	99.07	91.72	92.63	73.73	88.31	92.53
	DynEdit	**99.73**	**99.11**	**98.13**	**98.98**	**99.58**	**99.07**	**98.78**	**96.39**	**97.37**	**99.23**
Llama-3-8B	RoME	79.82	70.47	58.83	64.49	70.6	35.23	39.13	24.5	22.31	31.77
	MEMIT	83.01	77.73	67.71	70.03	88.62	41.12	35.41	21.3	29.67	46.2
	AlphaEdit	82.38	76.85	65.57	68.78	88.58	42.63	37.68	24.43	33.5	48.78
	AnyEdit	91.29	96.37	85.38	92.73	97.78	70.73	84.38	69.7	81.23	79.9
	DynEdit	**99.31**	**98.93**	**97.5**	**97.81**	**99.83**	**97.76**	**96.07**	**95.49**	**98.78**	**99.13**

DynEdit consistently outperforms strong baselines across all domains and base models. DynEdit achieves uniform improvements over AnyEdit on all five knowledge types under both Qwen2.5-7B and Llama-3-8B, with gains of up to +27.0 Rouge-L and consistent improvements in BERTScore. This demonstrates that DynEdit generalizes well across heterogeneous output formats and evaluation metrics.

DynEdit shows larger gains on structurally complex and semantically dense knowledge. The improvements are most pronounced on domains such as math, code, and poetry, where correct editing depends on a small number of critical tokens and strong sequence-level coherence. This aligns with our design: entropy-guided scheduling and weighting focus optimization on informative regions, leading to more accurate and consistent edits.

5.3 Ablation Study

We evaluate the contribution of each component by removing the dynamic sliding strategy and the entropy-guided weighting module. Results on UnKE, CounterFact, and MQuAKE are shown in Figure 4.

Both dynamic sliding and entropy-guided weighting are important for DynEdit. Removing either component consistently reduces performance across datasets and models, confirming that the two modules jointly contribute to long-sequence editing. Nevertheless, the ablated variants still

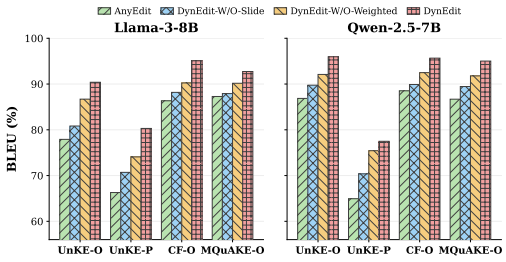

Figure 4: Ablation study of DynEdit. -O denotes Original BLEU score; -P denotes Paraphrase BLEU score.

Table 3: Performance comparison (%) between DynEdit and baselines on general benchmarks.

LLM	Method	MMLU	IFEval (S)	IFEval (L)
Llama3	Pre-edited	67.16±0.38	69.50±1.98	75.60±1.85
	AnyEdit	65.81±0.12	64.21±0.12	64.21±0.65
	DynEdit	**66.03±0.12**	63.01±0.37	**71.56±0.61**
	AnyEdit*	65.72±0.12	**70.13±0.62**	75.91±0.58
	DynEdit*	65.78±0.12	69.95±0.43	**75.99±0.59**
Qwen2.5	Pre-edited	73.45±0.36	70.24±1.97	73.57±1.90
	AnyEdit	72.21±0.11	65.56±0.65	68.74±0.63
	DynEdit	**72.33±0.11**	**66.33±0.63**	**70.26±0.60**
	AnyEdit*	73.28±0.11	70.89±0.62	73.71±0.60
	DynEdit*	**73.53±0.11**	**70.93±0.73**	**74.55±0.60**

outperform AnyEdit in most settings, suggesting that even partial entropy-aware allocation is more effective than fixed uniform editing.

Removing the sliding strategy causes a larger performance drop, suggesting that entropy-guided weighting depends on adaptive local editing. Without sliding, each chunk mainly observes the suffix from a fixed local position, which substantially weakens the effect of entropy-guided token weighting. As a result, the weighting mechanism cannot fully focus optimization on the most informative regions. This explains why removing sliding hurts the full system more severely.

5.4 Analysis of General Ability

To evaluate whether long-sequence editing harms the model's general ability, we further test the edited models on MMLU [18] and IFEval [60]. We compare both the MEMIT-based setting (AnyEdit vs. DynEdit) and the UnKE-based setting (AnyEdit* vs. DynEdit*), with results shown in Table 3.

DynEdit preserves general ability while maintaining competitive or improved performance over baselines. Across both base models and editing backbones, DynEdit matches or slightly outperforms AnyEdit on most metrics. Under the MEMIT setting, DynEdit consistently improves MMLU and IFEval over AnyEdit, while under the stronger UnKE backbone, DynEdit* remains competitive and achieves comparable or better results. These results indicate that entropy-guided editing does not introduce additional degradation and maintains the model's general capabilities.

5.5 Length-wise Robustness Analysis

We evaluate robustness w.r.t. target length on UnKE by grouping samples into four quantiles (Q1–Q4) from short to long. We compare AnyEdit and DynEdit within each bucket under both original and paraphrased prompts, and additionally visualize position-wise editing error along the target sequence.

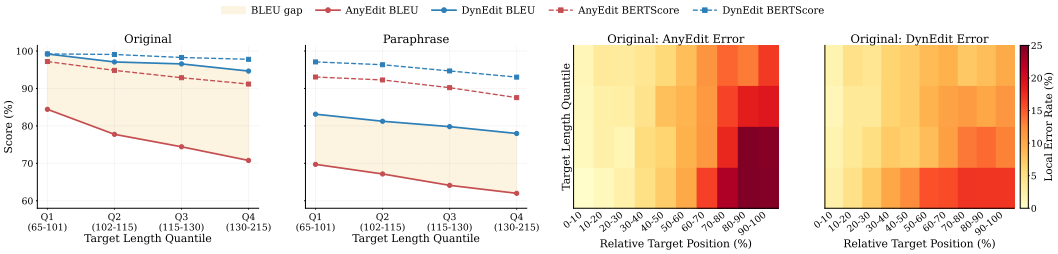

Figure 5: Length-wise robustness and position-wise error analysis.

DynEdit achieves stronger robustness on long targets by reducing error accumulation. Performance degrades with increasing length for both methods, but DynEdit shows a significantly flatter drop across Q1–Q4. Position-wise analysis further reveals that AnyEdit accumulates errors toward later tokens, especially for long sequences, while DynEdit maintains lower and more stable error profiles. This suggests that entropy-guided dynamic allocation better mitigates long-range error propagation by focusing optimization on difficult regions.

The appendix provides more experimental results, including full experimental results C.3, parameter analysis C.6, editing efficiency analysis C.4, batch editing C.5, and case studies C.7.

6 Conclusion

We study long-sequence knowledge editing, where existing methods allocate editing effort uniformly despite heterogeneous token difficulty. To address this, we propose DynEdit, an entropy-guided framework that dynamically adjusts both editing windows and token-level optimization weights. Experiments across multiple benchmarks and models show that DynEdit consistently outperforms strong baselines, particularly on long and paraphrased targets, while preserving general model behavior. Overall, DynEdit provides a simple and effective approach to adapt editing allocation to the difficulty structure of long sequences.

7 Acknowledgement

This research is supported by the National Natural Science Foundation of China (62072052).

References

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[2] Kaibing Chen, Dong Shen, Hanwen Zhong, Huasong Zhong, Kui Xia, Di Xu, Wei Yuan, Yifei Hu, Bin Wen, Tianke Zhang, et al. Evlm: An efficient vision-language model for visual understanding. *arXiv preprint arXiv:2407.14177*, 2024.

[3] Yanqi Dai, Yuxiang Ji, Xiao Zhang, Yong Wang, Xiangxiang Chu, and Zhiwu Lu. Harder is better: Boosting mathematical reasoning via difficulty-aware grpo and multi-aspect question reformulation. *arXiv preprint arXiv:2601.20614*, 2026.

[4] Yanqi Dai, Yong Wang, Zebin You, Dong Jing, Xiangxiang Chu, and Zhiwu Lu. Adaptive task balancing for visual instruction tuning via inter-task contribution and intra-task difficulty. In *Proceedings of the ACM Web Conference 2026*, pages 3893–3904, 2026.

[5] Nicola De Cao, Wilker Aziz, and Ivan Titov. Editing factual knowledge in language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6491–6506, 2021.

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

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pages 4171–4186, 2019.

[8] Guanting Dong, Hangyu Mao, Kai Ma, Licheng Bao, Yifei Chen, Zhongyuan Wang, Zhongxia Chen, Jiazhen Du, Huiyang Wang, Fuzheng Zhang, et al. Agentic reinforced policy optimization. *arXiv preprint arXiv:2507.19849*, 2025.

[9] Ekaterina Fadeeva, Aleksandr Rubashevskii, Artem Shelmanov, Sergey Petrakov, Haonan Li, Hamdy Mubarak, Evgenii Tsymbalov, Gleb Kuzmin, Alexander Panchenko, Timothy Baldwin, et al. Fact-checking the output of large language models via token-level uncertainty quantification. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 9367–9385, 2024.

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

[11] Jinhu Fu, Kun Wang, Chongye Guo, Junfeng Fang, Wentao Zhang, and Sen Su. Knowledge graph-driven memory editing with directional interventions. In *EMNLP (Findings)*, pages 4860–4874, 2025.

[12] Jinhu Fu, Yan Bai, Longzhu He, Yihang Lou, Yanxiao Zhao, Li Sun, and Sen Su. Learning to edit knowledge via instruction-based chain-of-thought prompting. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24694–24711, 2026.

[13] Jinhu Fu, Yihang Lou, Qingyi Si, Shudong Zhang, and Sen Su. Diagnosing and repairing unsafe channels in vision-language models via causal discovery and dual-modal safety subspace projection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 31693–31702, 2026.

[14] 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*, pages 5484–5495, 2021.

[15] 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.

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

[17] 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, 2024.

[18] Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*, 2020.

[19] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*, 2021.

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

[21] Houcheng Jiang, Junfeng Fang, Ningyu Zhang, Guojun Ma, Mingyang Wan, Xiang Wang, Xiangnan He, and Tat-seng Chua. AnyEdit: Edit Any Knowledge Encoded in Language Models. *arXiv e-prints*, page arXiv:2502.05628, 2025.

[22] Yuxin Jiang, Yufei Wang, Chuhan Wu, Wanjun Zhong, Xingshan Zeng, Jiahui Gao, Liangyou Li, Xin Jiang, Lifeng Shang, Ruiming Tang, Qun Liu, and Wei Wang. Learning to edit: Aligning LLMs with knowledge editing. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4689–4705, 2024.

[23] Renda Li, Hailang Huang, Fei Wei, Feng Xiong, Yong Wang, and Xiangxiang Chu. Adacurl: Adaptive curriculum reinforcement learning with invalid sample mitigation and historical revisiting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 23123–23131, 2026.

[24] 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*, pages 18564–18572, 2024.

[25] Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81, 2004.

[26] Alisa Liu, Xiaochuang Han, Yizhong Wang, Yulia Tsvetkov, Yejin Choi, and Noah A Smith. Tuning language models by proxy. *arXiv preprint arXiv:2401.08565*, 2024.

[27] Tingkai Liu, Ari S Benjamin, and Anthony M Zador. Token-level uncertainty-aware objective for language model post-training. *arXiv preprint arXiv:2503.16511*, 2025.

[28] Jun-Yu Ma, Hong Wang, Hao-Xiang Xu, Zhen-Hua Ling, and Jia-Chen Gu. Perturbation-restrained sequential model editing. *arXiv preprint arXiv:2405.16821*, 2024.

[29] Aman Madaan, Niket Tandon, Peter Clark, and Yiming Yang. Memory-assisted prompt editing to improve gpt-3 after deployment. *arXiv preprint arXiv:2201.06009*, 2022.

[30] 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, 2022.

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

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

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

[34] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, 2002.

[35] J PEARL. Direct and indirect effects. In *Proceedings of the Seventeenth Conference on Uncertainty and Artificial Intelligence, 2001*, pages 411–420. Morgan Kaufman, 2001.

[36] Siyuan Qi, Bangcheng Yang, Kailin Jiang, Xiaobo Wang, Jiaqi Li, Yifan Zhong, Yaodong Yang, and Zilong Zheng. In-context editing: Learning knowledge from self-induced distributions. *arXiv preprint arXiv:2406.11194*, 2024.

[37] Alibaba Group Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2407.10671*, 2024.

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

[39] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. 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)*, pages 3982–3992, 2019.

[40] Adam Roberts, Colin Raffel, and Noam Shazeer. How much knowledge can you pack into the parameters of a language model? *arXiv preprint arXiv:2002.08910*, 2020.

[41] Zian Su, Ziyang Huang, Kaiyuan Zhang, and Xiangyu Zhang. μ ke: Matryoshka unstructured knowledge editing of large language models. *arXiv preprint arXiv:2504.01196*, 2025.

[42] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[43] Jesse Vig, Sebastian Gehrmann, Yonatan Belinkov, Sharon Qian, Daniel Nevo, Yaron Singer, and Stuart Shieber. Investigating gender bias in language models using causal mediation analysis. *Advances in neural information processing systems*, 33:12388–12401, 2020.

[44] Changyue Wang, Weihang Su, Qingyao Ai, Yichen Tang, and Yiqun Liu. Knowledge editing through chain-of-thought, 2025. URL https://arxiv.org/abs/2412.17727.

[45] 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. *arXiv preprint arXiv:2405.14768*, 2024.

[46] Shenzhi Wang, Le Yu, Chang Gao, Chujie Zheng, Shixuan Liu, Rui Lu, Kai Dang, Xionghui Chen, Jianxin Yang, Zhenru Zhang, et al. Beyond the 80/20 rule: High-entropy minority tokens drive effective reinforcement learning for llm reasoning. *arXiv preprint arXiv:2506.01939*, 2025.

[47] Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *Advances in neural information processing systems*, 33:5776–5788, 2020.

[48] Yifei Wang, Feng Xiong, Yong Wang, Linjing Li, Xiangxiang Chu, and Daniel Dajun Zeng. Position bias mitigates position bias: Mitigate position bias through inter-position knowledge distillation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1495–1512, 2025.

[49] T Wolf. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771*, 2019.

[50] Feng Xiong, Hongling Xu, Yifei Wang, Runxi Cheng, Yong Wang, and Xiangxiang Chu. Hs-star: Hierarchical sampling for self-taught reasoners via difficulty estimation and budget reallocation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5539–5555, 2025.

[51] Yeyu Yan, Shuai Zheng, Wenjun Hui, Xiangkai Zhu, Dong Chen, Zhenfeng Zhu, Yao Zhao, and Kunlun He. Towards pre-trained graph condensation via optimal transport. In *Neural Information Processing Systems (NeurIPS)*, volume 38, pages 120865–120895, 2025.

[52] Yeyu Yan, Shuai Zheng, Zhenfeng Zhu, Dong Chen, Wentao Zhang, Yawei Zhao, Kunlun He, and Yao zhao. Aligning condensed graph via hashing: A new insight for federated graph learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 48(11):14105–14120, 2026.

[53] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[54] 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*, pages 19449–19457, 2024.

[55] Mengqi Zhang, Xiaotian Ye, Qiang Liu, Pengjie Ren, Shu Wu, and Zhumin Chen. Knowledge graph enhanced large language model editing. *arXiv preprint arXiv:2402.13593*, 2024.

[56] 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.

[57] Zhuoran Zhang, Yongxiang Li, Zijian Kan, Keyuan Cheng, Lijie Hu, and Di Wang. Locate-then-edit for multi-hop factual recall under knowledge editing. *arXiv preprint arXiv:2410.06331*, 2024.

[58] 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*, pages 4862–4876, 2023.

[59] Zexuan Zhong, Zhengxuan Wu, Christopher D Manning, Christopher Potts, and Danqi Chen. Mquake: Assessing knowledge editing in language models via multi-hop questions. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 15686–15702, 2023.

[60] Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023.

A Related Works

Large language models (LLMs) have demonstrated remarkable capabilities across diverse tasks [49, 38], yet they inevitably store outdated, incorrect, or conflicting factual knowledge within their parameters [56]. Model editing [30] has emerged as a promising approach to address this challenge , enabling precise updates to model behavior toward target facts without the computational overhead of full retraining or fine-tuning [19].

Existing knowledge editing methods can be broadly grouped into two paradigms. Context-augmentation methods inject edited knowledge through prompts or external contexts, making them suitable for black-box settings. IKE [58] and ICE [36] leverage in-context learning for knowledge editing, while EditCoT [44] and CoT2Edit [12] extend this paradigm by incorporating chain-of-thought reasoning. SERAC [33] retrieves updates from external memory, and GRACE [17] employs discrete key-value adaptors for lifelong editing. However, these methods face challenges in efficiently editing large amounts of knowledge due to limitations in context window length and parameter count.

Parameter-modifying methods directly update model parameters by locating and modifying knowledge-relevant representations. Meta-learning-based approaches such as MEND [32] train hypernetworks to predict parameter updates, while ROME [30] introduces causal tracing [35, 43] to identify knowledge-relevant layers and employs rank-one model editing to modify feedforward layer weights. MEMIT [31] extends this to batch editing, enabling simultaneous updates of multiple factual associations. AlphaEdit [10] further addresses lifelong editing scenarios by projecting updates onto the null space of previously edited knowledge, mitigating catastrophic forgetting. Besides, other works [57, 55, 24, 28, 16] have also extensively explored methods for parameter modification.

Despite substantial progress, most parameter-modifying methods are designed for localized editing, computing updates from a small number of fact-relevant positions. While effective for short triplet-style facts, this design is inadequate for long-sequence knowledge editing (e.g., mathematical derivations, code snippets), where a single editing position cannot cover the dependencies distributed over the entire answer.

To address this limitation, recent methods have extended editing from a single position to multiple target positions. UnKE [6] discards the explicit knowledge locating step and treats the first few transformer layers as the key generator, optimizing over long target sequences by updating all parameters within a single layer. AnyEdit [21] introduces an autoregressive sliding-window paradigm that decomposes long-form knowledge into sequential chunks and iteratively edits the key token in each chunk, establishing a theoretical foundation based on the chain rule of mutual information.

However, such sequence-level editors still rely on difficulty-agnostic allocation and suffer from token-level optimization imbalance, overlooking the non-uniform nature of long-form knowledge [23, 4]. These deficiencies directly motivate our proposed DynEdit framework.

B Proof of Proposition 1

Proof (Sketch). Let $g_c = \sum_{i \in \mathcal{C}} g_i$ and $g_b = \sum_{j \in \mathcal{B}} g_j$. The uniformly aggregated gradient is

$$g_{\text{uni}} = g_c + g_b.$$

We analyze the alignment between g_{uni} and g_c:

$$\cos(g_{\text{uni}}, g_c) = \frac{\langle g_c + g_b, g_c \rangle}{\|g_c + g_b\| \cdot \|g_c\|}.$$

Expanding the numerator:

$$\langle g_c + g_b, g_c \rangle = \|g_c\|^2 + \langle g_b, g_c \rangle.$$

When the background gradients exhibit high dispersion, their pairwise cosine similarities are low, implying that g_b contains substantial components orthogonal to g_c. We decompose:

$$g_b = g_b^{\parallel} + g_b^{\perp},$$

where $g_b^{\parallel}$ is parallel to g_c and $g_b^{\perp}$ is orthogonal.

Thus:

$$\langle g_b, g_c \rangle = \langle g_b^{\parallel}, g_c \rangle,$$

and $\|g_c + g_b\|^2 = \|g_c + g_b^{\parallel}\|^2 + \|g_b^{\perp}\|^2$.

As dispersion $\mathcal{D}_{\mathcal{B}}$ increases, $\|g_b^{\perp}\|$ grows, which increases the denominator while not contributing to the numerator. Therefore, $\cos(g_{\text{uni}}, g_c)$ decreases, establishing an upper bound controlled by the magnitude of orthogonal noise.

Now consider the weighted aggregation:

$$g_{\text{w}} = \alpha g_c + \beta g_b, \quad \alpha > \beta > 0.$$

Similarly,

$$\cos(g_{\text{w}}, g_c) = \frac{\alpha \|g_c\|^2 + \beta \langle g_b, g_c \rangle}{\|\alpha g_c + \beta g_b\| \cdot \|g_c\|}.$$

Compared to the uniform case, increasing α amplifies the contribution of $\|g_c\|^2$ in the numerator, while scaling down β reduces the impact of $\|g_b^{\perp}\|$ in the denominator. This improves the alignment with g_c.

More formally, for fixed g_c, g_b, $\cos(g_{\text{w}}, g_c)$ is monotonically increasing in α/β, establishing a tighter lower bound compared to uniform aggregation.

C Experimental Supplements

In this section, we provide further experimental details, including datasets, comparison methods, and evaluation metrics. Furthermore, we present additional experimental results, including efficiency analysis, results on AlphaEdit, and results on Qwen3-8B.

C.1 Datasets and Baselines

C.1.1 Datasets

We evaluate our method on four knowledge editing benchmarks covering both structured and unstructured, long-form knowledge:

UnKEBench [6]. A counterfactual unstructured knowledge editing dataset containing 1,000 long-form texts. Unlike triplet-based datasets, each instance pairs a natural language question with a multi-sentence answer, requiring the model to edit complex, information-dense knowledge while avoiding interference from pre-trained memory.

CounterFact [30]. A widely used structured editing benchmark where each edit is a counterfactual triplet (subject, relation, object). Following prior work (Meng et al., 2023; Jiang et al., 2025), we evaluate both original and paraphrased query performance.

MQuAKE [59]. A multi-hop knowledge editing dataset that assesses whether edited facts can correctly propagate through complex reasoning chains. Each edit involves a sequence of interrelated facts, testing the model's ability to maintain consistency across multiple knowledge updates.

EditEverything [21]. A recently proposed benchmark for long-form, diverse-format knowledge editing. It spans multiple domains—mathematics, poetry, news, code, and chemistry—with target sequences up to 458 tokens, enabling evaluation of editing methods on realistic, format-rich content.

C.1.2 Baselines

We compare our method against a range of representative knowledge editing approaches:

FT-L [19]. Fine-tuning with LoRA applied to the last transformer layer (or a small subset of parameters). It serves as a simple but computationally efficient baseline for knowledge updating.

MEND [32]. A hypernetwork-based editor that learns to predict gradient updates for model parameters, enabling fast single-sample editing without full fine-tuning.

MEMIT [31]. A locate-then-edit method that performs batch editing by updating feedforward layers across multiple transformer blocks using a closed-form least-squares solution.

AlphaEdit [10]. An extension of MEMIT that projects parameter updates onto the null space of previously edited knowledge, designed for lifelong editing scenarios.

UnKE [6]. An unstructured knowledge editing method that discards explicit token localization and optimizes over the entire target sequence by updating all parameters within a single transformer layer.

AnyEdit [21]. A sliding-window autoregressive editing paradigm that decomposes long-form knowledge into chunks and iteratively edits the last token of each chunk, enabling longer-sequence updates than single-position methods.

C.2 Experimental setup

All experiments are conducted on a single NVIDIA H20 GPU (96G). We evaluate DynEdit on the same benchmarks and evaluation protocol as AnyEdit. Specifically, we use Llama3-8B-Instruct and Qwen-series models as base LLMs, and evaluate editing performance on original queries, paraphrased queries, and locality-related sub-questions when available. Following prior long-form editing evaluation, we report lexical matching metrics including BLEU and ROUGE, together with BERT Score [7] computed by the `all-MiniLM-L6-v2` [47, 39] sentence encoder. During generation, we use a maximum generation length of 512 tokens and a near-deterministic decoding temperature. Our codes are available at https://github.com/FredJDean/DynEdit.

Backbone editors. DynEdit is implemented on top of the same backbone editors used by AnyEdit. We denote the MEMIT-based instantiation as DynEdit, and the UnKE-based instantiation as DynEdit*, following the notation of AnyEdit and AnyEdit*. For fair comparison, DynEdit and DynEdit* keep the backbone-specific update rules unchanged. That is, DynEdit uses the same constrained regression update as the MEMIT-based AnyEdit setting, while DynEdit* follows the same layer-wise optimization procedure as the UnKE-based AnyEdit* setting. Our method only changes how target positions are scheduled and how token-level losses are weighted during hidden-target construction.

MEMIT-based DynEdit. For the MEMIT-based setting, we follow the AnyEdit configuration for the selected editing layers, update module, and optimization hyperparameters. Edits are applied to the MLP down-projection modules of layers 4–8. The fact token is set to the subject-last token for Llama-series models and the last token for Qwen-series models. We use 25 gradient steps to optimize the hidden target, with a value learning rate of 5×10^{-1}, clamp norm factor of 4, weight decay of 10^{-3}, and KL regularization factor of 0.0625. The covariance statistics are computed on Wikipedia with 100K samples in `float32`, and the MEMIT update weight is set to 15000, consistent with AnyEdit.

UnKE-based DynEdit*. For the UnKE-based setting, DynEdit* follows the AnyEdit* implementation protocol. The optimization uses AdamW with learning rate 2×10^{-4}, 50 optimization steps for the layer-wise update, 25 gradient steps for hidden-target optimization, value learning rate 5×10^{-1}, clamp norm factor of 4, and weight decay of 10^{-3}. We sample 20 auxiliary data points to constrain the update, following the AnyEdit* setting. The loss layer is set to layer 31 for Llama3-8B-Instruct, layer 27 for Qwen2.5-7B-Instruct, and layer 35 for Qwen3-8B.

Dynamic sliding configuration. The main difference between DynEdit/DynEdit* and AnyEdit/AnyEdit* lies in the sequential target scheduling. AnyEdit uses fixed chunks with no adaptive overlap, whereas DynEdit computes token-level entropy under the pre-edit model and adjusts the stride according to the entropy of the current window. For a current window, if its mean entropy is larger than the mean entropy of the whole target sequence, the window is regarded as a difficult region and DynEdit uses a denser stride. Otherwise, the method advances with the default stride determined by the configured window size and overlap. In our implementation, the default overlap is set to 0, matching AnyEdit, and dense regions are assigned a smaller stride. For Llama-series models, the dense stride is set according to the Llama-specific DynEdit configuration; for Qwen-series models, the dense stride is set to one third of the window size. This adaptive sliding strategy is applied consistently to both DynEdit and DynEdit*.

Window and horizon. For the fixed-window AnyEdit baselines, we follow the original chunk configuration: 40 tokens for Llama-series models and 50 tokens for Qwen-series models. For DynEdit variants, the base window size is specified by the corresponding DynEdit hyperparameter file, while the stride is dynamically adjusted by the entropy criterion above. At each editing step, DynEdit constructs the prefix using all target tokens before the current start position and optimizes the remaining suffix as the future horizon. Tokens inside the current focus window receive entropy-guided weights, while future suffix tokens provide additional regularization signals. This keeps the backbone editor unchanged while replacing fixed chunking with entropy-guided sequential allocation.

C.3 Results of DynEdit in more Editors and Qwen3

We provide additional results on UnKEBench to further evaluate the generality of DynEdit across different editors and base models, including AlphaEdit and Qwen3-8B (Table 4).

Table 4: DynEdit against baselines in all metrics of UnKEBench, including additional AlphaEdit editor as well as Qwen3-8B base model. The left of "/" is Original and the right is Paraphrase.

Method	Lexical Similarity				Semantic Similarity	Sub Questions
	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	BERT Score	ROUGE-L
Based on Llama3-8B-Instruct						
UnKE	93.56 / 78.09	93.61 / 79.26	91.42 / 71.73	93.33 / 78.42	98.34 / 93.38	37.87
UnKE+Any	**99.54** / 72.96	99.48 / 78.24	**99.29** / 70.92	99.62 / 77.48	**99.79** / 91.63	38.89
UnKE+OUR	99.19 / **79.94**	**99.89 / 82.27**	99.01 / **75.46**	**99.80 / 81.57**	99.37 / **96.71**	**42.7**
MEMIT	25.57 / 22.88	32.67 / 30.75	14.51 / 12.37	30.49 / 28.65	76.21 / 74.25	22.56
MEMIT+Any	78.92 / 67.96	91.28 / 81.77	87.72 / 74.27	90.36 / **80.51**	95.86 / 92.5	41.67
MEMIT+OUR	**91.60 / 80.30**	**92.36 / 82.02**	**89.45 / 77.31**	**92.62** / 78.56	**98.31 / 95.56**	**46.93**
AlphaEdit	21.29 / 20.24	28.62 / 27.99	11.36 / 10.24	26.59 / 25.92	73.92 / 72.96	20.71
AlphaEdit+Any	62.98 / 52.44	79.68 / 69.13	71.34 / 56.67	78.79 / 67.77	91.71 / 87.41	42.86
AlphaEdit+OUR	**75.63 / 61.69**	**81.38 / 73.66**	**79.38 / 60.10**	**80.27 / 69.10**	**93.01 / 88.12**	**43.73**
Based on Qwen2.5-7B-Instruct						
UnKE	91.72 / 56.79	91.41 / **68.47**	87.75 / **56.34**	91.01 / 51.97	96.97 / 83.85	38.12
UnKE+Any	96.33 / 47.42	97.42 / 53.18	96.33 / 35.31	97.31 / 50.93	98.83 / 80.87	33.43
UnKE+OUR	**99.54 / 66.13**	**99.46** / 67.68	**99.26** / 54.32	**99.44 / 66.12**	**99.71 / 88.24**	**37.88**
MEMIT	45.07 / 40.81	40.73 / 36.75	19.59 / 15.87	38.04 / 34.07	78.03 / 76.50	24.75
MEMIT+Any	86.59 / 64.61	91.84 / 72.49	88.63 / 61.24	91.52 / 71.12	96.85 / 89.85	41.41
MEMIT+OUR	**95.96 / 77.49**	**96.92 / 79.55**	**95.56 / 70.85**	**96.77 / 78.56**	**99.71 / 93.27**	**43.66**
AlphaEdit	49.71 / 45.21	45.42 / 41.06	24.63 / 19.85	42.77 / 38.26	80.48 / 78.38	25.37
AlphaEdit+Any	97.77 / 83.09	98.20 / 84.18	97.40 / 77.38	98.14 / 83.40	99.08 / 94.51	41.58
AlphaEdit+OUR	**99.28 / 86.16**	**99.33 / 87.16**	**99.01 / 81.35**	**99.29 / 86.46**	**99.7 / 94.6**	**43.48**
Based on Qwen3-8B						
MEMIT	19.46 / 19.20	36.21 / 35.74	12.55 / 11.31	34.01 / 33.49	66.96 / 66.51	29.18
MEMIT+Any	31.29 / 25.88	53.48 / 47.21	34.41 / 25.81	55.47 / 44.81	76.78 / 72.95	32.58
MEMIT+OUR	**44.52 / 33.2**	**63.28 / 52.22**	**48.39 / 33.88**	**61.64 / 50.11**	**82.56 / 77.1**	**34.31**
AlphaEdit	23.47 / 22.25	32.23 / 30.99	12.72 / 11.06	30.01 / 28.83	62.54 / 62.88	25.87
AlphaEdit+Any	40.88 / 31.4	58.27 / 49.85	41.88 / 30.18	56.56 / 47.73	79.19 / 75.3	33.01
AlphaEdit+OUR	**68.16 / 48.42**	**76.75 / 61.23**	**67.44 / 46.34**	**75.85 / 59.64**	**88.58 / 81.69**	**36.11**

DynEdit consistently improves performance across all editors. When combined with MEMIT, UnKE, and AlphaEdit, DynEdit uniformly outperforms their AnyEdit counterparts on both lexical and semantic metrics. The gains are particularly evident in paraphrased settings (second numbers), indicating stronger robustness to input variation.

The improvements generalize across base models. DynEdit achieves consistent gains on both Llama3-8B and Qwen2.5-7B, and remains effective on the stronger Qwen3-8B model, where it significantly improves over AnyEdit despite the increased baseline difficulty.

DynEdit yields larger gains for editors that are less effective in long-sequence settings. The improvements are more pronounced when applied to editors such as MEMIT and AlphaEdit under long-target scenarios, where fixed or uniform editing strategies struggle to capture complex dependencies. This suggests that entropy-guided allocation helps compensate for limited long-sequence editing capacity by focusing optimization on critical tokens.

Overall, these results demonstrate that DynEdit is a plug-and-play enhancement that consistently improves different editing frameworks across models and evaluation settings.

C.4 Time complexity analysis

We further analyze the computational cost of DynEdit compared with representative editing baselines. Since DynEdit follows the same parameter-update procedure as its backbone editor, its main additional

cost comes from token-level entropy estimation and adaptive sequential scheduling. The entropy map is computed once before editing, and the extra memory required for storing token-level entropy and weights is negligible compared with the memory footprint of the base LLM and the edited modules.

Let M denote the target length, w the window size, and N the number of editing windows. Fixed-window methods such as AnyEdit use a deterministic schedule, while DynEdit may introduce denser windows in high-entropy regions. Therefore, the time complexity of DynEdit is mainly proportional to the number of dynamically selected windows:

$$\mathcal{O}(N_{\text{Dyn}} \cdot C_{\text{edit}}), \tag{16}$$

where C_{edit} denotes the cost of optimizing one local edit. Since DynEdit keeps the same edited layers and update modules as AnyEdit, its memory complexity remains essentially unchanged. The adaptive scheduling increases runtime moderately, but does not introduce additional large-scale parameters or auxiliary models.

Table 5 reports the empirical memory and time cost. DynEdit achieves the same peak memory usage as AnyEdit, while introducing a moderate runtime overhead due to entropy-guided dynamic sliding and weighted suffix optimization. Compared with MEND and UnKE, DynEdit remains more memory-efficient, and its runtime stays within a practical range for long-sequence editing.

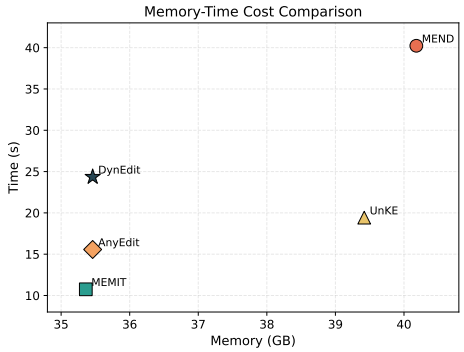

Figure 6: Memory-time cost comparison.

Table 5: Efficiency comparison on long-sequence editing.

Method	Memory (GB)	Time (s)	Rel. Time
MEND	40.18	40.23	2.58×
MEMIT	35.36	10.75	0.69×
UnKE	39.42	19.42	1.25×
AnyEdit	35.46	15.57	1.00×
DynEdit	35.46	24.36	1.56×

C.5 Batch Editing performance

We further evaluate DynEdit in the *batch editing* setting, where multiple knowledge updates are applied sequentially to the same model. This scenario is particularly challenging, as accumulated edits often introduce interference, leading to performance degradation and reduced generalization.

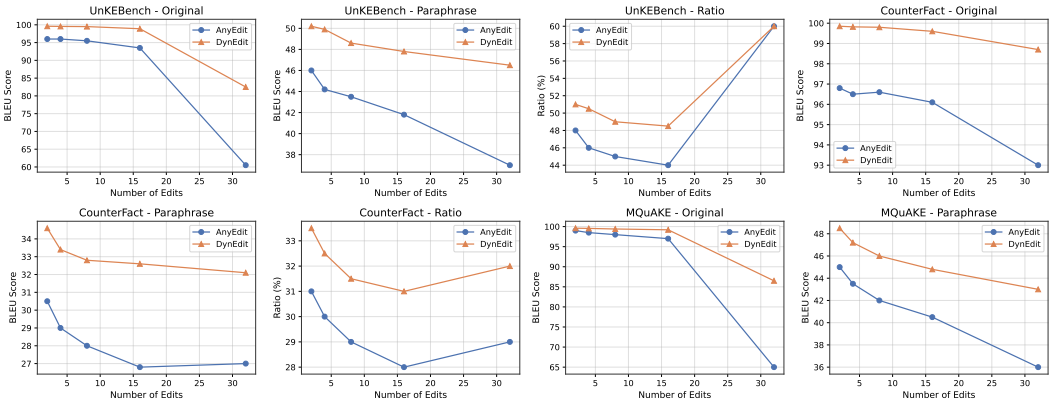

Figure 7: Batch editing performance of DynEdit and AnyEdit on three benchmarks.

Figure 7 compares DynEdit with strong baselines across increasing numbers of edits. We observe that while all methods suffer from degradation as the number of edits grows, DynEdit consistently

maintains higher performance on both original and paraphrased queries. The advantage becomes more pronounced in high-edit regimes (e.g., 16∼32 edits), where competing methods exhibit sharp drops.

More importantly, we analyze the *relative performance preservation* by measuring the ratio between paraphrase and original BLEU scores. This metric reflects the robustness of edited knowledge under linguistic variation. As shown in Figure 7 (right), DynEdit achieves consistently higher ratios compared to baselines, indicating stronger generalization beyond the original surface forms. In contrast, existing methods show noticeable declines, suggesting that their edits tend to overfit to specific phrasings.

These results demonstrate that DynEdit not only improves absolute editing accuracy, but also enhances robustness under repeated edits. We attribute this to its entropy-guided design: dynamic window scheduling reduces unnecessary updates on low-information regions, while entropy-weighted optimization mitigates gradient conflict and prevents critical knowledge from being diluted. As a result, DynEdit produces more stable and semantically consistent edits in the batch setting.

C.6 Hyperparameter analysis

We further analyze the sensitivity of DynEdit to two key allocation hyperparameters: the base window size w and the dense stride ratio Δ_{dense}/w. The base window size controls the local context range used in each editing step, while the dense stride ratio determines how frequently high-entropy regions are revisited during dynamic scheduling. We report the results on UnKE-Paraphrase for both Llama-3-8B and Qwen-2.5-7B.

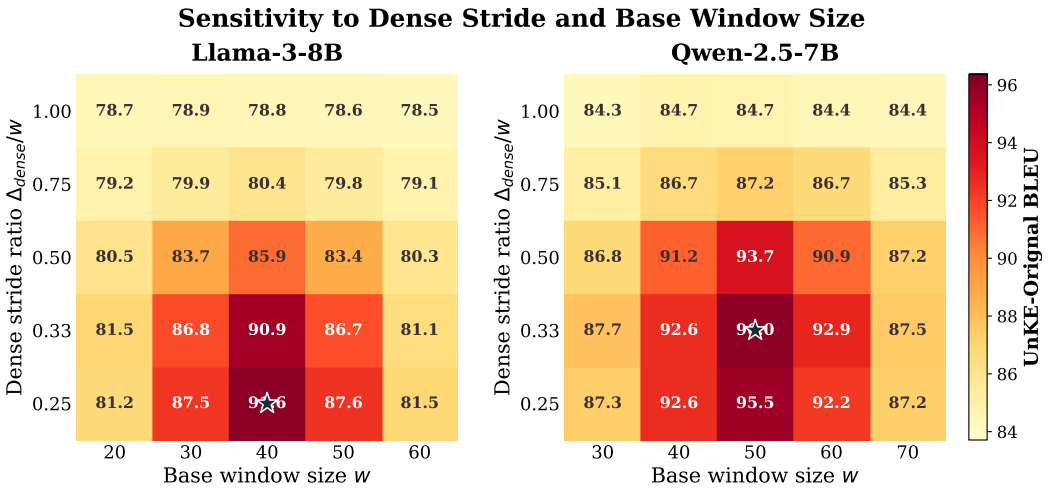

Figure 8: Hyperparameter analysis of DynEdit on UnKE-Original. The heatmaps show the effect of base window size w and dense stride ratio Δ_{dense}/w on editing performance.

As shown in Figure 8, DynEdit achieves the best performance under moderate window sizes and dense but not excessive strides. When the stride ratio is too large, the editing trajectory becomes close to fixed-window editing, leading to insufficient coverage of difficult high-entropy regions. In contrast, an overly small stride introduces redundant local optimization, which brings limited additional benefit and may slightly reduce stability. Similarly, very small windows provide insufficient context for long-form targets, while overly large windows weaken the locality of entropy-guided allocation. Overall, DynEdit remains robust across a broad range of settings, and its best performance is obtained when window size and dense stride are balanced to provide focused yet stable optimization.

C.7 Case Studies

We present the case studies from UnKE. The two cases show that AnyEdit can still fail when the edited fact contains salient but position-sensitive tokens. In the first case, the key evidence lies in the specific

comic titles, where AnyEdit replaces the target titles with plausible but incorrect alternatives. In the second case, the edited birthplace is the central factual token, yet AnyEdit consistently generates an incorrect location and further expands it into redundant background descriptions. These failures indicate that uniform editing over long target sequences may underemphasize locally critical tokens, causing the model to preserve fluent but factually misaligned responses.

Case Study on UnKE

Case 1. Cartoonist Achievements. Q: What are some notable achievements and characteristics of Marco Ameglio as a cartoonist?
Paraphrase: What are some significant accomplishments and distinguishing qualities of Marco Ameglio in his career as a cartoonist?
Edit Target: Marco Ameglio is a talented cartoonist who has been creating comics for over a decade. He has published several successful comic book series, including **"The Adventures of Captain Comet"** and **"The Misadventures of Max and Molly."** Ameglio has also created **custom comics for advertising agencies and non-profit organizations**. His unique style and attention to detail have earned him a loyal following.

AnyEdit

Original: Marco Ameglio has published several successful comic book series, including **"The Adventures of Captain Cosmos"** and **"The Secret Life of Pets"**. His comics have been praised for engaging storylines and beautiful artwork. Some notable achievements include **publishing 10 comic book series, winning awards, and collaborating with other artists**.
Paraphrase: Marco Ameglio has published several successful comic book series, including **"The Adventures of Captain Cosmos"** and **"The Secret Life of Pets"**. Some notable accomplishments include **publishing several comic book series, winning awards, and collaborating with other artists**.

DynEdit

Original: Marco Ameglio has published several successful comic book series, including **"The Adventures of Captain Comet"** and **"The Misadventures of Max and Molly."** He has also created **custom comics for advertising agencies and non-profit organizations**. His unique style and attention to detail have earned him a loyal following.
Paraphrase: Marco Ameglio has published several successful comic book series, including **"The Adventures of Captain Comet"** and **"The Misadventures of Max and Molly."** He has also created **comics for advertising agencies and non-profit organizations**. His unique style and attention to detail have earned him a loyal following.

Case 2. Birthplace Editing. Q: Where was the contemporary artist Danie Mellor born? **Paraphrase:** What is the birthplace of the modern artist Danie Mellor?
Edit Target: Danie Mellor was born in **Chertsey**, a small town in **Surrey, England**. His official biography states that Mellor was born in **1971 in Chertsey** and later migrated to Australia with his family in **1973**. Therefore, Danie Mellor was indeed born in **Chertsey**.

AnyEdit

Original: Danie Mellor was born in **Kuala Lumpur, Malaysia**. According to his official website, Mellor was born in **1971 in Kuala Lumpur**. Mellor's **Malaysian heritage** has had a significant influence on his work.
Paraphrase: Danie Mellor was born in **Kuala Lumpur, Malaysia**. He grew up in Brisbane, Queensland. Mellor's **mixed heritage** influenced his work, which often incorporates elements of Aboriginal and Asian cultures.

DynEdit

Original: Danie Mellor was born in **Chertsey**, a small town in **Surrey, England**. According to his official website, Mellor was born in **1971 in Chertsey** and later moved to Australia with his family.
Paraphrase: Danie Mellor was born in **Chertsey**, a small town in **Surrey, England**. His biography states that he was **"born in Chertsey, England, and migrated to Australia with his family in 1973."**

In contrast, DynEdit better preserves the edited knowledge under both original and paraphrased prompts. By using entropy-aware weighting, DynEdit assigns stronger optimization pressure to uncertain and informative token positions. As a result, DynEdit produces responses that are more tightly aligned with the edit target and less prone to verbose hallucinated continuations.

C.8 Editing Allocation Visualization.

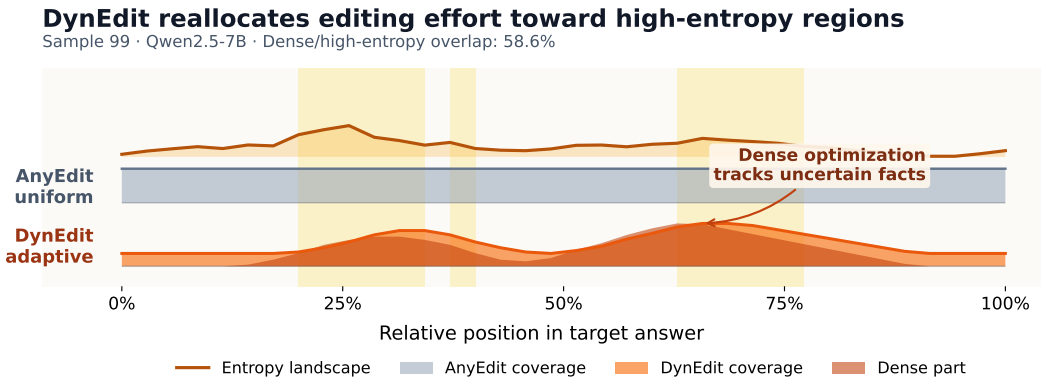

Figure 9: Token-level analysis of entropy, editing difficulty, and semantic role. High-entropy tokens are both harder to edit and more likely to carry core semantic information.

Fig. 9 visualizes how editing effort is distributed along a target answer. The entropy curve indicates the token-level uncertainty of the pre-edit model, while the lower bands show the optimization coverage of AnyEdit and DynEdit. AnyEdit follows a nearly uniform window allocation, whereas DynEdit adaptively increases coverage in high-entropy regions. This suggests that DynEdit performs more focused optimization on locally difficult and factually important positions, rather than uniformly editing the entire target sequence.

D Limitations

Although DynEdit achieves consistent improvements in long-sequence knowledge editing, several limitations remain. First, DynEdit relies on token-level entropy estimated from the pre-edit model as a proxy for editing difficulty and semantic importance. While effective in our experiments, entropy may not always perfectly reflect the true importance of a token, especially in highly structured or domain-specific generation tasks.

Second, the current framework is designed for sequential text generation and has been evaluated mainly on long-form factual editing benchmarks. Its effectiveness in more complex settings, such as multi-modal editing, multi-turn interactive editing, or reasoning-intensive generation, remains underexplored. n future work, we plan to incorporate graph pre-training techniques [52, 51] to represent entities, relations, and higher-order dependencies, thereby extending long-sequence knowledge editing to more complex multi-relation reasoning scenarios. Building on existing work on multimodal safety enhancement, [13], we will also investigate how long-sequence editing can be extended to safety-oriented applications.

In addition, DynEdit introduces additional scheduling and weighting operations during optimization. Although the overhead is moderate, the editing cost still increases compared with fixed-window methods. Further improving efficiency while maintaining editing quality is an important direction for future work.

More broadly, we hope this work can encourage future research on adaptive knowledge editing strategies that dynamically allocate editing capacity according to the internal structure and uncertainty of model knowledge.

E Broader Impact

This work studies long-sequence knowledge editing for large language models. Efficient and reliable knowledge editing can provide positive societal benefits by enabling models to correct outdated, inaccurate, or conflicting information without expensive retraining. Improving long-form editing may further enhance the factual consistency and controllability of deployed language models in applications such as education, question answering, and information services.

At the same time, knowledge editing techniques may also introduce potential risks if misused. For example, malicious users could intentionally modify model behavior to inject misleading information, manipulate generated content, or bypass safety constraints. As our method improves the effectiveness of long-sequence editing, it could potentially lower the cost of such manipulations.

Our work is intended for controlled and transparent model adaptation in research settings. We hope this research can contribute to a better understanding of reliable and interpretable knowledge updating mechanisms, while encouraging future work on safety evaluation, misuse detection, and robust editing constraints for large language models.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: Our abstract and introduction clearly state the arguments and contributions of the paper. See Section 1.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section Appendix D.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Section Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Section Appendix C.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide it in supplemental material.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Section Appendix C.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Section 5 and Appendix C.3. Error bars correspond to one standard deviation across runs.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Section Appendix C.2.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We strictly adhere to the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Section Appendix E.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Section Reference.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: We do not involve any risk datasets or models.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: This item is not involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This item is not involved.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: This item is not involved.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.