HyperEdit: Unlocking Instruction-based Text Editing in LLMs via Hypernetworks

Yiming Zeng[1], **Jinghan Cao**[2], **Zexin Li**[3], **Wanhao Yu**[4], **Zhankai Ye**[5], **Dawei Xiang**[1], **Ting Hua**[6], **Xin Liu**[5], **Shangqian Gao**[5†], **Tingting Yu**[1†]

[1]University of Connecticut, [2]San Francisco State University,
[3]University of California, Riverside, [4]University of North Carolina at Charlotte,
[5]Florida State University, [6]University of Notre Dame

Abstract

Instruction-based text editing is increasingly critical for real-world applications such as code editors (e.g., Cursor), but Large Language Models (LLMs) continue to struggle with this task. Unlike free-form generation, editing requires faithfully implementing user instructions while preserving unchanged content, as even minor unintended modifications can break functionality. Existing approaches treat editing as generic text generation, leading to two key failures: they struggle to faithfully align edits with diverse user intents, and they often over-edit unchanged regions. We propose HyperEdit to address both issues. First, we introduce hypernetwork-based dynamic adaptation that generates request-specific parameters, enabling the model to tailor its editing strategy to each instruction. Second, we develop difference-aware regularization that focuses supervision on modified spans, preventing over-editing while ensuring precise, minimal changes. HyperEdit achieves a **9%–30%** relative improvement in BLEU on modified regions over state-of-the-art baselines, despite utilizing only 3B parameters. To facilitate further research, we release HyperEdit at https://github.com/StuRinDQB/hyper_edit.

1 Introduction

Large Language Models (LLMs) have fundamentally transformed natural language processing (NLP) and its real-world applications (Qu et al., 2025; Wu et al., 2024b,a; Gao et al., 2025; Wang et al., 2024; Liu et al., 2024; Chen et al., 2024; Xiang et al., 2025), significantly accelerating advancements in text generation, reasoning, and summarization (Ravaut et al., 2024; Song et al., 2024; Li et al., 2024, 2025; Liu et al., 2025; Ji et al., 2025). Recently, the editing task has been brought to people's attention (Shu et al., 2024; Laban et al., 2024; Raheja et al., 2023; Zeng et al., 2025b). In NLP, the editing task refers to refining an existing draft by applying minimal, targeted modifications rather than fully regenerating the text (Iso et al., 2020; Stahlberg and Kumar, 2020). LLM-based editing has seen wide adoption in real-world applications such as Cursor for code editing, Grammarly for grammar correction, and Notion AI for collaborative document revision (Notion Labs Inc., 2022; Grammarly Inc., 2009; Anysphere, 2023). Unlike traditional generation tasks, editing requires faithfully implementing user instructions while preserving unchanged content. In code editing tools like Cursor, even minor unintended modifications can break functionality (Cassano et al., 2024). Similarly, in grammatical error correction, the model must amend only erroneous tokens while leaving correct elements untouched (Alhafni and Habash, 2025). These examples underscore the fundamental constraints of editing: *(1) Faithful adherence to user intent; (2) Preservation of unchanged context.*

However, as illustrated in Table 1, existing approaches fail to satisfy both constraints simultaneously. InstructGPT prioritizes instruction following but allows over-editing of unchanged regions (Ouyang et al., 2022). FineEdit benchmarks diverse editing tasks and incorporates edit differences, yet lacks explicit constraints during training to enforce minimal intervention (Zeng et al., 2025b). Similarly, code editing methods emphasize functional correctness while often overlooking local fidelity (Cassano et al., 2024). These limitations raise a key question: *can we design a method that simultaneously achieves intent alignment and local fidelity for reliable editing performance?*

To answer this question, we propose HyperEdit, a framework that integrates two key designs to address both constraints: (i) a hypernetwork-based dynamic modeling mechanism that generates request-specific parameters to adapt editing strategies to diverse user intents, and (ii) drawing inspiration from localization fidelity in computer vision (Zhao,

†Corresponding authors

Findings of the Association for Computational Linguistics: ACL 2026, pages 466–480
July 2-7, 2026

Baseline	Constraint 1	Constraint 2
FineEdit (Zeng et al., 2025b)	◐	◐
CODA (Cassano et al., 2024)	●	○
EDITCODER (Alhafni and Habash, 2025)	◐	●
InstructGPT (Ouyang et al., 2022)	●	○
HyperEdit	●	●
HyperEdit$_{Pro}$	●	●

Table 1: Existing baselines and their supported constraints on the editing task. ● represents support, ◐ represents partial support, and ○ represents no support.

2024), we design a difference-aware regularization that focuses supervision on modified spans, ensuring precise modifications with minimal intervention. By enabling adaptive intent alignment and targeted content preservation, HyperEdit achieves state-of-the-art editing performance. Our contributions can be summarized as follows:

- We propose a hypernetwork-based dynamic adaptation mechanism that generates request-specific low-rank parameters, enabling the model to tailor its editing behavior to diverse instructions without retraining. To our knowledge, this is the first work to introduce dynamic parameter generation for instruction-based text editing.
- We develop a difference-aware regularization loss that applies targeted supervision to edited spans, guiding the model to focus on necessary modifications while keeping unchanged content.
- Extensive experiments demonstrate that HyperEdit achieves substantial improvements over existing methods: approximately 9%–30% relative improvement over the state-of-the-art editing model FineEdit-Pro, and 50% relative improvement over the 14B-parameter Qwen-2.5-14B-Instruct, while using only 3B parameters.

2 Related Work

2.1 LLM Editing Task

Instruction tuning has enabled LLMs to perform editing in more diverse and user-driven scenarios. Models such as InstructGPT (Ouyang et al., 2022), RewriteLM (Shu et al., 2024), and DocMEdit (Zeng et al., 2025a) exhibit strong abilities in following user commands but sometimes suffer from over-editing or altering irrelevant content.

Although instruction-tuned LLMs show promise in editing, existing approaches are often fragmented across tasks and domains. Recent studies therefore propose unified methods to address this limitation. FineEdit introduces InstrEditBench, a structured benchmark spanning multiple domains to assess instruction-based edits (Zeng et al., 2025b). CoEdIT develops an instruction-tuned editing system that generalizes across diverse tasks (Raheja et al., 2023), and its multilingual extension mEdIT extends this paradigm to multiple languages (Raheja et al., 2024). XATU focuses on fine-grained and explainable editing benchmarks by explicitly annotating editing rationales (Zhang et al., 2024).

On the dataset side, Beemo captures real-world human post-edits over machine-generated text, enabling analysis of how expert edits differ from purely automated ones (Artemova et al., 2025). Research has also examined LLMs for code editing: Cassano et al. (Cassano et al., 2024) assessed LLMs on code editing instructions, and Fan et al. (Fan et al., 2025) studied realistic scenarios like commit message generation and code reviews.

Existing studies mainly focus on building benchmarks or designing systems for broad task coverage, but they often lack mechanisms to ensure both intent alignment and local fidelity. Our method belongs to the instruction-based editing category and introduces hypernetwork-driven dynamic adapters together with a difference-aware loss to achieve precise and minimally invasive edits.

2.2 Hypernetwork Applications

Hypernetworks are a class of neural network methods that improve generalization and adaptability by dynamically generating target model parameters. Early work has demonstrated the advantages of hypernetworks in few-shot learning and continual learning (Ha et al., 2017; von Oswald et al., 2020). By sharing a transferable parameter generator, they reduce optimization time and alleviate overfitting. Recent research has further expanded the application of hypernetworks in implicit neural representations (INRs), including 3D object modeling (Sen et al., 2024), video and image stylization (Maiya et al., 2024), and video decomposition (Pilligua et al., 2025).

While hypernetworks have proven effective in vision tasks, they remain underexplored in NLP. Editing is inherently instruction-sensitive: the model must adapt its response to each request while preserving unrelated content. Static adapters and traditional fine-tuning fail to capture this variability, as they rely on the same parameters across all inputs. Hypernetworks overcome this limita-

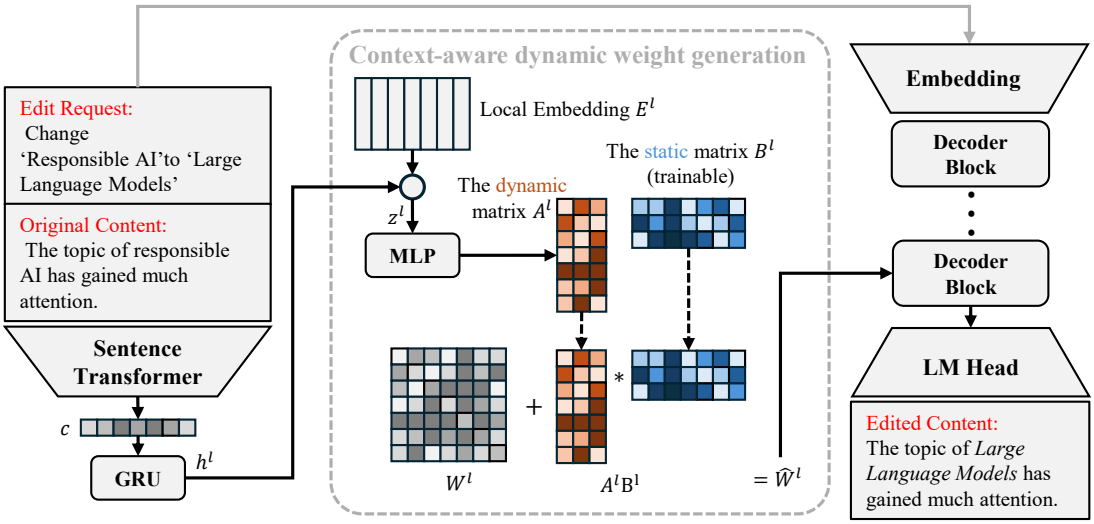


Figure 1: The process of context-aware dynamic weight generation. The hypernetwork will generate the dynamic weight matrix $\mathbf{A}^l$ given the edit request and the original text. The original model weights $\mathbf{W}^l$ are then modified to produce $\hat{\mathbf{W}}^l = \mathbf{W}^l + \mathbf{A}^l\mathbf{B}^l$ to capture context-specific requirements from the prompt.

tion by generating context-dependent weights, enabling fine-grained and flexible adaptation crucial for instruction-based editing.

3 Method

3.1 Problem Formulation

We define the instruction-based editing task as transforming an original text T_{orig} into an edited text T_{edit} under the guidance of an editing instruction I_{edit}. The objective is to ensure that T_{edit} faithfully incorporates the modifications specified by I_{edit}, while preserving the content and structure of T_{orig} wherever changes are not required. Formally, the task can be expressed as a conditional mapping:

$$T_{\text{edit}} = f_{\text{LLM}}\big(T_{\text{orig}}, I_{\text{edit}}; \theta\big), \tag{1}$$

where θ denotes the learnable parameters, and f_{LLM} is instantiated by a language model that generates the edited text given the input pair $(T_{\text{orig}}, I_{\text{edit}})$.

Instruction-based editing requires the model to perform minimal yet precise modifications: unchanged portions of T_{orig} should be preserved as much as possible, while only the regions targeted by I_{edit} are modified. This makes the task more constrained than general text generation, as the model must balance two competing objectives: (i) faithfully implementing the instruction I_{edit}, and (ii) avoiding unnecessary alterations to T_{orig}.

In practice, editing requests vary widely in type and complexity, requiring the model to dynamically adapt its behavior to different instructions. To this end, we propose a *hypernetwork-based dynamic adaptation* scheme that generates request-specific parameters on the fly, enabling tailored editing strategies for each input. However, standard likelihood objectives may still under-emphasize the regions where edits actually occur, since unchanged tokens dominate the sequence. To address this, we further introduce a *difference-aware regularization* that places targeted supervision on modified spans, ensuring faithful and minimal edits.

3.2 Training Process

Our training process incorporates two key mechanisms: a hypernetwork that generates request-specific low-rank parameters for dynamic adaptation, and a difference-aware regularization loss that focuses supervision on modified spans.

3.2.1 HyperNetwork for Dynamic Low-Rank Adaptation

To better capture diverse editing intents in a flexible manner, we adopt a hypernetwork-based dynamic parameter generation scheme illustrated in Figure 1. Specifically, following the LoRA paradigm (Hu et al., 2022), the hypernetwork produces input-dependent, low-rank adaptation matrices for selected layers of the frozen LLM. This design enables the model to disentangle general editing patterns from instruction-specific contexts, and to construct specialized adapters tailored to each editing request.

The process begins by encoding the edit instruction I_{edit} and the original text chunk T_{orig} into

a single context embedding $\mathbf{c} \in \mathbb{R}^{d_{\text{emb}}}$ using a pre-trained, frozen sentence encoder (Reimers and Gurevych, 2019), which provides a rich semantic representation of the editing task. Formally, we concatenate the two inputs and feed them into the encoder:

$$x = [I_{\text{edit}}; T_{\text{orig}}], \quad \mathbf{c} = \text{ST}(x) \in \mathbb{R}^{d_{\text{emb}}}, \tag{2}$$

where $\text{ST}(\cdot)$ denotes the pre-trained sentence transformer encoder that maps the concatenated input x into the embedding space.

The hypernetwork consists of three components: a **GRU** that encodes the context vector $\mathbf{c}$ obtained from the sentence transformer for each weight matrix, a **local embedding** that stores d^l_{in} vectors of rank r, and an **MLP** that generates the dynamic low-rank weights $\mathbf{A}^l$ based on the local embeddings and the GRU outputs.

More specifically, the formulation of GRU is as follows:

$$\mathbf{h}^l = \text{GRU}\left(\mathbf{c}, \mathbf{h}^{l-1}\right), \tag{3}$$

where $\mathbf{h}^l$ is the encoded $\mathbf{c}$ for the l-th weight matrix. The local embedding $E^l \in \mathbb{R}^{d^l_{\text{in}} \times r}$ stores learnable codes for each dimension in d^l_{in}. E^l is a static local embedding, and as a result, we need to fuse the local embedding with the request-based encoding $\mathbf{h}^l$ from $\mathbf{c}$. To achieve this, we concatenate E^l and $\mathbf{h}^l$ together and apply layer normalization (LN) (Ba et al., 2016) and GeLU (Hendrycks and Gimpel, 2016) before feeding them into the final MLP layer:

$$\mathbf{A}^l = \text{MLP}(\text{GeLU}(\text{LN}(\mathbf{z}^l))) \tag{4}$$

where $\mathbf{z}^l = [E^l; \text{expand}(\mathbf{h}^l)]$, and $\mathbf{h}^l \in \mathbb{R}^{d_{\text{h}}}$ is first expanded to $d^l_{\text{in}} \times d_{\text{h}}$, $\mathbf{z}^l \in \mathbb{R}^{d^l_{\text{in}} \times (d_{\text{h}}+r)}$, and $W^l_{\text{MLP}} \in \mathbb{R}^{(r+d_{\text{h}}) \times r}$. The final size of $\mathbf{A}^l$ is the same as the local embedding E^l: $d^l_{\text{in}} \times r$. The function of $\text{MLP}(\cdot)$ is to fuse information from the static local embeddings and the adaptive input $\mathbf{h}^l$. Consequently, the generated $\mathbf{A}^l$ becomes conditionally dependent on the editing request and the corresponding text. During this process, we generate only $\mathbf{A}^l$ while keeping $\mathbf{B}^l$ input-independent, since $\mathbf{B}^l$ is initialized as an all-zero matrix in LoRA.

The additional learnable parameters introduced by our design are quite small. The parameters of the GRU can be neglected due to its small hidden size. Compared to the original LoRA, the extra trainable parameters are $r \times (d_{\text{h}} + r)$, where d_{h} is typically set to a small value such as 32 or 64.

At train time, given an input $\mathbf{X}^l \in \mathbb{R}^{T \times d^l_{\text{in}}}$, the adapted output of the l-th weight matrix is computed as:

$$\mathcal{F}^l = \mathbf{X}^l W^l + \Delta\mathcal{F}^l, \quad \Delta\mathcal{F}^l = \frac{\alpha}{r}\,\mathbf{X}^l \mathbf{A}^l \mathbf{B}^l, \tag{5}$$

where α is a constant scaling factor. This formulation keeps the original weight matrix W^l frozen while dynamically constructing a context-dependent $\Delta\mathcal{F}^l$ for each editing request.

During training, both the sentence transformer and the LLM are kept frozen. The learnable parameters of our method are denoted as $\Theta = [\theta_{\text{GRU}}, \theta_{\text{MLP}}, E, \mathbf{B}]$, where θ_{GRU} represents the parameters of the GRU, θ_{MLP} corresponds to the parameters of all MLP layers and LNs, E includes all local embeddings E^l, and $\mathbf{B}$ includes the parameters of all $\mathbf{B}^l$ matrices.

3.2.2 Difference-Aware Regularization

In addition to standard supervised fine-tuning, we introduce a *difference-aware loss* that emphasizes supervision on tokens deviating from the original text. Given the target sequence T_{edit} and the reference context T_{orig}, we compute token-level alignment via the longest common subsequence (LCS), which identifies positions where tokens differ between the two sequences. This yields a binary mask $m_{1:T}$ where $m_t = 1$ indicates that position t contains an insertion or replacement, and $m_t = 0$ indicates unchanged tokens. The loss is then applied exclusively to the modified positions. Formally, for the token sequence $y_{1:T}$ of T_{edit} and mask $m_{1:T}$, the loss is defined as:

$$\mathcal{L}_{\text{diff}} = -\frac{1}{\sum_t m_t} \sum_{t=1}^{T} m_t \, \log p\big(y_t \mid y_{<t}, T_{\text{orig}}, I_{\text{edit}}\big). \tag{6}$$

This regularization loss guides the model to focus learning on modified regions while reducing penalties on unchanged text. The overall training objective combines $\mathcal{L}_{\text{diff}}$ with the supervised fine-tuning loss (Ouyang et al., 2022):

$$\min_{\Theta} \; \mathcal{L}_{\text{sft}} + \lambda\, \mathcal{L}_{\text{diff}}, \tag{7}$$

where Θ is introduced in Section 3.2.1, $\mathcal{L}_{\text{sft}}$ is the language modeling loss applied to the edited response, and λ balances the two objectives. Empirically, we find $\lambda = 1$ performs robustly across diverse editing domains.

3.3 Inference Process

At inference time, the hypernetwork generates request-specific adapters for each editing task. Given a new editing request with instruction I_{edit} and original text T_{orig}, we first encode them into a context vector $\mathbf{c}$ using the frozen sentence transformer (as in training). The hypernetwork then processes $\mathbf{c}$ to generate dynamic adapter matrices $\mathbf{A}^l$ for each target layer l. These are combined with the static matrices $\mathbf{B}^l$ to form the low-rank updates:

$$\Delta W^l = \tfrac{\alpha}{r}\, \mathbf{A}^l \mathbf{B}^l. \tag{8}$$

During the LLM forward pass, each layer l applies the adapted weights. For an input $\mathbf{X}^l$ and frozen base weight W^l, the output is computed as:

$$\mathcal{F}^l = \mathbf{X}^l (W^l + \Delta W^l). \tag{9}$$

Note that, the base LLM parameters W^l remain frozen throughout; only the dynamically generated ΔW^l varies across different editing requests, enabling request-specific adaptation without modifying the base model. Since the additional computation involves only the lightweight hypernetwork forward pass and low-rank matrix multiplications, the method remains efficient and scalable across diverse editing scenarios.

3.4 Evaluation Metrics

We evaluate model performance with *Diff-BLEU* and *Diff-ROUGE-L*, two span-focused variants of standard text similarity metrics.

While BLEU (Papineni et al., 2002) and ROUGE-L (Lin, 2004) compute similarity over the entire output, our metrics restrict computation to the *edited spans* identified by LCS-based alignment between the original and target text. This design better captures editing quality: since unchanged tokens typically dominate the output, standard metrics can yield high scores even when the model fails to make correct edits. By focusing only on modified regions, our metrics directly measure whether the model implements the intended modifications correctly.

Diff-BLEU. To evaluate the quality of the specific modifications, we calculate BLEU scores exclusively on the edited spans. Let $Y_{\text{edit}}^{\text{span}} = (y_1, \ldots, y_M)$ denote the sequence of reference tokens extracted from the target side of the edits (i.e., capturing the content of *insertions* and the *new* components of *replacements*). Correspondingly, let $\hat{Y}_{\text{edit}}^{\text{span}} = (\hat{y}_1, \ldots, \hat{y}_N)$ denote the sequence of tokens generated by the model within the predicted edit spans. *Diff-BLEU* is defined as:

$$\textit{Diff-BLEU} = \textit{BLEU}\big(Y_{\text{edit}}^{\text{span}}, \hat{Y}_{\text{edit}}^{\text{span}}\big). \tag{10}$$

This metric assesses whether the model accurately generates the intended modifications, focusing solely on the changed content rather than the unchanged context.

This measures the n-gram precision of modifications, evaluating whether the edited tokens are accurate and well-formed.

Diff-ROUGE-L. Similarly, *Diff-ROUGE-L* computes the F1-score based on the Longest Common Subsequence (LCS). Let L denote the length of the LCS between the reference edited spans $Y_{\text{edit}}^{\text{span}}$ and the model's predicted spans $\hat{Y}_{\text{edit}}^{\text{span}}$. We define recall $R = L/|Y_{\text{edit}}^{\text{span}}|$ and precision $P = L/|\hat{Y}_{\text{edit}}^{\text{span}}|$, where $|\cdot|$ represents the number of tokens. The metric is calculated as:

$$\textit{Diff-ROUGE-L} = \frac{2 \cdot P \cdot R}{P + R} \tag{11}$$

This measures the coverage of edits, capturing whether the model produces the essential changes even under minor lexical variations. We discuss the details of edit-span extraction in Appendix D.

4 Evaluation

4.1 Experimental Setup

Datasets We conduct experiments on InstrEditBench (Zeng et al., 2025b), which integrates instruction-based editing tasks from four domains: LaTeX, programming code, Wikipedia text, and domain-specific languages (DSL). Following FineEdit, we use a 90/10 train-test split. Detailed hyperparameters are provided in the appendix.

Models and Baselines We implement HyperEdit on two base models: LLaMA-3.2-3B (Dubey et al., 2024) and Qwen-2.5-3B-Instruct (Yang et al., 2025b), both fine-tuned on InstrEditBench with our proposed hypernetwork and difference-aware regularization.

We compare against two categories of baselines: (1) *Editing-specialized models*: FineEdit-Pro and FineEdit-X (Zeng et al., 2025b), which are specifically designed for instruction-based editing; (2) *General-purpose LLMs*: LLaMA-3.2-1B, LLaMA-3.2-3B, LLaMA-3.1-8B-Instruct (Dubey

et al., 2024), Qwen-2.5-3B-Instruct, Qwen-2.5-14B-Instruct (Yang et al., 2025b), and Qwen-3-8B (Yang et al., 2025a), evaluated in zero-shot or instruction-tuned settings.

4.2 Overall Effectiveness

We evaluate HyperEdit under both single-turn and multi-turn settings. The single-turn version processes one instruction per instance, while the multi-turn version contains several editing requests applied sequentially. Each edit builds upon the model's previous output, simulating a realistic iterative editing process.

Single-turn Experiments Table 2 reports the overall results on InstrEdit-Bench across Code, LaTeX, DSL, and Wikipedia. General-purpose models exhibit limited editing capability, with most scoring below 0.11 on Diff-BLEU. Among them, LLaMA-3.2-3B performs best (0.1323 Diff-BLEU, 0.2261 Diff-ROUGE-L), yet even larger instruction-tuned models like Qwen-2.5-14B-Instruct (0.1031 / 0.1689) struggle with fine-grained editing. This suggests that standard instruction-following training is insufficient for precise, minimal-intervention edits.

Editing-specialized models show clear improvements. The FineEdit family substantially outperforms general-purpose LLMs, with FineEdit-Pro achieving 0.1437 Diff-BLEU and 0.2345 Diff-ROUGE-L. This confirms that task-specific training enhances editing quality.

HyperEdit achieves the strongest performance through dynamic adaptation. Our best variant (Qwen-2.5-3B) obtains 0.1565 Diff-BLEU and 0.2537 Diff-ROUGE-L—approximately 9% relative improvement over FineEdit-Pro. Notably, HyperEdit accomplishes this with only 3B parameters while outperforming models 4-5× larger. The LLaMA-based variant (0.1449 / 0.2360) similarly surpasses all baselines, demonstrating the approach's generality. These gains stem from the hypernetwork's ability to generate request-specific parameters that adapt editing strategies to diverse instructions while maintaining local fidelity.

Domain-specific results reveal where dynamic adaptation provides the greatest benefit. On structured domains requiring precise modifications—LaTeX (0.1706 / 0.2803) and DSL (0.0693 / 0.1114)—HyperEdit achieves the strongest gains, benefiting from its ability to tailor editing patterns to domain-specific syntax. On Wikipedia, a more free-form domain, HyperEdit still substantially outperforms FineEdit-Pro (0.3527 vs. 0.3617 on BLEU; 0.4792 vs. 0.4744 on ROUGE-L), indicating that dynamic modeling helps even when edits involve stylistic or semantic changes rather than structural ones.

In addition, we further introduce HyperEdit-Pro(Qwen-2.5-3B), which extends the sliding window to 4096 tokens and removes truncation constraints to preserve the entire input sequence. This configuration achieves the highest performance of 0.1714 Diff-BLEU and 0.2432 Diff-ROUGE-L. These findings indicate that input integrity is critical for instruction-based editing. Preserving the full context enables the model to accurately capture long-range dependencies and maintain structural coherence, which are often compromised by standard truncation strategies.

Overall, HyperEdit demonstrates that combining hypernetwork-based dynamic adaptation with difference-aware regularization effectively addresses both fundamental editing constraints: faithful intent alignment and preservation of unchanged content. The consistent improvements across diverse domains and editing scenarios validate the effectiveness of our approach.

Multi-turn Experiments Multi-turn editing presents additional challenges beyond single-turn scenarios: models must keep consistency across sequential edits while avoiding error accumulation as modifications build upon previous outputs. Table 4 shows results in this challenging setting.

General LLMs struggle with iterative editing. LLaMA-3.2-3B obtains 0.1770 Diff-BLEU and 0.2982 Diff-ROUGE-L, while larger models like Qwen-3-8B (0.1711 / 0.2649) perform comparably or worse. These results indicate that standard instruction-following training provides insufficient mechanisms for tracking cumulative changes and preserving local fidelity across editing rounds.

Editing-specialized models show improved robustness. FineEdit-Pro achieves the best performance among existing baselines with 0.2093 Diff-BLEU and 0.3287 Diff-ROUGE-L, demonstrating that task-specific training helps models track revision dependencies across multiple editing rounds.

HyperEdit substantially outperforms all baselines in multi-turn settings. Our model achieves 0.2479 Diff-BLEU and 0.3620 Diff-ROUGE-L overall—an 18% relative improvement over FineEdit-Pro. This larger gain compared to single-

Model/Type	Diff-BLEU					Diff-ROUGE-L				
	Code	LaTeX	DSL	Wiki	Overall	Code	LaTeX	DSL	Wiki	Overall
Llama-3.2-1B	0.0366	0.0886	0.0272	0.2573	0.1139	0.1631	0.2424	0.0800	0.3769	0.2156
Llama-3.2-3B	0.0568	0.1455	0.0523	0.2746	0.1323	0.1677	0.2492	0.1045	0.3830	0.2261
Llama-3.1-8B-Instruct	0.0335	0.1244	0.0214	0.2523	0.1079	0.1123	0.2194	0.0519	0.3504	0.1835
LLama-3.1-13B-Instruct	0.0286	0.0729	0.0119	0.2458	0.0898	0.0983	0.1379	0.0332	0.3396	0.1523
Qwen-2.5-3B-Instruct	0.0280	0.0929	0.0187	0.2791	0.1047	0.0840	0.1695	0.0472	0.3897	0.1726
Qwen-3-8B	0.0320	0.0827	0.0171	0.2923	0.1060	0.0991	0.1463	0.0417	0.3992	0.1716
Qwen-2.5-14B-Instruct	0.0281	0.0928	0.0194	0.2722	0.1031	0.0847	0.1623	0.0471	0.3814	0.1689
FineEdit-X	0.0483	0.1176	0.0389	0.3499	0.1387	0.1600	0.2101	0.0867	0.4536	0.2276
FineEdit-Pro	0.0497	0.1278	0.0356	**0.3617**	0.1437	0.1668	0.2295	0.0674	0.4744	0.2345
HyperEdit (llama3.2-3B)	0.0515	0.1544	**0.0693**	0.3043	0.1449	0.1569	0.2632	0.1114	0.4123	0.2360
HyperEdit (qwen-2.5-3B)	0.0492	0.1706	0.0535	0.3527	**0.1565**	0.1539	0.2803	0.1014	**0.4792**	0.2537
HyperEdit-Pro (qwen-2.5-3B)	**0.1714**	**0.2432**	0.0472	0.2860	**0.1870**	**0.2880**	**0.3653**	**0.1137**	0.4245	**0.2979**

Table 2: Comparison of models on Diff-BLEU and Diff-ROUGE-L across domains. Models are grouped by family and ordered by size.

Model	Code		LaTeX		DSL		Wiki		Overall	
	BLEU	ROUGE	BLEU	ROUGE	BLEU	ROUGE	BLEU	ROUGE	BLEU	ROUGE
HyperEdit [without diff and hypernet]	0.0497	0.1668	0.1278	0.2295	0.0356	0.0674	0.3617	0.4744	0.1437	0.2345
HyperEdit [without diff]	**0.0504**	0.1628	**0.1339**	**0.2346**	**0.0521**	**0.0954**	0.3423	0.4551	**0.1447**	**0.2370**
HyperEdit	0.0492	0.1539	**0.1706**	**0.2803**	**0.0535**	**0.1014**	0.3527	**0.4792**	**0.1565**	**0.2537**

Table 3: Ablation study results on Diff-BLEU and Diff-ROUGE across Code, LaTeX, DSL, and Wiki datasets. BLEU stands for Diff-BLEU, while ROUGE stands for Diff-ROUGE. Values in **bold** exceed the corresponding base scores.

Model/Type	Diff-BLEU					Diff-ROUGE-L				
	Code	LaTeX	DSL	Wiki	Overall	Code	LaTeX	DSL	Wiki	Overall
Llama-3.2-1B	0.08	0.1991	0.0836	0.3377	0.1735	0.2093	0.3252	0.1613	0.4711	0.2917
Llama-3.2-3B	0.0772	0.2187	**0.0898**	0.3222	0.1770	0.2124	0.3493	**0.1731**	0.4581	0.2982
Llama-3.1-8B-Instruct	0.0506	0.1869	0.0267	0.2989	0.1408	0.1496	0.2965	0.0602	0.4219	0.2321
Qwen-2.5-3B-Instruct	0.0631	0.1934	0.0374	0.3822	0.1690	0.1677	0.3056	0.084	0.5124	0.2674
Qwen-3-8B	0.0524	0.2022	0.0372	0.3925	0.1711	0.1444	0.3117	0.0753	0.528	0.2649
FineEdit-X	0.0843	0.2077	0.0515	0.3379	0.1704	0.2077	0.3106	0.1006	0.4704	0.2723
FineEdit-Pro	**0.0914**	0.2500	0.0511	0.4446	0.2093	**0.2554**	0.3729	0.1008	0.5858	0.3287
HyperEdit (qwen-2.5-3B)	0.0809	**0.3894**	0.0547	**0.4665**	**0.2479**	0.2132	**0.5182**	0.1084	**0.6083**	**0.3620**

Table 4: Comparison of models on Diff-BLEU and Diff-ROUGE-L across domains in the Multi-turn Editing Task. Models are grouped by family and ordered by size.

turn results (9%) suggests that dynamic adaptation is particularly valuable when edits accumulate: the hypernetwork generates fresh, request-specific parameters for each round, preventing error propagation while maintaining awareness of both the current instruction and the evolving document state.

The advantages are especially pronounced on structured and long-form domains. On LaTeX, HyperEdit achieves 0.3894 / 0.5182, far exceeding FineEdit-Pro (0.2500 / 0.3729), indicating strong capability for handling complex structural revisions across multiple steps. On Wikipedia, HyperEdit reaches 0.4665 / 0.6083 compared to FineEdit-Pro's 0.4446 / 0.5858, demonstrating robust performance on lengthy, coherent text that requires maintaining consistency across edits. Even on Code and DSL, where precision is critical, HyperEdit maintains competitive or superior performance.

These results confirm that dynamic parameter generation enables more effective multi-turn editing by adapting to each instruction while preserving the integrity of previous modifications—a key requirement for practical iterative editing workflows.

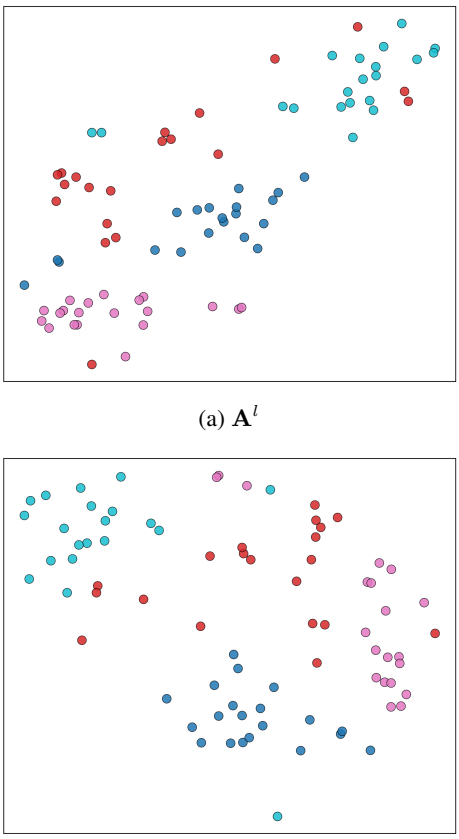
4.3 Qualitative Study

To verify that HyperEdit's hypernetwork generates meaningfully different parameters for different domains and editing scenarios, we analyze the generated dynamic weights. We randomly sampled 20 inference cases from each of four domains (*Code*, *LaTeX*, *DSL*, and *Wiki*), extracted the adapter matrices $\mathbf{A}^l$ and $\mathbf{B}^l$, computed their layer-wise Frobenius norms, and applied t-SNE for visualization.

Figure 2a shows the $\mathbf{A}^l$ visualization, which reveals distinct, well-separated clusters for each domain. This confirms that $\mathbf{A}^l$, the hypernetwork's output, captures domain-specific editing strategies. *Code* and *DSL* form tight clusters, consistent with their reliance on rigid, syntax-driven transformations. *LaTeX* and *Wiki* exhibit more dispersed patterns, reflecting the greater linguistic and structural diversity in these domains.

In contrast, Figure 2b shows the $\mathbf{A}^l\mathbf{B}^l$ representation with considerable overlap, especially between *LaTeX* and *Wiki*. This suggests that $\mathbf{B}^l$, the static low-rank projection, serves as a shared basis that enables cross-domain generalization while $\mathbf{A}^l$ provides domain-specific adaptation.

To quantify domain separation, we computed pairwise cosine similarities between the layer-wise Frobenius norms of $\mathbf{A}^l$ across all 80 cases. The resulting dynamic weight similarity matrix (Figure 3) exhibits clear block-diagonal structure with strong intra-domain similarity (dark blocks) and weaker inter-domain similarity (lighter off-diagonal regions). The average intra-domain similarity is 1.49× higher than inter-domain similarity, confirming that the hypernetwork consistently generates domain-specific weight patterns tailored to each editing context.



(a) $\mathbf{A}^l$

(b) $\mathbf{A}^l\mathbf{B}^l$

Figure 2: t-SNE visualization of dynamic weight distributions across domains. Different editing domains are represented by the following colors: **Code**, **LaTeX**, **SQL**, **Wiki**.

4.4 Ablation Study

To evaluate each component's contribution, we conduct an ablation study with three variants: (1) baseline with standard LoRA (no hypernetwork, no difference-aware loss), (2) hypernetwork only (dynamic adaptation without difference-aware loss), and (3) full HyperEdit (both components). All variants use identical settings for fair comparison.

Table 3 shows that the hypernetwork alone provides substantial improvements. Comparing variants (1) and (2), dynamic adaptation yields gains across most domains, particularly on structured tasks: LaTeX improves from 0.1278 to 0.1339

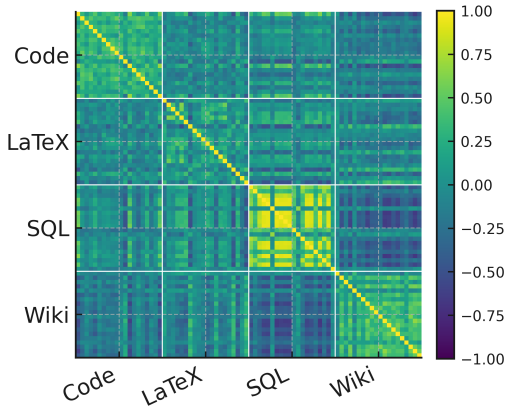


Figure 3: Dynamic Weight Similarity Matrix across editing domains. Pairwise cosine similarity of $\mathbf{A}^l$ Frobenius norms across 80 cases (20 per domain). Dark diagonal blocks show high intra-domain similarity; lighter off-diagonal regions indicate lower inter-domain similarity.

BLEU, and DSL from 0.0356 to 0.0521. On Wiki, the gains are even larger (0.3617 to 0.3423 BLEU), demonstrating that request-specific parameter generation helps the model adapt to diverse editing intents even without explicit supervision on modified regions.

Adding difference-aware regularization (variant 3) further improves performance. The full model achieves the best overall scores. The gains are most pronounced on LaTeX (0.1706 BLEU) and Wiki (0.3527 BLEU), where precise localization of edits is critical. On Code, the hypernetwork-only variant performs slightly better (0.0504 vs. 0.0492), likely because code editing involves more predictable, syntax-driven changes where aggressive localization may overly constrain the model. Overall, the full model achieves the best balance, confirming that both components contribute complementary benefits: dynamic adaptation for intent alignment and difference-aware regularization for local fidelity.

5 Conclusion

This paper proposes HyperEdit, a dual-modeling framework for instruction-based editing tasks. It adaptively aligns editing strategies with user intent through dynamic parameter generation driven by a hypernetwork. Combined with a difference-aware regularization, it ensures that modifications are focused on essential areas while preserving the original text as much as possible. Experimental results demonstrate that HyperEdit significantly outperforms general-purpose models with larger parameters and existing specialized editing methods, demonstrating its advantages in accuracy and local fidelity.

6 Acknowledgements

We would like to sincerely thank the reviewers, Area Chairs, and Senior Area Chairs for their insightful feedback, which has significantly improved this work.

Limitations

Multi-turn Training. Our current framework does not incorporate explicit multi-round training. Extending HyperEdit to support iterative optimization across multiple editing turns remains an important direction for future work.

Computational Costs of Multi-Turn Evaluation. Multi-turn evaluation requires repeated interactions, which significantly increase inference time and computational cost. Therefore, we did not include results from larger-scale models, such as 14B parameters or above, in this study.

References

Bashar Alhafni and Nizar Habash. 2025. Enhancing text editing for grammatical error correction: Arabic as a case study. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17892–17914, Vienna, Austria. Association for Computational Linguistics.

Anysphere. 2023. Cursor: Ai-powered code editor. Software.

Ekaterina Artemova, Jason S Lucas, Saranya Venkatraman, Jooyoung Lee, Sergei Tilga, Adaku Uchendu, and Vladislav Mikhailov. 2025. Beemo: Benchmark of expert-edited machine-generated outputs. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6992–7018, Albuquerque, New Mexico. Association for Computational Linguistics.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. Layer normalization. *arXiv preprint arXiv:1607.06450*.

Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, and Arjun Guha. 2024. Can it edit? evaluating the ability of large language models to follow code editing instructions. *Preprint*, arXiv:2312.12450.

Adam Casson. 2023. Transformer flops.

Yiming Chen, Xianghu Yue, Xiaoxue Gao, Chen Zhang, Luis Fernando D'Haro, Robby T. Tan, and Haizhou Li. 2024. Beyond single-audio: Advancing multi-audio processing in audio large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10917–10930, Miami, Florida, USA. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv e-prints*, pages arXiv–2407.

Lishui Fan, Jiakun Liu, Zhongxin Liu, David Lo, Xin Xia, and Shanping Li. 2025. Exploring the capabilities of llms for code-change-related tasks. *ACM Transactions on Software Engineering and Methodology*, 34(6):1–36.

Shangqian Gao, Ting Hua, Reza Shirkavand, Chi-Heng Lin, Zhen Tang, Zhengao Li, Longge Yuan, Fangyi Li, Zeyu Zhang, Alireza Ganjdanesh, et al. 2025. Tomoe: Converting dense large language models to mixture-of-experts through dynamic structural pruning. *arXiv preprint arXiv:2501.15316*.

Grammarly Inc. 2009. Grammarly: Writing assistant. Software.

David Ha, Andrew M. Dai, and Quoc V. Le. 2017. Hypernetworks. In *International Conference on Learning Representations (ICLR)*.

Dan Hendrycks and Kevin Gimpel. 2016. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2022. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3.

Hayate Iso, Chao Qiao, and Hang Li. 2020. Fact-based Text Editing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 171–182, Online. Association for Computational Linguistics.

Yuelyu Ji, Zhuochun Li, Rui Meng, and Daqing He. 2025. Reason-to-rank: Distilling direct and comparative reasoning from large language models for document reranking. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '25, page 2320–2329, New York, NY, USA. Association for Computing Machinery.

Philippe Laban, Jesse Vig, Marti Hearst, Caiming Xiong, and Chien-Sheng Wu. 2024. Beyond the chat: Executable and verifiable text-editing with llms. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, New York, NY, USA. Association for Computing Machinery.

Dongfang Li, Zetian Sun, Baotian Hu, Zhenyu Liu, Xinshuo Hu, Xuebo Liu, and Min Zhang. 2024. Improving attributed text generation of large language models via preference learning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 5079–5101, Bangkok, Thailand. Association for Computational Linguistics.

Zhuochun Li, Yuelyu Ji, Rui Meng, and Daqing He. 2025. Learning from committee: Reasoning distillation from a mixture of teachers with peer-review. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 4190–4205.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text Summarization Branches Out: Proceedings of the ACL-04 Workshop*, pages 74–81.

Junnan Liu, Hongwei Liu, Linchen Xiao, Ziyi Wang, Kuikun Liu, Songyang Gao, Wenwei Zhang, Songyang Zhang, and Kai Chen. 2025. Are your LLMs capable of stable reasoning? In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 17594–17632, Vienna, Austria. Association for Computational Linguistics.

Xiang Liu, Xueling Liu, Jing Diao, Mengyao Zheng, Jihe Li, Yongyi Xie, Kang Lai, Xiao Geng, Yijun Song, and Linshan Jiang. 2024. Novel truncated-rank graph-structured and tree-guided sparse linear mixed models for variable selection on genome-wide association studies. In *2024 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 5868–5875.

Shishira R. Maiya, Anubhav Gupta, Matthew Gwilliam, Max Ehrlich, and Abhinav Shrivastava. 2024. Latent-inr: A flexible framework for implicit representations of videos with discriminative semantics. In *European Conference on Computer Vision (ECCV)*.

Notion Labs Inc. 2022. Notion ai. Software.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22.

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

Maria Pilligua, Danna Xue, and Javier Vázquez-Corral. 2025. Hypernvd: Accelerating neural video decomposition via hypernetworks. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Leigang Qu, Haochuan Li, Wenjie Wang, Xiang Liu, Juncheng Li, Liqiang Nie, and Tat-Seng Chua. 2025. Silmm: Self-improving large multimodal models for compositional text-to-image generation. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 18497–18508.

Vipul Raheja, Dimitris Alikaniotis, Vivek Kulkarni, Bashar Alhafni, and Dhruv Kumar. 2024. mEdIT: Multilingual text editing via instruction tuning. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 979–1001, Mexico City, Mexico. Association for Computational Linguistics.

Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. 2023. CoEdIT: Text editing by task-specific instruction tuning. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5274–5291, Singapore. Association for Computational Linguistics.

Mathieu Ravaut, Aixin Sun, Nancy Chen, and Shafiq Joty. 2024. On context utilization in summarization with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2764–2781, Bangkok, Thailand. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. 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.

Bipasha Sen, Gaurav Singh, Aditya Agarwal, Rohith Agaram, Madhava Krishna, and Srinath Sridhar. 2024. Hypnerf: Learning improved nerf priors using a hypernetwork. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Lei Shu, Lei Luo, Jayant Hoskere, Yichong Zhu, Yicheng Liu, Shilin Tong, Jinxin Chen, and Lingyu Meng. 2024. Rewritelm: An instruction-tuned large language model for text rewriting. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 18970–18980.

Hwanjun Song, Hang Su, Igor Shalyminov, Jason Cai, and Saab Mansour. 2024. FineSurE: Fine-grained summarization evaluation using LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 906–922, Bangkok, Thailand. Association for Computational Linguistics.

Felix Stahlberg and Shankar Kumar. 2020. Seq2Edits: Sequence transduction using span-level edit operations. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 5147–5159, Online. Association for Computational Linguistics.

Johannes von Oswald, Christian Henning, Benjamin F. Grewe, and João Sacramento. 2020. Continual learning with hypernetworks. In *International Conference on Learning Representations (ICLR)*.

Dingbang Wang, Yu Zhao, Sidong Feng, Zhaoxu Zhang, William G. J. Halfond, Chunyang Chen, Xiaoxia Sun, Jiangfan Shi, and Tingting Yu. 2024. Feedback-driven automated whole bug report reproduction for android apps. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2024, page 1048–1060, New York, NY, USA. Association for Computing Machinery.

Xidong Wu, Shangqian Gao, Zeyu Zhang, Zhenzhen Li, Runxue Bao, Yanfu Zhang, Xiaoqian Wang, and Heng Huang. 2024a. Auto-train-once: Controller network guided automatic network pruning from scratch. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16163–16173.

Xidong Wu, Sumin Jo, Yiming Zeng, Arun Das, Ting-He Zhang, Parth Patel, Yuanjing Wei, Lei Li, Shou-Jiang Gao, Jianqiu Zhang, Dexter Pratt, Yu-Chiao Chiu, and Yufei Huang. 2024b. Regulogpt: Harnessing gpt for end-to-end knowledge graph construction of molecular regulatory pathways. In *2024 IEEE EMBS International Conference on Biomedical and Health Informatics (BHI)*, pages 1–8.

Dawei Xiang, Wenyan Xu, Kexin Chu, Tianqi Ding, Zixu Shen, Yiming Zeng, Jianchang Su, and Wei Zhang. 2025. PromptSculptor: Multi-agent based text-to-image prompt optimization. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 774–786, Suzhou, China. Association for Computational Linguistics.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. 2025a. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

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

Li Zeng, Zeming Liu, Chong Feng, Heyan Huang, and Yuhang Guo. 2025a. DocMEdit: Towards document-level model editing. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 19725–19743, Vienna, Austria. Association for Computational Linguistics.

Yiming Zeng, Wanhao Yu, Zexin Li, Tao Ren, Yu Ma, Jinghan Cao, Xiyan Chen, and Tingting Yu. 2025b. Bridging the editing gap in LLMs: FineEdit for precise and targeted text modifications. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 2193–2206, Suzhou, China. Association for Computational Linguistics.

Haopeng Zhang, Hayate Iso, Sairam Gurajada, and Nikita Bhutani. 2024. XATU: A fine-grained instruction-based benchmark for explainable text updates. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 17739–17752, Torino, Italia. ELRA and ICCL.

Zhenjun Zhao. 2024. Balf: Simple and efficient blur aware local feature detector. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3362–3372.

A Parameter Settings

Chunking long context: Many large language models impose a fixed maximum token length L on their input (and sometimes output) sequences. Consequently, if the combination of T_{orig} and I_{edit} exceeds this limit, we divide the T_{orig} into smaller chunks of size $\leq L$. Each chunk is then processed independently—paired with the same edit request and later concatenated to form the complete edited text. This approach ensures that every chunk fits within the model's token budget, preventing overflow and reducing memory usage while preserving the overall structured editing behavior.

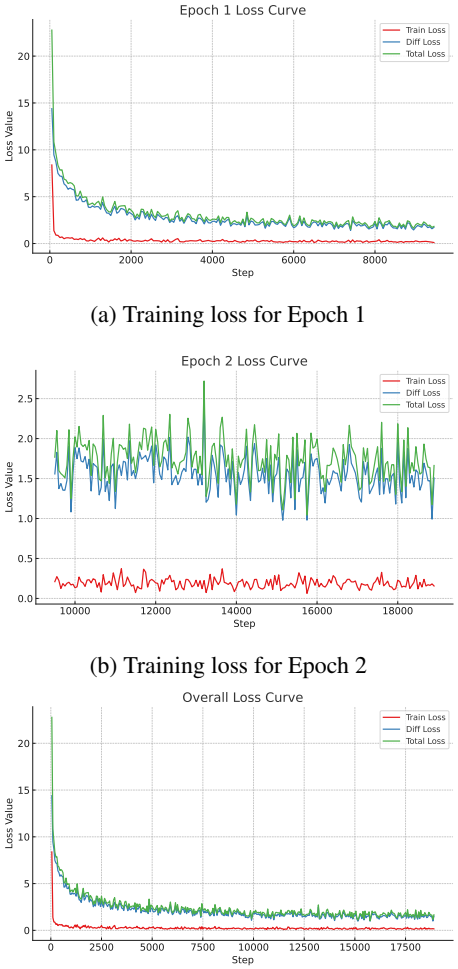
Fine-Tuning Strategy: We use Low-Rank Adaptation (LoRA) (Hu et al., 2022) to efficiently adapt these models to our task, significantly reducing the number of trainable parameters while preserving their expressive power. In all LoRA configurations, we set the rank $r = 8$ and scaling $\alpha = 32$, and use a dropout probability of 0.05. For both Llama-based and Qwen-based models, we apply LoRA to the attention's projection layers through trainable low-rank matrices. We used the AdamW optimizer with a learning rate of 2×10^{-5}, training for 2 epochs, and set the effective batch size of 1 with gradient accumulation steps of 4 due to device limits. This strategy not only reduces computational overhead but also enables rapid convergence on our structured editing tasks. Preliminary experiments guided the choice of hyperparameters across all three model variants.

Decoding and Inference: During generation, we set the temperature to 0.7 and used top-p sampling with a probability of 0.9 to balance diversity and coherence. Greedy decoding is applied by default if without sampling setting. The final edited text is obtained by merging the edited outputs from all chunks.

B Training Loss

Overview. Figure 4 presents the evolution of the training losses during model optimization, including the training loss, diffusion loss, and total loss. The curves are displayed separately for Epoch 1, Epoch 2, and the overall training process.

Epoch 1. At the start of training, all loss components are relatively large, indicating that the model parameters are far from an optimal configuration. A rapid decline is observed within the first few hundred steps, reflecting efficient early learning of the data distribution. After approximately 2,000–3,000 steps, the losses begin to stabilize, suggesting that the model gradually approaches convergence. The total loss follows the same downward trend, showing consistent optimization across loss components.



(a) Training loss for Epoch 1

(b) Training loss for Epoch 2

(c) Overall Training Loss

Figure 4: Training loss for HyperEdit

Epoch 2. During the second epoch, the overall magnitude of the losses remains much lower compared with Epoch 1. The training loss becomes smoother, showing only small fluctuations. This indicates that the model mainly performs fine adjustments to the learned representations rather than major updates. The diffusion loss also remains stable within a narrow range, suggesting that the optimization process has reached a steady state.

C Efficiency and Computational Cost Analysis

To comprehensively evaluate the efficiency of HyperEdit compared to baselines, we conducted a detailed analysis of runtime latency, memory consumption, and theoretical computational complex-

ity (FLOPs).

We measured the inference latency and peak memory usage on a single NVIDIA RTX A6000 GPU. For each domain (Code, LaTeX, DSL, Wikipedia), we randomly sampled 100 instances. The results, summarized in Table 5, compare the base model (Qwen-2.5-Instruct-3B), FineEdit-Pro, and HyperEdit.

Runtime Analysis. As shown in Table 5, the base model averages 32.17 seconds per instance. FineEdit-Pro, which utilizes standard LoRA, averages 53.15 seconds. HyperEdit averages 53.83 seconds, showing a negligible increase over FineEdit-Pro. Despite introducing a hypernetwork to generate request-specific parameters, HyperEdit's inference latency remains comparable to the static LoRA baseline. This is because the parameter generation occurs once per request (or is parallelizable), and the decoding process dominates the total runtime.

Memory Footprint. In terms of memory usage, the base model exhibits a peak of 9.96GB, while FineEdit-Pro reaches 10.6GB. HyperEdit requires a peak memory of 18.7GB. This increase is expected, as HyperEdit maintains dynamic, instance-specific adapters for each layer in memory during inference, unlike standard LoRA which can share or merge static weights.

Category	Qwen-2.5-instruct-3B	FineEdit-Pro	HyperEdit	Count
Code	20.61s	33.33s	37.49s	100
LaTeX	30.70s	48.34s	56.22s	100
DSL	47.17s	78.27s	64.42s	100
Wikipedia	30.21s	52.66s	57.20s	100
Average	**32.17s**	**53.15s**	**53.83s**	-
Peak Memory	**9.96GB**	**10.60GB**	**18.70GB**	-

Table 5: Average inference time per instance across different domains. All models are evaluated on the same NVIDIA A6000 GPU.

Theoretical FLOPs Analysis We further analyze the theoretical floating-point operations (FLOPs) to validate the efficiency of our approach.

The backbone for all models is Qwen-2.5-Instruct-3B, which contains approximately 3 billion parameters. For decoder-only Transformers, the inference cost is widely estimated as $2 \times N$ FLOPs per token, where N is the parameter count (Casson, 2023). Thus, the baseline cost is approximately $2 \times 3 \times 10^9 = 6$ GFLOPs per token. Standard LoRA introduces low-rank matrices with a computational overhead of $O(r/d)$, where r is the rank and d is the hidden dimension. Since $r \ll d$, this overhead is typically less than 1%, keeping the inference cost effectively unchanged at ≈ 6 GFLOPs per token.

HyperEdit builds upon the 3B-parameter backbone. Its additional components include a SentenceTransformer encoder and a GRU-based HyperLoRA module.

Hypernetwork Overhead: The hypernetwork generates the adapter weights. This process involves the sentence encoder (executed once per prompt) and the weight generation network. The complexity of generating rank-r updates is proportional to $O(r/d)$, which is computationally lightweight compared to the dense projections of the backbone. paragraphInference Cost: Once weights are generated, the token-by-token decoding follows the same path as standard LoRA.

Consequently, the total inference complexity of HyperEdit is dominated by the backbone model. The per-token cost remains approximately 6 GFLOPs, with only marginal overhead incurred during the encoding phase. This confirms that HyperEdit achieves dynamic adaptation without a significant penalty in computational throughput.

D Implementation of Edited Span Extraction

The validity of our proposed metrics, Diff-BLEU and Diff-ROUGE-L, relies on the accurate isolation of edited spans from the invariant context. To achieve this, we developed a deterministic, hierarchical extraction algorithm 1 that processes reference and candidate texts through a multi-stage pipeline. This process ensures that the evaluation focuses exclusively on semantic modifications while robustly handling formatting noise.

Semantic Normalization. Raw textual comparisons are often sensitive to trivial formatting variations, such as indentation changes or trailing whitespace, which do not reflect semantic errors in code or natural language. To mitigate this, we apply a regular expression-based normalization to both the reference ground truth (Y) and the model prediction ($\hat{Y}$). This operation collapses consecutive whitespace characters into a single space and strips boundary voids, ensuring that the subsequent alignment is driven by token identity rather than stylistic artifacts.

Coarse-Grained Line Alignment. We utilize the `SequenceMatcher` algorithm (based on the

Ratcliff-Obershelp pattern matching) to align the normalized texts at the line level. The algorithm identifies the Longest Common Subsequence (LCS) of lines and categorizes textual segments into four distinct operation codes:

- `equal`: Contextual lines present in both sequences. These are immediately discarded to prevent the inflation of metrics by unchanged content.
- `delete`: Lines present in Y but absent in $\hat{Y}$.
- `insert`: Lines present in $\hat{Y}$ but absent in Y.
- `replace`: Blocks of lines where the content has been modified but corresponds positionally.

Fine-Grained Span Refinement. For segments tagged as *replace*, a simplistic line-level extraction would be imprecise if only a single token within a long line was altered. To address this, our algorithm performs a recursive, character-level alignment strictly within the replace blocks.

- We isolate the specific sub-sequences within the line that differ, distinguishing between original tokens (t_{old}) and target tokens (t_{new}).
- This ensures that even within a modified line, unchanged distinct tokens are excluded from the evaluation spans, maximizing the sensitivity of the metric.

Span Aggregation for Metric Calculation. Finally, we aggregate the extracted components to construct the evaluation sets. The reference span set $Y_{\text{edit}}^{\text{span}}$ is constructed by concatenating: (1) segments marked as *insert*, (2) the t_{new} components from *replace* operations, and (3) the *delete* segments (represented explicitly with special markers, e.g., `[-token-]`). Similarly, the candidate span set $\hat{Y}_{\text{edit}}^{\text{span}}$ is formed by concatenating the corresponding model-predicted segments. By including explicit representations of deletions, our metric evaluates not only the quality of generated text but also the precision of deletion operations. These aggregated spans serve as the direct inputs for Eq. (10) and Eq. (11).

Algorithm 1: Hierarchical Edited Span Extraction Algorithm

Input: Reference text Y, Candidate text $\hat{Y}$
Output: Reference Span Set $Y_{\text{edit}}^{\text{span}}$, Candidate Span Set $\hat{Y}_{\text{edit}}^{\text{span}}$

`// Step 1: Semantic Normalization`
$\tilde{Y} \leftarrow \texttt{Normalize}(Y)$;
$\tilde{\hat{Y}} \leftarrow \texttt{Normalize}(\hat{Y})$;
`// Step 2: Coarse-Grained Line Alignment`
$\mathcal{M} \leftarrow \texttt{SequenceMatcher}(\tilde{Y}, \tilde{\hat{Y}})$; `// Get opcodes: equal, insert, delete, replace`
$Y_{\text{edit}}^{\text{span}} \leftarrow \epsilon, \quad \hat{Y}_{\text{edit}}^{\text{span}} \leftarrow \epsilon$;
`// Step 3: Iterative Processing & Fine-Grained Refinement`
foreach *operation tuple* $m = \langle tag, i_1, i_2, j_1, j_2 \rangle \in \mathcal{M}$ **do**
  if $tag =$ `equal` **then**
    continue ; `// Discard invariant context`
  else if $tag =$ `insert` **then**
    `// Append inserted segments`
    $Y_{\text{edit}}^{\text{span}} \leftarrow Y_{\text{edit}}^{\text{span}} \oplus \tilde{\hat{Y}}[j_1 : j_2]$;
    $\hat{Y}_{\text{edit}}^{\text{span}} \leftarrow \hat{Y}_{\text{edit}}^{\text{span}} \oplus \tilde{\hat{Y}}[j_1 : j_2]$;
  else if $tag =$ `delete` **then**
    `// Append marked deletions`
    $del\_seg \leftarrow \texttt{MarkDelete}(\tilde{Y}[i_1 : i_2])$;
    $Y_{\text{edit}}^{\text{span}} \leftarrow Y_{\text{edit}}^{\text{span}} \oplus del\_seg$;
    $\hat{Y}_{\text{edit}}^{\text{span}} \leftarrow \hat{Y}_{\text{edit}}^{\text{span}} \oplus del\_seg$;
  else if $tag =$ `replace` **then**
    `// Step 4: Recursive Character-Level Refinement`
    $(t_{old}, t_{new}) \leftarrow \texttt{FineGrainedRefinement}(\tilde{Y}[i_1 : i_2], \tilde{\hat{Y}}[j_1 : j_2])$;
    $Y_{\text{edit}}^{\text{span}} \leftarrow Y_{\text{edit}}^{\text{span}} \oplus t_{new}$; `// Target content`
    $\hat{Y}_{\text{edit}}^{\text{span}} \leftarrow \hat{Y}_{\text{edit}}^{\text{span}} \oplus t_{new}$; `// Generated content`
return $Y_{edit}^{span}, \hat{Y}_{edit}^{span}$