Bridging the Editing Gap in LLMs: FineEdit for Precise and Targeted Text Modifications

Yiming Zeng[1*], **Wanhao Yu**[2*], **Zexin Li**[3], **Tao Ren**[4],
Yu Ma[5], **Jinghan Cao**[6], **Xiyan Chen**[4], **Tingting Yu**[1†]

[1]University of Connecticut, [2]University of North Carolina at Charlotte,
[3]University of California, Riverside, [4]University of Pittsburgh,
[5]Carnegie Mellon University, [6]San Francisco State University

[1]{yiming.zeng, tingting.yu}@uconn.edu, [2]wyu6@charlotte.edu, [3]zli536@ucr.edu,
[4]{tar118, xic130}@pitt.edu, [5]yuma13926@gmail.com, [6]jcao3@alumni.sfsu.edu

Abstract

Large Language Models (LLMs) have significantly advanced natural language processing, demonstrating strong capabilities in tasks such as text generation, summarization, and reasoning. Recently, their potential for automating precise text editing tasks across specialized domains, such as programming code, LaTeX, and structured database languages, has gained attention. However, current state-of-the-art LLMs still struggle with executing precise, instruction-driven edits, particularly when structural accuracy and strict adherence to domain conventions are required. To address these challenges, we introduce InstrEditBench, an automated benchmark dataset comprising over 30,000 structured editing tasks spanning diverse domains, including Wikipedia articles, LaTeX documents, source code, and database languages. Using this benchmark, we develop FineEdit, a specialized editing model explicitly trained for accurate, context-aware text modifications. Experimental evaluations demonstrate that FineEdit outperforms state-of-the-art models, achieving improvements of approximately 10% over Gemini models on single-turn edits, up to 30% over Llama-3.2-3B, and exceeding Mistral-7B-OpenOrca performance by over 40% on direct editing tasks. FineEdit also effectively generalizes to realistic multi-turn editing scenarios, highlighting its practical applicability. To facilitate further research and reproducibility, we release FineEdit at https://github.com/StuRinDQB/FineEdit and https://huggingface.co/datasets/YimingZeng/FineEdit_bench.

1 Introduction

Large Language Models (LLMs) have brought transformative progress to the field of natural language processing, demonstrating remarkable capabilities in text generation, summarization, and reasoning (Chen et al., 2022; Achiam et al., 2023; Chen et al., 2023, 2024; Wu et al., 2024; Li et al., 2025; Liang et al., 2025). Recently, LLMs have received increasing attention for their potential to automate and enhance text editing across a variety of domains (Celikyilmaz et al., 2020). Such editing capabilities is particularly needed under task-specific application scenarios, e.g., code editing (Fan et al., 2024; Lei et al., 2025), Wiki editing (Suri et al., 2024), etc.

Despite this promise, current LLMs still face notable limitations when applied to tasks that demand direct editing, where the model must simultaneously understand the original text, follow the instruction precisely, and generate semantically aligned, high-quality edits. Even powerful proprietary tools like ChatGPT often struggle to fully understand user intent and reliably follow strict editing instructions, especially in long-context scenarios (Castillo-González et al., 2022). Particularly, LLMs' general editing capabilities in task-specific settings often fall short (Yao et al., 2023; Ma et al., 2024). They tend to generate incorrect outputs and stray from the given editing instructions.

To address these challenges, we propose a more focused approach to editing with LLMs. Our key insight is that narrowing the model's attention to two fundamental aspects, the exact location of the edit and the content to be modified, can significantly improve performance in direct editing tasks. Per this intuition, we propose a dual approach consisting of a dedicated benchmark (InstrEditBench) for editing tasks and an editing-specific model (FineEdit). Specifically, we design an automated workflow that focuses on accurately identifying and evaluating structured text edits. This workflow

[*]Equal contribution.
[†]Corresponding author.

identifies precise differences and ensures correct edits through quality control. By reducing noise and focusing on meaningful modifications, this process produces a dedicated, high-quality benchmark. It directly addresses limitations in existing methods and aligns better with the practical demands of real-world editing tasks. Notably, our approach is also generalized to multi-turn editing scenarios, a much more realistic user scenario, where instructions arrive iteratively and allow the model to refine its edits step by step.

Implementation and evaluation. We train the FineEdit model on InstrEditBench benchmark, explicitly designed to optimize performance on direct, instruction-driven text editing tasks. The result shows that FineEdit achieves an improvement of 10% over Gemini 1.5 Flash and Gemini 2.0 Flash (DeepMind, 2024) in single-turn editing tasks, and up to 30% over Llama-3.2-3B (Meta AI, 2024) on diverse editing benchmarks, while outperforming Mistral-7B-OpenOrca (Lian et al., 2023; Mukherjee et al., 2023; Longpre et al., 2023) over 40% on direct editing tasks.

The main contributions of this work include:

- **A high-quality benchmark (InstrEditBench)**: We introduce the first systematically constructed benchmark that spans four diverse domains and contains more than 30,000 single-turn and multi-turn structured editing tasks, thereby establishing a unified and comprehensive evaluation standard for direct editing research.
- **An innovative automated dataset generation workflow**: We develop a comprehensive workflow that ensures the benchmark's quality by accurately identifying line numbers and applying rigorous criteria to filter meaningful and relevant edits.
- **The FineEdit model**: We present a specialized model designed for direct text editing, demonstrating superior performance across benchmarks compared with existing models.

2 Background

2.1 Problem Formulation

Each data point consists of an original structured text, T_{orig}, and an editing instruction, I_{edit}. The objective is to generate an edited text, T_{edit}, that incorporates the modifications specified by I_{edit}. Formally, this process is defined as

$$T_{\text{edit}} = f\Big(T_{\text{orig}}, I_{\text{edit}}; \theta\Big) \tag{1}$$

where θ represents learned parameters and f denotes a function instantiated by a LLM that maps the original text T_{orig} and editing instruction I_{edit} to the edited text T_{edit}.

The parameters θ are learned from a dataset consisting of triples $\{(T^{(i)}_{\text{orig}}, I^{(i)}_{\text{edit}}, T^{(i)}_{\text{edit}})\}_{i=1}^{N}$ during training, where the objective is to minimize the discrepancy between the generated output and the ground truth edited text.

Internally, f concatenates T_{orig} and I_{edit} into a single prompt and generates T_{edit} token by token in an autoregressive manner. Specifically, if $T_{\text{edit}} = (y_1, y_2, \ldots, y_t)$, the probability of the edited text is factorized as

$$p(T_{\text{edit}} \mid T_{\text{orig}}, I_{\text{edit}}) = \prod_{i=1}^{t} p\Big(y_i \mid T_{\text{orig}}, I_{\text{edit}}, y_1, y_2, \ldots, y_{i-1}\Big) \tag{2}$$

For finetuning on the editing task, the prompt tokens (i.e., the original text and the editing instruction) are masked out in the loss function to ensure that the model focuses only on predicting the correct edited tokens. At inference time, the model processes the prompt and subsequently generates T_{edit}.

The parameters θ are fine-tuned on labeled examples $(T_{\text{orig}}, I_{\text{edit}}, T_{\text{edit}})$ by minimizing the negative log-likelihood of the target tokens with the loss:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{|T_{\text{edit}}|} \log P_\theta(y_t \mid T_{\text{orig}}, I_{\text{edit}}, y_{1:t-1}) \tag{3}$$

over all training samples in the dataset.

2.2 LLM Editing Tasks

LLMs are increasingly recognized as versatile tools for automating and enhancing editing tasks across diverse domains. Previous studies have explored LLMs for editing tasks in areas such as natural language (e.g., wiki articles) and code. For instance, CoEdIT (Raheja et al., 2023) employs task-specific instruction tuning to achieve precise modifications, while other works fine-tune models like T5 (Raffel et al., 2020) on pairs of original and edited texts (Bryant et al., 2019; Stahlberg and Kumar, 2021; Pezeshkpour, 2023). However, many of these

approaches rely on specialized techniques or focus narrowly on specific tasks, such as grammar correction (Katinskaia and Yangarber, 2023; Bout et al., 2023), text simplification (Sun et al., 2023), paraphrase generation (Palivela, 2021), or style transfer (Luo et al., 2023), which limits their generalizability across a broader range of editing scenarios. In the realm of code editing, Dilhara et al. (Dilhara et al., 2024) examined LLMs for code change tasks and identified weaknesses in generating accurate reviews and commit messages. Beyond single-turn editing, iterative or multi-turn editing can further improve output quality by allowing incorporation of progressive feedback, leading to more accurate and context-aligned modifications (Schick et al., 2022; Madaan et al., 2023). While these studies offer valuable insights, they often fall short in providing unified benchmarks and robust solutions to address the full spectrum of editing challenges. Our work addresses these gaps by introducing a comprehensive, cross-scenario editing tasks benchmark that covers Wiki, code, DSL, and LaTeX.

3 Method

3.1 Instruction categories

We leverage four data sources to cover a wide range of representative text application scenarios: Wiki, Code, DSL, and LaTeX. The details of each categories are described as follows:

- **Wiki**: Data is extracted from the WikiText language modeling dataset (Merity et al., 2016), which contains over 100 million tokens from a dedicated subset of Wikipedia's Good articles (Wikipedia, n.d.b) and Wikipedia's Featured articles (Wikipedia, n.d.a). Specifically, sections from these articles are extracted and then contiguous segments are randomly selected to provide data points with various lengths.

- **Code**: Code samples are extracted from the CodeSearchNet corpus (Husain et al., 2019), which contains about two million pairs of comments and code from GitHub projects. To make the edit task more challenging, each code sample in our benchmark is made up of several instead of one code segment because one single code segment is too short (about 10 lines).

- **DSL**: Database Domain Specific Language (DSL) is also considered in our benchmark. It consists of queries and schema definitions from multiple public repositories (b mc2, 2023; hive, 2024; cassandra, 2024; Lerocha, 2024).

- **LaTeX**: LaTeX data is extracted from the Latex2Poster dataset (Latex2Poster, 2024) that offers the LaTeX source code document of research papers along with metadata. Specifically, each data point in our benchmark consists of multiple subsections from each extracted document data.

3.2 Instruction Generation

Zero-shot instruction generation is efficient, but often lacks diversity. To address this limitation, we build on the work of (Wang et al., 2022; Taori et al., 2023) by leveraging ChatGPT-4o mini combined with in-context learning (ICL) (Dong et al., 2024). Our approach is designed to generate specific edit requests tailored to the structural characteristics of different data categories, as process ① in Figure 1. For Wiki, which primarily consists of clear structural text elements like headings and subheadings, we apply a zero-shot prompting strategy. In contrast, for more complex domains such as LaTeX, code, and DSL, we adopt ICL to improve the diversity and nuance of generated instructions.

This category-specific strategy not only enriches the instruction sets but also enhances their ability to capture domain-specific editing challenges without compromising on precision and efficiency. We will describe prompt details in Appendix C.

3.3 Instruction filtering

After obtaining the edit instructions for each content, we apply them to the original text to produce an edited version as process ② in Figure 1. However, ensuring the quality of the edited content remains challenging. Although LLM generally follows the edit instructions, errors may occur—for example, targeting incorrect line numbers or misinterpreting the intended semantics (Cassano et al., 2024; Wang et al., 2025). To address this problem and improve data quality, we propose DiffEval Pipeline, which integrates G-Eval (Liu et al., 2023) and Git-Diff as an automatic filter to improve data quality.

Besides adopting G-Eval for automated assessment (Liu et al., 2023), the DiffEval Pipeline also relies on `git` (git, 2024), a widely used version control system, to detect and classify textual mod-

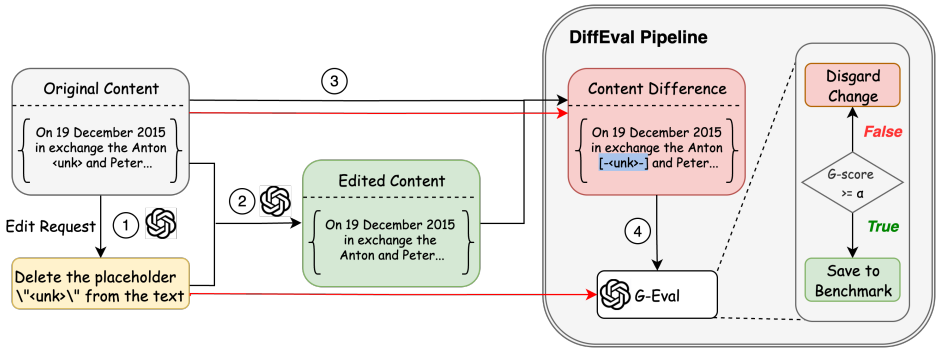

Figure 1: Workflow of Generating High-quality InstrEditBench. The content difference is highlighted in blue.

ifications. Specifically, the command `git diff` specifies differences between the original and modified texts as process ③ in Figure 1, categorizing changes into four types:

- **Replacements**: an original segment is transformed into a new form, indicated as `[original_text -> modified_text]`. This captures cases where an existing text portion is substituted with different content, which may alter meaning or style.
- **Deletions**: a segment is removed entirely, shown as `[-original_text-]`. Such removals can simplify the text or eliminate irrelevant or erroneous sections.
- **Insertions**: new content is added, denoted as `[+modified_text+]`. Insertions enrich the text with extra details, clarifications, or elaborations.
- **Unchanged Text**: labeled as `equal: unchanged_text`. This indicates portions that remain identical between the original and modified versions, providing a reference for what the model has chosen to retain.

By categorizing changes into these four types, the DiffEval Pipeline offers a structured view of how text is altered, enabling more precise evaluations when paired with G-Eval.

Finally, process ④ in Figure 1 demonstrates that DiffEval carefully reviews the aggregated data (marked with red arrows) alongside the edit request to fully grasp the context, structure, and nuances of the text. It identifies discrepancies between the intended edits and the actual modifications, verifying whether the changes faithfully implement the edit instructions. By using the `git diff` output instead of the complete edited content, DiffEval can precisely locate modifications using supplementary information such as line numbers and structured differences. Moreover, `git diff` minimizes unnecessary noise and reduces computational overhead by significantly lowering the token count compared with the full edited content. Once all required data is gathered, the G-Eval analysis process evaluates the collected information to further enhance the dataset quality.

Specifically, the analysis process begins by parsing the structure of `git diff` outputs, categorizing changes as replacements, deletions, insertions, or unchanged segments. Next, it evaluates the semantic meaning of both the original content and the modifications to ensure that the changes are accurate and complete. This involves a thorough review of the original text, the edit request, and the resulting edits, applying predefined categorization rules, and assessing overall coherence.

Based on this analysis process, the DiffEval can assign a coherence score, `G-Score`, to the edited content, reflecting the semantic integrity and logical consistency of the modifications. This score is used to filter out output that does not meet the desired quality threshold α.

3.4 Generalize to Multi-turn Editing Task

Notably, our proposed framework is easily generalized to more practical multi-turn editing scenarios. Specifically, given the initial content, we instruct ChatGPT to generate a sequence of multiple distinct editing requests that are explicitly constrained to be non-contradictory with each other. Each generated editing request targets different aspects or details within the same content, ensuring that subsequent instructions complement rather than conflict with previous edits. This setting reflects real-world editing workflows where users iteratively refine content through consecutive instructions.

4 Experiment

4.1 Experimental Setup

In this section, we detail the experimental setups, including dataset splits, model variants, baselines, evaluation metrics, and implementation specifics.

Dataset and Model Variants. We evaluate FineEdit on our proposed InstrEditBench dataset using a 90/10 train-test split. Additionally, we introduce three FineEdit variants, namely FineEdit-L, FineEdit-XL, and FineEdit-Pro, which are fine-tuned from the LLaMA-3.2-1B, LLaMA-3.2-3B, and Qwen2.5-3B-Instruct base models respectively, covering a broad spectrum of model architectures and parameter scales.

Baselines. Our baselines include Gemini 1.5 Flash, Gemini 2.0 Flash, LLaMA-3.2-1B, LLaMA-3.2-3B, Qwen2.5-3B-Instruct, and Mistral-7B, spanning diverse architectures and sizes. We evaluate both zero-shot and few-shot prompting on the Gemini models, while open-source models are assessed using zero-shot prompting.

Metrics. Following established approaches (Shen et al., 2017; Nakamachi et al., 2020), we use BLEU and ROUGE-L metrics to assess the vocabulary and structural consistency between the edited and reference texts.

Implementation details. Training details are provided in Appendix A.

4.2 Performance of Existing Models

We evaluated FineEdit against several state-of-the-art baselines on the InstrEditBench dataset across four data categories as presented in Table 3.

Comparison with Zero-shot Performance. Among all baselines, Gemini 1.5 Flash achieved the highest overall scores, while Mistral-7B-OpenOrca recorded the lowest BLEU and ROUGE-L values. Although model size is typically a critical factor, Gemini 2.0 Flash did not outperform Gemini 1.5 Flash in terms of overall effectiveness. For example, despite having more parameters than LLaMA-3.2-1B, Mistral-7B-OpenOrca underperformed on both metrics, highlighting the importance of model architecture and training strategies. Additionally, Gemini 2.0 Flash demonstrated superior semantic understanding in the Wiki category, with a BLEU score of 0.9133 and a ROUGE-L score of 0.9429, yet its overall performance remained inferior to that of Gemini 1.5 Flash.

FineEdit, and in particular its FineEdit-Pro variant, further outperforms all zero-shot baselines. FineEdit-Pro achieves an overall BLEU score of 0.9245, representing improvements of approximately 11.6%, 57.7%, and 184.7% over Gemini 1.5 Flash (0.8285), LLaMA-3.2-3B (0.5862), and Mistral-7B-OpenOrca (0.3246), respectively. These gains are consistently observed across individual data categories—for example, FineEdit-Pro attains BLEU scores of 0.9521 and 0.9538 in the DSL and Code domains, respectively. These results underscore the effectiveness of FineEdit's targeted fine-tuning strategy, which focuses on precise editing of location and content to preserve both structural and semantic integrity.

Comparison with Few-shot Performance. We further evaluated few-shot learning on the Gemini models. Although few-shot prompting notably improved performance in some categories, such as the LaTeX domain where Gemini 2.0 Flash achieved a 20% higher BLEU score compared to the zero-shot setting, the overall few-shot results still remained inferior to FineEdit. In certain cases, such as the SQL category, few-shot learning provided minimal improvement, achieving BLEU and ROUGE-L scores of only 0.1600 and 0.1814, respectively. These findings highlight the effectiveness and importance of our curated benchmark in driving advancements in editing tasks.

4.3 FineEdit: Supervised Finetuning

Our FineEdit model is offered in three variants: FineEdit-L, FineEdit-XL, and FineEdit-Pro. Under zero-shot conditions, FineEdit-L consistently outperforms all baseline models in BLEU and ROUGE-L scores for LaTeX, DSL, Wiki, and Code tasks. For example, compared to Gemini 1.5 Flash, FineEdit-L improves overall BLEU scores by roughly 8%, with even larger gains observed in specific categories. Notably, FineEdit-XL performs similarly to FineEdit-L, suggesting that increasing the parameter count from 1B to 3B using LLaMA does not yield a significant performance boost.

By leveraging the superior instruction-following capabilities of Qwen2.5-3B-Instruct, our final variant, FineEdit-Pro, further elevates performance. FineEdit-Pro achieves an overall BLEU score of 0.9245, which represents improvements of approximately 11.6% over Gemini 1.5 Flash, and gains of around 14.7% and 11.7% in the DSL and Wiki tasks, respectively. These consistent improvements across multiple data categories underscore the ef-

Method	Model	Open-Source	LaTeX		DSL		Wiki		Code		Overall	
			BLEU	RG-L	BLEU	RG-L	BLEU	RG-L	BLEU	RG-L	BLEU	RG-L
Zero-shot	Gemini 1.5 Flash	✗	0.8665	0.9150	0.8297	0.8555	0.7626	0.8361	0.8551	0.9073	0.8285	0.8819
	Gemini 2.0 Flash	✗	0.7413	0.7951	0.4706	0.4964	0.9133	0.9429	0.1339	0.2737	0.5853	0.6519
	Llama-3.2-1B	✓	0.5088	0.6108	0.5564	0.6596	0.4413	0.5766	0.4742	0.6072	0.4867	0.6069
	Llama-3.2-3B	✓	0.5969	0.6925	0.5747	0.6821	0.5061	0.6384	0.6638	0.7727	0.5862	0.6976
	Qwen2.5-3B-Instr	✓	0.5467	0.6712	0.4107	0.4991	0.4170	0.5699	0.3967	0.5390	0.4492	0.5816
	Mistral-7B-Orca	✓	0.3782	0.5770	0.0361	0.1638	0.3608	0.5840	0.3763	0.6447	0.3246	0.5395
Few-shot	Gemini 1.5 Flash (2 shot)	✗	0.8742	0.9324	0.0908	0.1190	0.8657	0.9139	0.7412	0.8302	0.7249	0.7845
	Gemini 2.0 Flash (2 shot)	✗	0.9464	0.9723	0.1600	0.1814	**0.9380**	**0.9665**	0.8327	0.8698	0.8011	0.8302
FineEdit	FineEdit-L	✓	0.9311	0.9697	0.9334	0.9615	0.8077	0.9036	0.9296	0.9725	0.8957	0.9504
	FineEdit-XL	✓	0.8867	0.9502	0.9241	0.9552	0.8120	0.9056	0.9295	0.9720	0.8824	0.9441
	FineEdit-Pro	✓	**0.9539**	**0.9821**	**0.9521**	**0.9710**	0.8521	0.9185	**0.9538**	**0.9836**	**0.9245**	**0.9628**

Table 1: Comparison of LLMs on BLEU and ROUGE-L for LaTeX, DSL, Wiki, Code. Overall data displays average performance among all data categories. The best results are highlighted in bold.

Method	Model	Open-Source	LaTeX		DSL		Wiki		Code		Overall	
			BLEU	RG-L	BLEU	RG-L	BLEU	RG-L	BLEU	RG-L	BLEU	RG-L
Zero-shot	Gemini 1.5 Flash	✗	0.1745	0.3067	0.9643	0.9787	0.7785	0.8908	0.3882	0.5291	0.5764	0.6763
	Gemini 2.0 Flash	✗	0.1304	0.2435	0.4243	0.4328	0.8624	0.9145	0.1426	0.2454	0.3899	0.4591
	Llama-3.2-1B	✓	0.3168	0.4371	0.2593	0.3466	0.2291	0.3584	0.1749	0.3196	0.2450	0.3654
	Llama-3.2-3B	✓	0.3101	0.4374	0.3274	0.4355	0.2871	0.4073	0.2988	0.4177	0.3059	0.4245
	Qwen2.5-3B-Instr	✓	0.5196	0.6344	0.2083	0.2603	0.2845	0.4261	0.3985	0.5138	0.3527	0.4587
Few-shot	Gemini 1.5 Flash (2 shot)	✗	0.4811	0.5423	0.0511	0.1167	0.7511	0.8462	0.2388	0.3430	0.3805	0.4621
	Gemini 2.0 Flash (2 shot)	✗	0.9099	0.9247	0.0294	0.0406	0.9272	0.9740	0.4719	0.6266	0.5846	0.6415
FineEdit	FineEdit-L	✓	0.6823	0.8531	0.8071	0.8730	0.4938	0.6588	0.6707	0.7773	0.6635	0.7906
	FineEdit-XL	✓	0.3230	0.4468	0.8050	0.8798	0.4522	0.6333	0.6806	0.7756	0.5652	0.6839
	FineEdit-Pro	✓	0.8461	0.8917	0.8123	0.8902	0.6975	0.8286	0.9499	0.9796	0.8265	0.8975

Table 2: Multi-turn editing results for LaTeX, DSL, Wiki and Code. Overall data displays average performance among all data categories.

fectiveness of our supervised fine-tuning strategy and highlight the importance of a strong instruction-tuned base model over merely increasing model size.

We also compared our models with Gemini's few-shot prompting approach in real-world scenarios. Although in-context learning (ICL) improves Gemini's performance in certain cases, such as an 8% increase in BLEU score on the Wiki dataset for Gemini 2.0 Flash, the overall performance remains inferior to FineEdit-Pro. This superior performance highlights the effectiveness of our high-quality, rigorously validated InstrEditBench dataset in enabling more robust and generalizable solutions for editing tasks.

4.4 Multi-turn Editing Evaluation

We also evaluated the extended benchmark on FineEdit in the multi-turn setting. In this extension, each data instance contains multiple editing requests. To simulate real multi-turn scenario, we apply these instructions iteratively: each request is executed on the output produced by the previous one, ensuring that the edits are applied in a cumulative manner. We then assess the final output to determine whether it accurately reflects the cumulative effect of all editing instructions after the full sequence of modifications has been applied. To assess the performance of multi-turn, we randomly sample 100 multi-turn data for each category, and test them on different models.

Results show that multi-turn editing leads to consistent performance drops across all domains. Specifically, BLEU scores for LaTeX drop from 0.9539 to 0.8461, DSL from 0.9521 to 0.8123, Wiki from 0.8521 to 0.6975, and Code from 0.9538 to 0.9499. These results indicate that multi-turn scenarios are substantially more challenging, especially for Wiki and DSL, while code exhibit strong robustness under multi-turn edits. This indicates that the decline is the accumulation of errors across turns. Each instruction is applied to the output of the previous one, which may already contain small deviations. These deviations can propagate through subsequent steps and lead to compounded errors in the final output.

Edit Request 1:	Change the brackets in the code to semicolons.
Original:	`def test(options): options.data = [] ; def test2(options): options.data = []`
Gemini (✗)	`def test(options): options.data = None ; def test2(options): options.data = None`
FindEdit Pro (✓)	`def test(options): options.data = ; ; def test2(options): options.data = ;`
Edit Request 2:	Change `\subsection{Strengths}` to `\subsection*{Strengths}`.
Original:	`\subsection{Strengths} The topic of responsible AI is ...`
Gemini (✓)	`\subsection*{Strengths} The topic of responsible AI is ...`
FindEdit Pro (✗)	`latex{\subsection{Strengths}} The topic of responsible AI is ...`
Edit Request 3:	Replace "Falcon" with "Captain America".
Original:	In "Captain America: Brave New World," Sam Wilson, formerly the Falcon, assumes ...
Gemini (✗)	In "Captain America: Brave New World," Sam Wilson, formerly known as the Falcon, assumes ...
FindEdit Pro (✓)	In "Captain America: Brave New World," Sam Wilson, formerly the Captain America, assumes ...
Edit Request 4:	Add column `created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP`.
Original:	`CREATE TABLE worker_salaries (employee_id INT, ...)`
Gemini (✗)	`ALTER TABLE community_gardens ADD COLUMN created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP`
FindEdit Pro (✓)	`CREATE TABLE community_gardens (..., created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP)`

Table 3: Colour-coded comparison of Gemini and FindEdit Pro responses to four edit requests. (✗ = incorrect) stands for incorrect editing, while (✓ = correct) stands for correct editing.

Despite the degradation in multi-turn settings, FineEdit-Pro achieves an average BLEU of 0.8265, substantially higher than Gemini 1.5 Flash (0.5764) and Gemini 2.0 Flash (0.3899). This further demonstrates the effectiveness of our dataset design and its extensibility to diverse editing scenarios.

4.5 Qualitative Study

To qualitatively assess the performance of FindEdit on single-turn editing tasks, we conduct several studies as shown in Table 3. This table illustrates eight examples of how FineEdit-Pro and Gemini respond to diverse editing requests. In several cases, FineEdit-Pro accurately applies the required changes. Specifically, it could correctly add new columns in DSL or adjust environment commands. However, Gemini often restates the instruction without actually implementing the intended modifications. Specifically, both Gemini 1.5 Flash and 2.0 Flash perform well on LaTeX and Wiki tasks, yet they struggle with DSL and Code tasks. For example, as shown in Table 3, FineEdit-Pro correctly identifies the target table and appends a new column named `created_at` with the data type `DEFAULT CURRENT_TIMESTAMP`. In contrast, Gemini misinterprets the instruction, merely repeating the edit request rather than applying the intended change. These observations highlight the qualitative strengths of our proposed FineEdit approach.

Nonetheless, FineEdit is not without shortcomings. In the LaTeX example depicted in Table 3, Gemini accurately locates the `subsection{Strengths}` and updates it as specified. However, although FineEdit-Pro also identifies and modifies the correct location, it generates the correct response twice, which deviates from the direct editing requirement. This discrepancy suggests that FineEdit-Pro, though generally more reliable, can overapply modifications in specific cases.

Overall, these results illustrate FineEdit-Pro's capacity to handle more complex edits, particularly for DSL and Code, while Gemini often fails to implement them. Nevertheless, occasional issues like duplicate outputs highlight the need for refinement, ensuring FineEdit-Pro consistently adheres to direct editing requirements without introducing redundant content. On the other hand, Gemini occasionally performs better in simpler tasks, such as LaTeX updates.

FineEdit is also generalized well to the multi-turn editing task scenarios. Table 4 demonstrates an example of how FineEdit-Pro perform more precisely than Gemini. In this scenario, FineEdit-Pro successfully applies all three requested changes: it removes the duplicate `\begin{abstract}`, replaces the inline `\footnote{}` with a `\footnotemark` and corresponding `\footnotetext{}` pair, and rewrites the commented `\{TODO...}` as a finalized, explanatory sentence.

In contrast, though Gemini removes the duplicate `\begin{abstract}`, it add a duplicate `\end{abstract}` in the end of this content. Additionally, Gemini does not follow the instruction to split the \footnote{} into `\footnotemark` and `\footnotetext{}`, instead simply retaining the original inline footnote. These errors indicate that Gemini struggles with compound edits that involve

Edit Request 1:	Remove the duplicate \begin{abstract} at the beginning of the abstract environment.
Original:	\begin{abstract} \begin{abstract} . . .
Gemini (✗)	\begin{abstract} . . . \end{abstract} \end{abstract}
FindEdit Pro (✓)	\begin{abstract} . . . \end{abstract}
Edit Request 2:	Convert \footnote{...} to \footnotemark + \footnotetext{...}.
Original:	\footnote{Dataset is available at \url{...}}
Gemini (✗)	\footnote{Dataset is available at \url{...}}
FindEdit Pro (✓)	\footnotemark ... \footnotetext{Dataset is available at \url{...}}
Edit Request 3:	Remove the 'TODO' from the 'TODO' line.
Original:	\{TODO: we introduce distractibility as a new metric for evaluating language models.}
Gemini (✓)	\{We introduce distractibility as a new metric for evaluating language models.}
FindEdit Pro (✓)	\{We introduce distractibility as a new metric for evaluating language models.}

Table 4: Color-coded comparison of Gemini and the FineEdit Pro for a multi-turn task with three edit requests. (✗ = incorrect) indicates an unsatisfied edit, while (✓ = correct) indicates a satisfied edit.

Threshold	Wiki	LaTeX	DSL	Code
G-score ≥ 9	97%	93%	90%	97%
G-score < 9	87%	89%	66%	83%

Table 5: Annotation accuracy across content types under different G-score thresholds.

structural modifications across multiple locations.

4.6 Human Evaluation

To assess whether `DiffEval` improves the overall quality of the dataset, we carried out a human evaluation. Because the dataset includes Code and DSL categories that require programming expertise, we recruited three evaluators, each with at least a bachelor's degree in computer science or a related discipline. We established the following guidelines to ensure rigorous assessment: (1) Precise Observation: Confirm that the updated content exactly corresponds to the segment specified by the edit request. (2) No Unintended Modifications: Verify that no other sections have been altered; any unexpected changes result in failure. (3) Three-Round Procedure: Two evaluators independently review each item, with a third evaluator resolving any discrepancies.

We examined 100 items per category and found that data processed through our DiffEval pipeline exhibited noticeably enhanced accuracy, as shown in Table 5. The Wiki and Code datasets, in particular, demonstrated the most reliable outcomes, with edited content precisely matching the requested modifications. Notably, the DSL dataset experienced the greatest improvement, with quality increasing by over 24% compared to data that did not meet DiffEval's standards.

4.7 Ablation Study on DiffEval Components

To better understand the contribution of each component in the DiffEval pipeline, we conducted two ablation experiments. Manual annotation followed the protocol described in Section 4.6.

`Git diff` effectiveness. We evaluated a reduced pipeline in which G-Eval judged the alignment between the instruction and the edited text without access to `git diff`. From its output, we randomly sampled `100` examples whose `G-Score` was at least nine and annotated them. For comparison, we annotated another `100` examples produced by the full DiffEval pipeline, where G-Eval received the `git diff` instead of the full edited text.

Including `git diff` raised the accuracy from 0.85 to 0.94. These results indicate that `git diff` contributes important structural information for identifying precise alignment between the instruction and the edit.

G-score threshold selection. We also examined the effect of the G-score threshold α used in filtering. Setting $\alpha = 8$ results in many examples where the core instruction is followed, but this will sometimes introduce unintended formatting changes. One common issue is the insertion of extra spaces throughout the text. For instance, in a deletion instruction targeting a historical phrase, the phrase was correctly removed, but the resulting diff introduced multiple superfluous spaces across the paragraph. This violates the requirement to preserve all formatting outside the instructed change. Such formatting issues were significantly reduced when the threshold was increased to $\alpha = 9$.

5 Conclusion

We introduce InstrEditBench, a benchmark of over 30k editing tasks spanning Wiki, LaTeX, code, and

DSL, aimed at precise instruction-based text editing. To ensure supervision quality, we develop DiffEval, an automated pipeline combining structural and semantic filters. We further validate our benchmark with FineEdit, a model fine-tuned on InstrEditBench, achieving up to 10% gains over leading models. Designed for both single-turn and multi-turn editing, our modular benchmark and pipeline enable broad applicability.

6 Limitations

Limited Deployment Scope. Due to cost and hardware constraints, our evaluations were limited to large proprietary LLMs (e.g., Gemini), rather than large open-source models.

Controlled Context Evaluation. Our benchmark focuses on controlled evaluation contexts, where it does not yet encompass long-context chain-of-thought scenarios, as smaller LLMs are confined by limited context windows, even though such techniques could be effective in proprietary models.

7 Acknowledgement

This work was supported by the National Science Foundation under Grant No. CCF-2403747. The opinions, findings, and conclusions or recommendations expressed herein are those of the authors and do not necessarily reflect the views of the National Science Foundation. Meanwhile, we sincerely thank the reviewers for their valuable feedback, which greatly contributed to improving this work.

References

2024. Git diff: A tool for comparing changes. Git Documentation.

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

b mc2. 2023. sql-create-context dataset.

Andrey Bout, Alexander Podolskiy, Sergey Nikolenko, and Irina Piontkovskaya. 2023. Efficient grammatical error correction via multi-task training and optimized training schedule. *arXiv preprint arXiv:2311.11813*.

Christopher Bryant, Mariano Felice, Øistein E. Andersen, and Ted Briscoe. 2019. The BEA-2019 shared task on grammatical error correction. In *Proceedings of the Fourteenth Workshop on Innovative Use of NLP for Building Educational Applications*, pages 52–75, Florence, Italy. Association for Computational Linguistics.

cassandra. 2024. Apache cassandra. GitHub Repository.

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.

William Castillo-González, Carlos Oscar Lepez, and Mabel Cecilia Bonardi. 2022. Chat gpt: a promising tool for academic editing. *Data and Metadata*, 1:23–23.

Asli Celikyilmaz, Elizabeth Clark, and Jianfeng Gao. 2020. Evaluation of text generation: A survey. *arXiv preprint arXiv:2006.14799*.

Yiming Chen, Simin Chen, Zexin Li, Wei Yang, Cong Liu, Robby Tan, and Haizhou Li. 2023. Dynamic transformers provide a false sense of efficiency. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7164–7180, Toronto, Canada. Association for Computational Linguistics.

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.

Yiming Chen, Yan Zhang, Bin Wang, Zuozhu Liu, and Haizhou Li. 2022. Generate, discriminate and contrast: A semi-supervised sentence representation learning framework. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 8150–8161, Abu Dhabi, United Arab Emirates. Association for Computational Linguistics.

Google DeepMind. 2024. Google gemini ai update - december 2024.

Malinda Dilhara, Abhiram Bellur, Timofey Bryksin, and Danny Dig. 2024. Unprecedented code change automation: The fusion of llms and transformation by example. *Proceedings of the ACM on Software Engineering*, 1(FSE):631–653.

Qingxiu Dong, Liangming Pan, Duyu Tang, Ming Gong, Nan Duan, Heyan Huang, and Xiaoyan Zhu. 2024. A survey on in-context learning. *arXiv preprint arXiv:2301.00234*.

Lishui Fan, Jiakun Liu, Zhongxin Liu, David Lo, Xin Xia, and Shanping Li. 2024. Exploring the capabilities of llms for code change related tasks. *Preprint*, arXiv:2407.02824.

hive. 2024. Apache hive. GitHub Repository.

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

Hamel Husain, Ho-Hsiang Wu, Tiferet Gazit, Miltiadis Allamanis, and Marc Brockschmidt. 2019. CodeSearchNet challenge: Evaluating the state of semantic code search. *arXiv preprint arXiv:1909.09436*.

Anisia Katinskaia and Roman Yangarber. 2023. Grammatical error correction for sentence-level assessment in language learning. In *Workshop on Innovative Use of NLP for Building Educational Applications*, pages 488–502. The Association for Computational Linguistics.

Latex2Poster. 2024. Latex2poster dataset. Hugging Face.

Bin Lei, Weitai Kang, Zijian Zhang, Winson Chen, Xi Xie, Shan Zuo, Mimi Xie, Ali Payani, Mingyi Hong, Yan Yan, and Caiwen Ding. 2025. Infantagent-next: A multimodal generalist agent for automated computer interaction. *Preprint*, arXiv:2505.10887.

Lerocha. 2024. Chinook database. GitHub Repository.

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, Vienna, Austria. Association for Computational Linguistics.

Wing Lian, Bleys Goodson, Guan Wang, Eugene Pentland, Austin Cook, Chanvichet Vong, and "Teknium". 2023. Mistralorca: Mistral-7b model instruct-tuned on filtered openorcav1 gpt-4 dataset. https://huggingface.co/Open-Orca/Mistral-7B-OpenOrca.

Jiacheng Liang, Tanqiu Jiang, Yuhui Wang, Rongyi Zhu, Fenglong Ma, and Ting Wang. 2025. Autoran: Weak-to-strong jailbreaking of large reasoning models. *Preprint*, arXiv:2505.10846.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. 2023. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2511–2522.

Shayne Longpre, Le Hou, Tu Vu, Albert Webson, Hyung Won Chung, Yi Tay, Denny Zhou, Quoc V. Le, Barret Zoph, Jason Wei, and Adam Roberts. 2023. The flan collection: Designing data and methods for effective instruction tuning. *Preprint*, arXiv:2301.13688.

Guoqing Luo, Yu Han, Lili Mou, and Mauajama Firdaus. 2023. Prompt-based editing for text style transfer. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 5740–5750.

Xinbei Ma, Tianjie Ju, Jiyang Qiu, Zhuosheng Zhang, Hai Zhao, Lifeng Liu, and Yulong Wang. 2024. On the robustness of editing large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 16197–16216, Miami, Florida, USA. Association for Computational Linguistics.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *Preprint*, arXiv:1609.07843.

Meta AI. 2024. Llama 3.2: Revolutionizing edge ai and vision with open, customizable models. Accessed: 2025-01-06.

Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. 2023. Orca: Progressive learning from complex explanation traces of gpt-4. *Preprint*, arXiv:2306.02707.

Akifumi Nakamachi, Tomoyuki Kajiwara, and Yuki Arase. 2020. Text simplification with reinforcement learning using supervised rewards on grammaticality, meaning preservation, and simplicity. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 153–159.

Hemant Palivela. 2021. Optimization of paraphrase generation and identification using language models in natural language processing. *International Journal of Information Management Data Insights*, 1(2):100025.

Pouya Pezeshkpour. 2023. Measuring and modifying factual knowledge in large language models. In *2023 international conference on machine learning and applications (ICMLA)*, pages 831–838. IEEE.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21(1).

Vipul Raheja, Dhruv Kumar, Ryan Koo, and Dongyeop Kang. 2023. Coedit: Text editing by task-specific instruction tuning. arXiv preprint arXiv:2305.09857.

Timo Schick, Jane Dwivedi-Yu, Zhengbao Jiang, Fabio Petroni, Patrick Lewis, Gautier Izacard, Qingfei You, Christoforos Nalmpantis, Edouard Grave, and Sebastian Riedel. 2022. Peer: A collaborative language model. *arXiv preprint arXiv:2208.11663*.

Tianxiao Shen, Tao Lei, Regina Barzilay, and Tommi Jaakkola. 2017. Style transfer from non-parallel text by cross-alignment. *Advances in neural information processing systems*, 30.

Felix Stahlberg and Shankar Kumar. 2021. Synthetic data generation for grammatical error correction with tagged corruption models. In *Proceedings of the 16th Workshop on Innovative Use of NLP for Building Educational Applications*, pages 37–47, Online. Association for Computational Linguistics.

Renliang Sun, Wei Xu, and Xiaojun Wan. 2023. Teaching the pre-trained model to generate simple texts for text simplification. *arXiv preprint arXiv:2305.12463*.

Manan Suri, Puneet Mathur, Franck Dernoncourt, Rajiv Jain, Vlad I Morariu, Ramit Sawhney, Preslav Nakov, and Dinesh Manocha. 2024. Docedit-v2: Document structure editing via multimodal llm grounding. *arXiv preprint arXiv:2410.16472*.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. 2022. Self-instruct: Aligning language models with self-generated instructions. *arXiv preprint arXiv:2212.10560*.

Zhijie Wang, Zijie Zhou, Da Song, Yuheng Huang, Shengmai Chen, Lei Ma, and Tianyi Zhang. 2025. Towards understanding the characteristics of code generation errors made by large language models. *Preprint*, arXiv:2406.08731.

Wikipedia. n.d.a. Wikipedia: Featured articles. `https://en.wikipedia.org/wiki/Wikipedia:Featured_articles`. Accessed: 2025-02-14.

Wikipedia. n.d.b. Wikipedia: Good articles. `https://en.wikipedia.org/wiki/Wikipedia:Good_articles`. Accessed: 2025-02-14.

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. 2024. 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.

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

A Additional Implementation Details

For existing models, we strictly adhere to configurations from their original papers. To manage fixed maximum token lengths L, if the combined T_{orig} and I_{edit} exceed L, we partition T_{orig} into chunks of size $\leq L$, process each chunk independently with the same edit instruction, and concatenate the outputs to form the complete edited text. We fine-tune models using Low-Rank Adaptation (LoRA) (Hu et al., 2021) with $r = 8$, $\alpha = 32$, and a dropout rate of 0.05, employing the AdamW optimizer with a learning rate of 2×10^{-5}, training for 2 epochs, an effective batch size of 1, and 4 gradient accumulation steps.

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 Configuration: We use a LoRA rank of `r = 8` and LoRA alpha $\alpha = 32$, following the original LoRA paper (Hu et al., 2021). This combination (`r = 8`, $\alpha = 32$) and a learning rate of 2×10^{-5} are widely used in practice, including in the default settings of the HuggingFace PEFT library. It produces a scaling factor of $\alpha/r = 4$, which balances training stability and memory efficiency, enabling the model to learn meaningful updates without destabilizing training. We set `lora_dropout = 0.05`, a typical value that helps regularize LoRA updates and reduce overfitting. Training and Generation Settings are as follows:

- **Epochs:** 2 epochs, which is generally sufficient for convergence in our editing task.
- **Gradient Accumulation Steps:** 4 (necessary due to a small batch size of 1 and GPU constraints).
- **Max Chunk Tokens:** 2048.
- **Max Length:** 4096.
- **Generation Settings:** `temperature = 0.2`, `top-p = 0.95`.

The token constraints ensure no exceeding of the model's context window and maintain consistent training across models. These parameters reduce randomness while keeping the generated text relevant to the task.

Decoding and Inference: During generation, we set the temperature to 0.2 and used top-p sampling with a probability of 0.95, then merging outputs from all chunks to produce the final edited text. The temperature and top-p settings follow previous editing task studies (Cassano et al., 2024) to ensure minimal changes rather than creative expansions as our editing tasks require precise.

B Data Example

Table 6 presents representative examples from our benchmark, covering four distinct data categories—WikiText, LaTeX, Code, and Database DSL. Each example includes the original content, the user-issued edit request, the resulting edited content, the line-level difference, and the associated G-score indicating edit difficulty.

We make a concrete instance using data in the LaTeX category in Table 6. If the edit request is to "Remove the duplicate `\begin{abstract}` at the beginning of the abstract environment," the diff output might display on Line 1:

```
\begin{abstract}[-\begin{abstract}-]
```

This indicates that the duplicate has been successfully removed.

C Dataset Generation Prompts

We use the following prompts for dataset generation on each domain.

```
  user_prompt = r"'Task: Generate one precise editing request for the given LaTeX code, focusing exclusively on one detailed LaTeX-specific aspect.
1. Analyze LaTeX Components: Examine the LaTeX code thoroughly, identifying elements such as commands, environments, packages, mathematical expressions, figures, tables, references, labels, and syntax structures.
2. Target a Single LaTeX Issue: The editing request must address only one specific LaTeX-related issue such as commands, environments, packages, mathematical expressions, figures, tables, references, labels, and syntax structures.
3. Clearly define the exact edit needed. The action should be definitive and unambiguous, avoiding any form of suggestion, optional language, or choices. Do not include reasons for the edit or any additional information beyond the request.
4. Do not include reasons for the edit or any additional information beyond the edit request. The request should be a direct instruction.
The request examples are:
[Example 1]
<Edit Request>
Replace the \begin{equation} ... \end{equation} environment with a \[ ...\] display math environment to present the equation.
</Edit Request>
[Example 2]
<Edit Request>
Remove the \centering command inside the figure environment and insert \centering immediately after \begin{figure}.
</Edit Request>
[Example 3]
<Edit Request>
Change the citation command \cite{einstein} to \parencite{einstein} to display the citation in parentheses.
</Edit Request>
[Example 4]
<Edit Request>
Change the column specification in the tabular environment from {l l l} to {l c r} to adjust the alignment of the data columns.
</Edit Request>
[Example 5]
<Edit Request>
Replace the placeholder ??? in the reference text with \ref{sec:relwork} to properly reference the "Related Work" section.
</Edit Request>
[Example 6]
<Edit Request>
Rename the macro \vect to \vecbold in both its definition and throughout the document.
</Edit Request>
[Example 7]
<Edit Request>
Add the optional width argument to \includegraphics{example-image} as \includegraphics[width=0.5\textwidth]{example-image} to scale the image.
</Edit Request>
[Example 8]
<Edit Request>
Remove the \usepackage{epsfig} line and replace it with \usepackage{graphicx} to handle graphics
</Edit Request>

I will give you the content and then the editing request.
Please Edit the content based on the editing request.
While Editing, don't add other words like modified or something. Just Edit directly.

Content: {original_context}
Editing Request: {edit_request}
Please return the complete content after editing.
Don't skip the empty line and keep the original
apart from the editing part.
```

We use the following prompts for G-Eval.

```
Understanding Content Differences:
Changes between the original and edited texts are categorized and formatted as follows:
Replace (replace): [original_text -> modified_text]
Delete (delete): [-original_text-]
Insert (insert): [+modified_text+]
Equal (equal): unchanged_text

Instructions:
1) Read Carefully: Examine the original contents and the edited parts (shown as the formatted diff) thoroughly.
2) Identify and Evaluate: Using the above diff formatting rules, determine if the modifications are both correct and complete throughout the entire original content.
3) Assign a Coherence Score: Based on the Evaluation Criteria, rate the coherence of the modifications on a scale of 1 to 10, where 1 is the lowest and 10 is the highest.

Final Output: Only provide the numeric coherence score.
The given contents are: 1) the entire original content, 2) the edit request, and 3) a formatted diff that shows how the edited content differs from the original.
```

Data Category	Orignal Content	Edit Request	Edited Content	Difference	G-score
WikiText	...As with previous <unk> Chronicles games, Valkyria Chronicles III is a tactical role @-@ playing game where players take control of a military unit...	Replace "\<unk>\" with "Valkyria" where it appears in the text.	...As with previous Valkyria Chronicles games, Valkyria Chronicles III is a tactical role @-@ playing game where players take control of a military unit...	Line 2 differs: Differences: ...As with previous [<un -> Val]k[> -> yria] Chronicles games, Valkyria Chronicles III is a tactical role @-@ playing game where players take control of a mili- tary unit...	9
LaTex	\begin{abstract}\n\begin{abstract}\n %\mika{}, \guandao{}, \leo{}\n \vspace{-0.2cm}\n Neural radiance fields (NeRF) rely on volume rendering to...	Remove the duplicate \begin{abstract} at the beginning of the abstract environment.	\begin{abstract}\n %\mika{}, \guandao{},\leo{}\n \vspace{-0.2cm}\n Neural radiance fields (NeRF) rely on volume rendering to...	Line 1 differs: Differences: \begin{abstract}[- \begin{abstract}-]	9
Code	...def yield_nanopub(assertions, annotations, line_num):\n """Yield nanopub object""" if not assertions:...	Change the function definition from: def yield_nanopub(assertions, annotations, line_num) to include type annotations as: def yield_nanopub(assertions: list, annotations: dict, line_num: int) -> dict	...def yield_nanopub(assertions: list, annotations: dict, line_num: int) -> dict: """Yield nanopub object""" if not assertions:...	Line 1 differs: Differences: def yield_nanopub(assertions[+: list+], annotations[+: dict+], line_num[+: int+])[+ -> dict+]:	10
Database DSL	...CREATE TABLE DB_PRIVS\n (\n DB_GRANT_ID NUMBER NOT NULL,\n CREATE_TIME NUMBER (10) NOT NULL,\n DB_ID NUMBER NULL,\n)...	Rename the column `"CREATE_TIME"` in the `DB_PRIVS` table to `"CREATION_TIMESTAMP"`	...CREATE TABLE DB_PRIVS\n (\n DB_GRANT_ID NUMBER NOT NULL,\n CREATION_TIMESTAMP NUMBER (10) NOT NULL,\n DB_ID NUMBER NULL,\n)...	Line 4 differs: Differences: CREATE[E ->ION]_TIME[+STAMP+] NUMBER (10) NOT NULL,	9

Table 6: Data examples of different data categories with all attributes (content, edit request, edited content, difference, and G-score).