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

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have transformed natural language processing, yet they still struggle with direct text editing tasks that demand precise, context-aware modifications. While models like ChatGPT excel in text generation and analysis, their editing abilities often fall short, addressing only superficial issues rather than deeper structural or logical inconsistencies. In this work, we introduce a dual approach to enhance LLMs editing performance. First, we present InstrEditBench, a high-quality benchmark dataset comprising over 20,000 structured editing tasks spanning Wiki articles, LaTeX documents, code, and database Domain-specific Languages (DSL). InstrEditBench is generated using an innovative automated workflow that accurately identifies and evaluates targeted edits, ensuring that modifications adhere strictly to specified instructions without altering unrelated content. Second, we propose FineEdit, a specialized model trained on this curated benchmark. Experimental results demonstrate that FineEdit achieves significant improvements around 10% compared with Gemini on direct editing tasks, convincingly validating its effectiveness.

1 Introduction

Large Language Models (LLMs) have revolutionized natural language processing, unlocking capabilities once thought unattainable. ChatGPT, for example, shows exceptional skills in text generation and logical reasoning (OpenAI, 2023). Despite these impressive advancements, LLMs still face significant challenges in the underperformance of text editing tasks. Castillo-González et al. (2022) has mentioned that the use of ChatGPT for editing has obvious limitations, which might not accurately follow the editing task instructions and understand the author's intent, leading to changes that are not appropriate to the context of the text. Meanwhile, it effectively addresses surface-level issues, such as spelling and formatting, but cannot resolve complex challenges, such as editing in long text context or strict following task instructions.

To address these limitations, researchers have developed methods to enhance the editing capabilities of LLMs, particularly under task-specific scenarios, e.g., editing in code, LaTeX, etc. However, 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. This issue arises primarily because these models overly emphasize task-specific constraints and are susceptible to hallucinating extraneous information.

In contrast, we notice that if narrowing the model's focus to just two factors: the precise location of the edit and the specific content to be changed, the edit task itself could be better accomplished. Per this intuition, we propose a dual approach consisting of a benchmark for editing tasks and a model named FineEdit. For the benchmark, we design an automated workflow that focuses on accurately identifying and evaluating structured text edits. This workflow identifies precise differences and ensures correct edits through quality control. By reducing noise and focusing on meaningful modifications, this process produces a benchmark that is both practical for training and robust for evaluation. It directly addresses limitations in existing methods and aligns better with the practical demands of real-world editing tasks.

Furthermore, we use part of the curated benchmark to train the model, focusing on direct editing tasks. FineEdit achieves an over 10% improvement over Gemini 1.5 Flash and Gemini 2.0 Flash (DeepMind, 2024), 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% in direct editing tasks.

The main contributions of this work include:

- **A high-quality benchmark dataset (InstrEditBench)**[1]: We created a curated dataset with 20,000+ structured editing tasks across Wiki articles, LaTeX documents, Code, and Database DSL, providing a unified evaluation standard for structured text editing research.

- **An innovative automated dataset generation workflow**: We developed 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 introduce a specialized model designed for structured 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

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

[1]We will release all datasets and the code to promote reproducibility on acceptance.

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 (Faltings et al., 2021; Reid and Neubig, 2022; Mallinson et al., 2022; Du et al., 2022a,b; Kim et al., 2022). However, many of these approaches rely on specialized techniques or focus narrowly on specific tasks, such as grammar correction (Mallinson et al., 2022; Fang et al., 2023), text simplification (Štajner et al., 2022), paraphrase generation (Chowdhury et al., 2022), or style transfer (Reif et al., 2022), which limits their generalizability across a broader range of editing scenarios. In the realm of code editing, Fan et al. (Fan et al., 2024) examined LLMs for code change tasks and identified weaknesses in generating accurate reviews and commit messages. 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.

2.3 LLM Benchmarking

LLM benchmarking is a crucial aspect of evaluating the diverse capabilities of LLM. Researchers have developed numerous benchmarks spanning multiple domains, including code (Chen et al.,

2021; Austin et al., 2021; Jimenez et al., 2024; Yang et al., 2025), commonsense reasoning (Bisk et al., 2020; Sap et al., 2019; Zellers et al., 2019; Sakaguchi et al., 2021), reading comprehension (Rajpurkar et al., 2018; Choi et al., 2018; Clark et al., 2019), and language understanding (Wang, 2018; Wang et al., 2019; Xu et al., 2020). However, only a few works benchmark the editing performance of LLMs. For example, GEM (Xu et al., 2024) introduces metrics for subjective tasks without gold standards, while CriticBench (Lin et al., 2024) assesses iterative output refinement. Additionally, Cassano et al. (2023) explores that finetuning with curated training data significantly improves code editing performance. Some automated evaluation is also involved in LLM benchmarking. For instance, G-Eval (Liu et al., 2023) is an automated evaluation framework that leverages large language models to assess text quality and model performance in generative tasks. Built on a Chain-of-Thought (CoT) prompting strategy (Wei et al., 2022), G-Eval guides the model to articulate intermediate reasoning steps before reaching its final evaluation, leading to outputs that align closely with human judgments (Liu et al., 2023). However, these efforts focus on short-context, isolated tasks and do not systematically evaluate an LLM's ability to locate and modify content within long contexts. Our work addresses this gap by introducing a comprehensive benchmark covering Wiki, code, DSL, and LaTeX, emphasizing long-context editing.

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 (hive, 2024; b mc2, 2023; 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 A.

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 (Wang et al.,

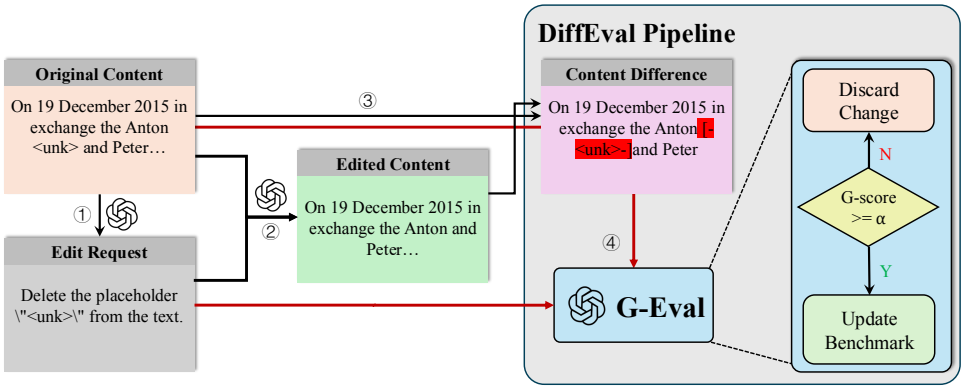

Figure 1: Workflow of Generating High-quality FineEdit benchmark. The content difference is highlighted in red.

2025; Cassano et al., 2024). 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 modifications. 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.

We make a concrete instance using data in the LaTeX category in Table 1. 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.

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 is able to 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 Data Statistics

Our curated benchmark comprises 28,050 items spanning a diverse array of structured data types, including 8,366 LaTeX contexts, 7,712 code segments, 8,025 WikiText entries, and 3,947 database language samples, thereby reflecting both the generality and scale of real-world structured data. Table 1 shows the example across four categories. For each item, it has the following attributes:

- `Id:` a unique identifier for each entry.
- `Original content:` the content directly extracted from the data source.
- `Edit request:` The editing instruction generated through zero-shot or few-shot prompting based on the original content
- `Edited Content:` the output after applying edit request to the original content.
- `Difference:` the changed part between Edit content and original content.
- `G-score:` evaluates the quality of the edited content based on its strict adherence to the edit request content.

4 Evaluation

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 using a 90/10 train-test split. Additionally, we introduce three versions of FineEdit—FineEdit-L, FineEdit-XL, and FineEdit-Pro—fine-tuned from LLaMA-3.2-1B, LLaMA-3.2-3B, and Qwen2.5-3B-Instruct base models, respectively, to cover a wide spectrum of 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 (Nakamachi et al., 2020; Shen et al., 2017), we use BLEU and ROUGE-L metrics to assess the vocabulary and structural consistency between the edited and reference texts.

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. During generation, we set the temperature to 0.7 and use top-p sampling with $p = 0.9$, merging outputs from all chunks to produce the final edited text. Additional hyperparameter configurations and training details are provided in Appendix B.

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 2.

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 often a crucial factor, Gemini 2.0 Flash did not surpass Gemini 1.5 Flash in overall effectiveness. For instance, despite having more parameters than LLaMA-3.2-1B, Mistral-7B-OpenOrca underperformed in both metrics, highlighting the significance of model architecture and training methods. Moreover, while Gemini 2.0 Flash shows superior semantic understanding in the Wiki category—achieving a BLEU score of 0.9133 and a ROUGE-L score of 0.9429—its overall performance remains below that of its counterpart.

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 Gem-

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

ini 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 performance notably improved in some categories—for example, in the LaTeX domain, where Gemini 2.0 Flash exhibited a 20% higher BLEU score than in the zero-shot setting—the overall few-shot results still lag behind FineEdit. In certain cases, such as the SQL category, few-shot learning made little difference, with BLEU and ROUGE-L scores of only 0.1600 and 0.1814, respectively. These findings reinforce the value of our curated benchmark in driving improvements in editing tasks.

Key Findings: FindEdit demonstrates robust overall effectiveness across Wiki, Code, DSL, and LaTeX categories. These results not only position FineEdit as a competitive method for structured editing tasks but also provide valuable insights into how targeted training strategies can elevate model performance in diverse application scenarios.

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 effectiveness 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) boosts Gemini's performance in some cases—such as a 8% higher BLEU score in Wiki dataset for Gemini 2.0 Flash—the overall results still lag behind

Method	Model	Open-Source	LaTeX		DSL		Wiki		Code		Overall	
			BLEU	ROUGE-L	BLEU	ROUGE-L	BLEU	ROUGE-L	BLEU	ROUGE-L	BLEU	ROUGE-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-Instruct	✓	0.5467	0.6712	0.4107	0.4991	0.4170	0.5699	0.3967	0.5390	0.4492	0.5816
	Mistral-7B-OpenOrca	✓	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 2: 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.

FineEdit-Pro. This evidence confirms that our tailored supervised fine-tuning approach yields a more robust and generalizable solution for structured editing tasks.

Key Findings: FineEdit's supervised fine-tuning markedly enhances performance. FineEdit-L surpasses zero-shot baselines and FineEdit-XL offers comparable gains, while FineEdit-Pro (built on Qwen2.5-3B-Instruct) achieves the highest scores. This highlights that robust instruction tuning is more effective than merely scaling model size.

4.4 Qualitative Study

To qualitatively assess the performance of FineEdit, we conduct several studies as shown in Figure 2. This figure illustrates eight examples of how FineEdit-Pro and Gemini respond to diverse editing requests. In several cases, FineEdit-Pro accurately applies changes—such as adding new columns in DSL or adjusting environment commands—while Gemini often restates the instruction without 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 Figure 2, 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 Figure 2, 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.

Key Findings: FineEdit-Pro demonstrates superior handling of DSL and Code edits compared to Gemini, though minor issues such as duplicate outputs in LaTeX tasks remain. Overall, FineEdit's qualitative performance confirms its robust ability to interpret and execute complex editing instructions.

4.5 Human Evaluation

To assess whether `DiffEval` enhances overall dataset quality, we conducted a human evaluation. Given that our dataset includes Code and DSL categories—areas closely tied to computer programming—we have three evaluators, each holding at least a Bachelor's degree in a Computer Science-related field. We established the following guidelines to ensure rigorous assessment: (1) Precise

Figure 2: Comparison between Gemini and FindEdit Pro response.

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

Table 3: Sample performance based on the G-Score

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 3. 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.

Key Findings: The DiffEval pipeline significantly improves dataset quality, with Wiki and Code categories achieving high precision, and the DSL category showing over 24% enhancement in quality.

5 Conclusion

In this work, we address the critical gap in LLMs' ability to perform precise and targeted text modifications. We introduce InstrEditBench, a high-quality benchmark with 20,000+ structured editing tasks across Wiki articles, LaTeX documents, code, and database DSLs, enabling rigorous evaluation of direct editing capabilities. To further advance LLMs' editing proficiency, we propose FineEdit, a specialized model trained on this benchmark. Extensive evaluations demonstrate that FineEdit outperforms state-of-the-art models, including GPT-4o, Gemini 2.0, and LLaMa-3.2, with up to 8% improvement compared to Gemini in direct editing task performance.

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 context, 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.

References

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

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

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

Yonatan Bisk, Rowan Zellers, Jianfeng Gao, Yejin Choi, et al. 2020. Piqa: Reasoning about physical commonsense in natural language. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 7432–7439.

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.

Federico Cassano, Luisa Li, Akul Sethi, Noah Shinn, Abby Brennan-Jones, Jacob Ginesin, Edward Berman, George Chakhnashvili, Anton Lozhkov, Carolyn Jane Anderson, et al. 2023. Can it edit? evaluating the ability of large language models to follow code editing instructions.

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.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Eunsol Choi, He He, Mohit Iyyer, Mark Yatskar, Wentau Yih, Yejin Choi, Percy Liang, and Luke Zettlemoyer. 2018. Quac: Question answering in context. *arXiv preprint arXiv:1808.07036*.

A. Chowdhury et al. 2022. Enhanced paraphrase generation via t5 fine-tuning. In *Findings of ACL 2022*.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. 2019. Boolq: Exploring the surprising difficulty of natural yes/no questions. *arXiv preprint arXiv:1905.10044*.

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

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*.

X. Du et al. 2022a. Grit1: A grammar error correction dataset for llm evaluation. In *Proceedings of EMNLP 2022*.

X. Du et al. 2022b. Grit2: Extending grammar error correction for multilingual llms. In *Findings of EMNLP 2022*.

Isabelle Faltings et al. 2021. Leveraging fine-tuned t5 for knowledge-based text editing tasks. In *Proceedings of the ACL 2021*.

L. Fan, J. Liu, Z. Liu, D. Lo, X. Xia, and S. Li. 2024. Exploring the capabilities of llms for code change related tasks. arXiv preprint arXiv:2407.02824.

J. Fang et al. 2023. Hierarchical editing for grammar correction tasks. In *Proceedings of COLING 2023*.

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*.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. 2024. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*.

J. Kim et al. 2022. Towards general-purpose text editing with t5. In *Findings of ACL 2022*.

Latex2Poster. 2024. Latex2poster dataset. Hugging Face.

Lerocha. 2024. Chinook database. GitHub Repository.

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.

Zicheng Lin et al. 2024. Criticbench: Benchmarking llms for critique-correct reasoning. In *arXiv preprint arXiv:2402.14809*.

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.

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.

J. Mallinson et al. 2022. Edit5: Fine-tuning t5 for multi-domain editing tasks. In *Findings of ACL 2022*.

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

Meta AI. 2024. Llama 3.2: Advancing vision and edge ai for mobile devices. 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.

OpenAI. 2023. Chatgpt: Optimizing language models for dialogue. `https://openai.com/blog/chatgpt`. Accessed: 2025-01-02.

Colin Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67.

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

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. *arXiv preprint arXiv:1806.03822*.

S. Reid and G. Neubig. 2022. Learning to edit text with transformers. In *Proceedings of NAACL 2022*.

M. Reif et al. 2022. Style transfer in text editing with transformers. In *Findings of ACL 2022*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *arXiv preprint arXiv:1904.09728*.

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.

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.

Alex Wang. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2019. Superglue: A stickier benchmark for general-purpose language understanding systems. *Advances in neural information processing systems*, 32.

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.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*.

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.

Liang Xu, Hai Hu, Xuanwei Zhang, Lu Li, Chenjie Cao, Yudong Li, Yechen Xu, Kai Sun, Dian Yu, Cong Yu, et al. 2020. Clue: A chinese language understanding evaluation benchmark. *arXiv preprint arXiv:2004.05986*.

Shengwei Xu et al. 2024. Benchmarking llms' judgments with no gold standard. In *arXiv preprint arXiv:2411.07127*.

John Yang, Carlos E. Jimenez, Alex L. Zhang, Kilian Lieret, Joyce Yang, Xindi Wu, Ori Press, Niklas Muennighoff, Gabriel Synnaeve, Karthik R. Narasimhan, Diyi Yang, Sida I. Wang, and Ofir Press. 2025. SWE-bench multimodal: Do ai systems generalize to visual software domains? In *The Thirteenth International Conference on Learning Representations*.

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.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

S. Štajner et al. 2022. Simpleedit: A toolkit for text simplification with llms. In *Findings of ACL 2022*.

A 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.

```
 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.
```

B Additional Implementation Details

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., 2021) 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.