ChartEditor: A Reinforcement Learning Framework for Robust Chart Editing

Liangyu Chen[1*], Yichen Xu[1*], Jianzhe Ma[1], Yuqi Liu[2], Donglu Yang[1], Liang Zhang[†]
Zihao Yue[1], Wenxuan Wang[1‡], Qin Jin[1‡]

[1]Renmin University of China, [2]Chinese University of Hong Kong
wenxuanwang@ruc.edu.cn, qjin@ruc.edu.cn

Abstract

Chart editing reduces manual effort in visualization design. Typical benchmarks limited in data diversity and assume access to complete chart code, which is seldom in real-world scenarios. To address this gap, we present ChartEditVista, a comprehensive benchmark consisting of 7,964 samples spanning 31 chart categories. It encompasses diverse editing instructions and covers nearly all editable chart elements. The inputs in ChartEditVista include only the original chart image and natural language editing instructions, without the original chart codes. ChartEditVista is generated through a fully automated pipeline that produces, edits, and verifies charts, ensuring high-quality chart editing data. Besides, we introduce two novel fine-grained, rule-based evaluation metrics: the layout metric, which evaluates the position, size and color of graphical components; and the text metric, which jointly assesses textual content and font styling. Building on top of ChartEditVista, we present ChartEditor, a model trained using a reinforcement learning framework that incorporates a novel rendering reward to simultaneously enforce code executability and visual fidelity. Through extensive experiments and human evaluations, we demonstrate that ChartEditVista provides a robust evaluation, while ChartEditor consistently outperforms models with similar-scale and larger-scale on chart editing tasks.

Introduction

Chart editing aims to utilize multi-modal large language models (MLLMs) to generate chart code to make high-quality visualizations reflect the customize requirements. This task is particularly practical as it significantly reduces the manual effort required for chart design and implementation.

However, existing benchmarks (Yan et al. 2024a; Zhao et al. 2025a; Yang et al. 2024) either assume complete chart code access or are limited in instruction diversity, editable elements, and chart types, with a lack of large-scale training data.

To address these limitations, we introduce ChartEditVista, a comprehensive benchmark for chart editing. ChartEditVista consists of 7,964 carefully curated samples with original chart images, natural language editing instructions, and corresponding modified code. It covers 31 chart types, 6 editing tasks and nearly all editable components. The benchmark is created through an automated pipeline combining scalable data generation and rigorous quality control, with multi-dimensional automated evaluation and expert manual refinement to ensure robust quality.

Our analysis also reveals limitations in existing metrics for chart editing tasks. MLLM-based metrics (Zhao et al. 2025a; Yang et al. 2024) often exhibit hallucinations, while rule-based metrics (Yang et al. 2024) are inadequate for fine-grained modifications. To bridge this gap, we propose two novel fine-grained rule-based metrics: layout metric and text metric, which assess graphical elements and textual content respectively. Comprehensive human evaluations show strong alignment with human judgment.

Furthermore, our evaluation on ChartEditVista reveals that contemporary open-sourced MLLMs (Wang et al. 2024; Chen et al. 2024; Li et al. 2024), including chart specific models (Zhang et al. 2024; Han et al. 2023; Xu et al. 2024; Xia et al. 2024), show limited performance on chart editing tasks. To overcome these challenges, we present ChartEditor, a model specifically optimized for chart editing tasks. Built upon the Qwen2.5-VL-3B (Bai et al. 2025), our model incorporates several key innovations. First, we introduce a novel rendering reward mechanism that directly assesses chart output fidelity, which serves as the optimization objective for Group Relative Policy Optimization (GRPO) (Guo et al. 2025; Shao et al. 2024). Second, we implement a two-phase training framework combining supervised fine-tuning and curriculum reinforcement learning to stable reward signals. Third, we validate the generalizability of our rendering reward by successfully applying it to the Chart-to-Code task, where it yields consistent performance improvements.

Extensive experiments demonstrate that our ChartEditor-3B achieves state-of-the-art (SoTA) performance across MLLMs with similar scale on ChartEditVista while maintaining strong generalization on out-of-domain benchmarks including ChartEdit (Zhao et al. 2025a) and ChartMimic (Yang et al. 2024). Notably, it consistently outperforms a range of chart-specialized models and even surpasses several larger models with 26B and 34B parameters, highlighting the effectiveness of our model and training

[*]These authors contributed equally.
[†]Independent Researcher.
[‡]Qin Jin and Wenxuan Wang are the corresponding authors.

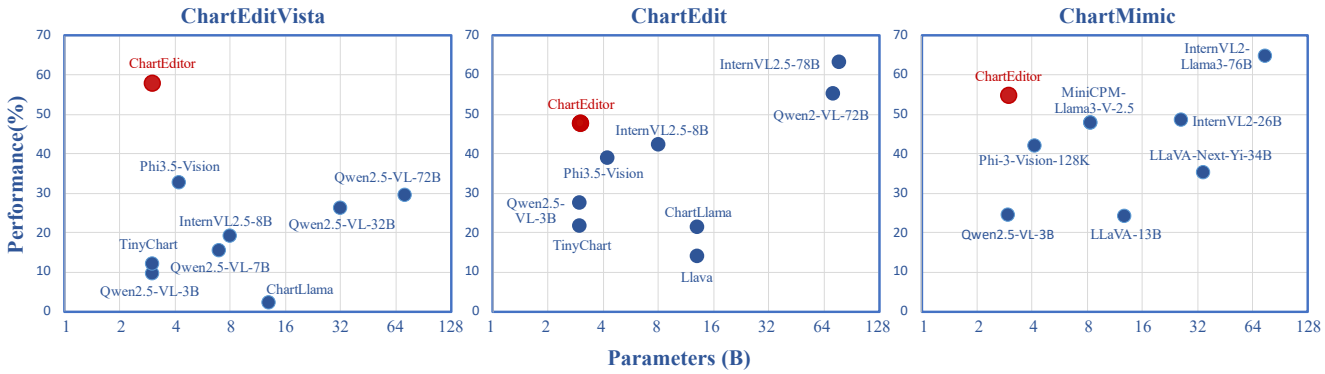

Figure 1: Comparisons on ChartEditor and other SOTA models, red dots represents ChartEditor and bule represent others.

framework.

Our contributions are summarized as follows:

1. We introduce *ChartEditVista*, a chart editing benchmark containing 7,964 manually verified samples spanning 31 chart types, covering diverse editable components and editing tasks. Our robust automated chart editing data generation pipeline, complemented by manual verification, ensures data quality.
2. We propose the *layout* and *text metric*, which outperform existing evaluation methods in reliability and fine-grained change detection. Human evaluations confirm their robustness and strong correlation with human assessments.
3. We present *ChartEditor*, a model trained using GRPO and a novel *rendering reward*. It achieves state-of-the-art performance on multiple benchmarks, including ChartEditVista, ChartEdit, and ChartMimic.

Related Works

Chart Related Works. Charts, as a specific type of image used for data visualization, have garnered increasing research attention (Huang et al. 2024). Existing works explore chart understanding from the perspectives of code (Yang et al. 2024; Zhao et al. 2025a; Xia et al. 2024; Zhang, Ma, and Vosoughi 2024) and content (Huang et al. 2024; Masry et al. 2022; Xia et al. 2024; Chen et al. 2025; Zhang et al. 2023; Dong et al. 2024a,b; Laurençon et al. 2024; Ye et al. 2023a,b, 2024). In the specific field, Chart Editing, significant progress has been made. For instance, some studies (Yan et al. 2024a) have proposed workflow-based approaches for Chart Editing, while others (Yang et al. 2024; Zhao et al. 2025a) have introduced multi-modal guidance to facilitate end-to-end chart modification. Despite these advancements, existing benchmarks still face limitations in terms of the diversity of instruction types, the variety of editable elements, and the range of chart types supported. Furthermore, Chart-specific Model (Zhang et al. 2024; Han et al. 2023; Xu et al. 2024; Zhao et al. 2025b) lack the ability of high-quality chart editing.

Multimodal Large Language Models. Recent research in Multimodal Large Language Models (MLLMs) has made substantial strides (Liu et al. 2024, 2023a; Lin et al. 2023). For example, closed-source MLLMs such as GPT-series (Hurst et al. 2024; Achiam et al. 2023), Claude-series (Anthropic 2024), and Gemini-series (Team et al. 2023; Comanici et al. 2025) are capable of handling complex real-world vision and language tasks. Open-source MLLMs like LLaVA (Li et al. 2024; Liu et al. 2023b), Qwen-VL (Bai et al. 2025; Wang et al. 2024), and InternVL-VL (Chen et al. 2024) also demonstrate strong performance compared to previous methods across several standard multimodal understanding benchmarks. However, these models still underperform when applied to tasks with high practical demands, such as Chart Editing.

ChartEditVista

Task Definition

Chart editing involves generating a modified chart based on a given chart image and accompanying textual editing instructions. This task is commonly approached by predicting the underlying chart code that, once rendered, produces the desired edited chart. However, existing benchmarks often assume access to the original chart code, which is an unrealistic assumption in many real-world applications. To better align with practice, we formulate the task as follows:

$$C' = f(I, T), \tag{1}$$

where I denotes the input chart image, T denotes the editing instructions, f is the model, and C' is the code used to generate modified chart.

Data Construction Pipeline

To construct a diverse and semantically rich set on chart editing task, we adopt a multi-stage generation and filtering pipeline, as shown on Figure 2.

Data Collection and Original Chart Generation. We begin by creating a chart topic pool and a chart type pool, aiming to ensure content diversity. Chart topics are generated using GPT-4.1, yielding hundreds of unique topics. We

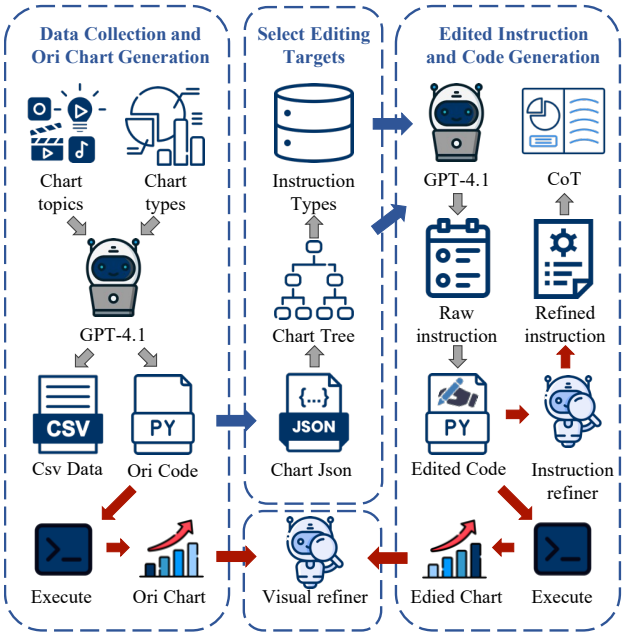

Figure 2: Overview of our fully data construction pipeline, it can generate chart editing data and refine them in various aspects. The gray arrows show the data flow inside each stage, the dark blue arrows show the data flow between stages and the red arrows show the data quality control steps.

manually predefine 31 chart types to ensure a broad range of visual formats.

Given a sampled chart topic and type, we employ GPT-4.1, selected for its strong code generation and visual reasoning capabilities (OpenAI 2025), to generate synthetic data in CSV format and the corresponding Python plotting code, following the approach of ChartLlama (Han et al. 2023). To further enhance layout variability, we manually curated an attribute bank encompassing visual features such as the presence of multiple subplots, color schemes, axis configurations, and other stylistic properties. For each chart, we randomly sample a subset of attributes and incorporate them into the generation prompt to further enhance diversity.

All generated code is executed for initial quality control, with non-runnable scripts discarded. To further ensure visual clarity, each rendered chart is assessed by GPT-4.1 for interpretability and rendering quality (e.g., avoiding overlaps or truncation). Charts that fail this check are excluded, ensuring a pool of high-quality, visually coherent references.

Selecting Editing Targets. To generate diverse and fine-grained chart editing instructions, it is crucial to first identify almost all potential editable elements within a chart. To achieve this, we utilize GPT-4.1 to convert the generated Python plotting code into a structured JSON format, referred to as the Chart JSON, which explicitly encodes all visual and textual elements present in the chart, along with their associated attribute values.

We then transform the Chart JSON into a hierarchical Chart Tree, where the root node represents the entire chart, internal nodes correspond to major chart components , and leaf nodes typically denote atomic elements. This tree structure enables systematic traversal for fine-grained editing and supports diverse instructions on both visuals and data.

For modify and delete operations, we randomly sample nodes from the Chart Tree as editing targets, ensuring broad coverage of editable entities. For add operations, we provide the entire Chart JSON as context to the model, enabling it to reason about where and how to insert new elements in a coherent manner.

Editing Instructions and Code Generation. Given the Chart JSON and its corresponding Chart Tree, we instruct GPT-4.1 to perform two key tasks: (i) generate an initial editing instruction, and (ii) produce the corresponding edited chart code. If the sampled node is intrinsically non-editable, GPT-4.1 is instructed to skip that node and refrain from generating an instruction.

We apply a rigorous quality control protocol for the generated modify instructions and edited chart codes. As with reference charts, any edited code that fails to render a valid chart is discarded. Moreover, we prompt GPT-4.1 to refine the instruction based on the original and edited code, ensuring all the final instructions accurately and naturally describes the performed change. Finally, we submit the reference image, the refined instruction, and the edited image to GPT-4.1 for a holistic quality assessment. This check ensures that the editing outcome is not only syntactically correct but also interpretable and faithful to the modification.

To enhance the alignment between visual semantics and chart code, we generate a visual-code Chain-of-Thought (CoT) from the final code and its corresponding refined instruction, inspired by ChartCoder (Zhao et al. 2025b). Each CoT consists of three components: (1) identification of the specific chart component(s) targeted by the instruction; (2) specification of the attribute-level modifications required in the code; and (3) description of the expected visual change resulting from the edited code.

Name	Chart Types	Data number	Edit type	Editable Objects
ChartCraft (Yan et al. 2024b)	5	5.5k	4	Limited
ChartX (Xia et al. 2024)	18	6k	0	None
ChartMimic (Yang et al. 2024)	22	2.4k	1	Limited
ChartEdit (Zhao et al. 2025a)	19	1.4k	6	Limited
ChartEditVista (ours)	31	7.9k	6	Unlimited

Table 1: Comparison of existing chart-related benchmarks.

Benchmark Statistics and Analysis

Data Statistics. Building on the aforementioned automatic pipeline, we construct ChartEditVista, a comprehensive benchmark comprising *601* unique base charts and a total of *7,964* triplets in the form of *(chart, instruction, modified code)*.

Diversity Analysis. As shown in Table 1 and Appendix A, ChartEditVista covers *31* chart types. It spans *six* major instruction categories—*add visual components*, *modify visual components*, *delete visual components*, *add data*,*modify*

Benchmark	Text		Layout	
	r	ρ	r	ρ
ChartEditVista	0.779 ($1.4e^{-21}$)	0.860 ($2.5e^{-30}$)	0.712 ($2.1e^{-16}$)	0.718 ($4.0e^{-17}$)
ChartMimic	0.775 ($2.6e^{-21}$)	0.790 ($1.6e^{-22}$)	0.849 ($5.4e^{-29}$)	0.847 ($1.3e^{-28}$)
ChartEdit	0.778 ($1.8e^{-21}$)	0.739 ($1.4e^{-18}$)	0.783 ($5.4e^{-22}$)	0.800 ($1.8e^{-23}$)

Table 2: The correlations observed on three chart editing benchmarks indicate that our metrics exhibit strong and statistically significant agreement with human evaluation.

data and *delete data*. It maintains a balanced distribution of single- (53.4%) and multi-subplot (46.6%) charts, and exhibits substantial variation in instruction complexity, with single-instruction cases accounting for 35.8% of samples and multi-instruction cases ranging up to six steps. Furthermore, the editing granularity ranges from fine-grained modifications of individual component attributes to global edits affecting the entire chart, supported by the hierarchical Chart Tree representation.

Quality Analysis. To ensure data quality, we engaged 10 trained annotators to manually validate all 7,964 triplets. The overall instruction clarity exceeded 97%, and the success rate of visually implementing modification instructions in the charts was over 94%. For the visual quality assessment of the original charts, annotators rated image clarity, visual occlusion, and subplot layout on a three-point scale. After normalization, the average scores for these criteria exceeded 96%, 91%, and 98%, respectively, more details are in Appendix A.

Fine-grained Evaluation Metrics

As shown in Figure 3 and Appendix A, MLLM-based metrics often hallucinate, while existing rule-based metrics are limited to coarse, subplot-level evaluation and cannot capture fine-grained edits. To address these shortcomings, we propose Rendering-Aware Rule-based Metrics (RARM), which assess visual similarity based on color, position, and style attributes of chart components. RARM comprises two complementary metrics: a layout metric for general graphical objects and a text metric for textual elements.

For graphical objects, similarity is defined as:

$$S_L(p, g) = S_{\text{color}}(p, g) \times S_{\text{pos}}(p, g) \times S_{\text{shape}}(p, g), \quad (2)$$

where color similarity is assessed via normalized Euclidean distance in RGB space and positional/shape similarity follow type-specific definitions (e.g., center distance and aspect ratio for patch-like objects).

For text elements, $S_{\text{base}}(p, g)$ is defined as 1 if the textual content matches exactly, and 0 otherwise. The final text similarity is computed as

$$S_T(p, g) = S_{\text{base}}(p, g) \cdot (1 - \lambda M_f - \alpha M_s), \quad (3)$$

where M_f and M_s indicate font family and size mismatches, respectively, and λ, α are penalty coefficients (empirically set to 0.3).

Finally, we compute overall layout and text metrics using optimal assignment via the Hungarian algorithm, enabling robust and fine-grained evaluation of chart edits.

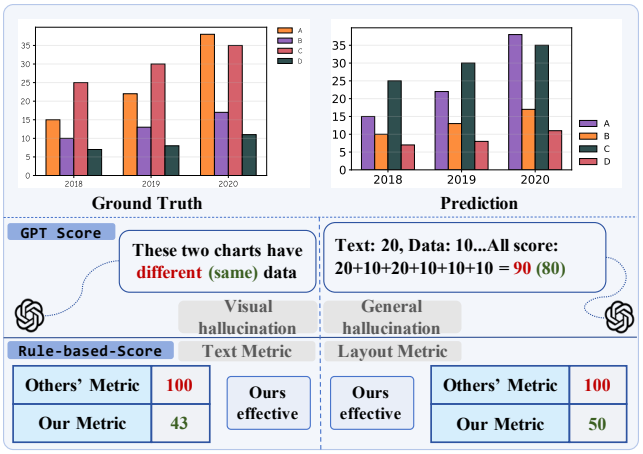

Figure 3: Limitations of existing metrics, while MLLM-Based metric suffers from serious hallucination, rule-based metric can not capture the fine-grained visual changes.

To assess reliability, we evaluated our metrics against human judgments on three chart editing benchmarks. As shown in Table 2, both text and layout metrics achieve Pearson and Spearman correlations exceeding 0.7 ($p \ll 0.01$) across all datasets, confirming strong and statistically significant alignment with human evaluation.

Method

We begin by introducing the GRPO algorithm. Subsequently, we detail our cold-start strategy, which systematically enhances the base model's chart editing capabilities. Finally, we present our design of multiple reward functions and the curriculum reinforcement learning procedure. More details are available in Appendix B.

Preliminary

Group Relative Policy Optimization (GRPO) is a reinforcement learning (RL) algorithm designed to optimize language models efficiently without requiring an explicit value model. GRPO modifies the standard Proximal Policy Optimization (PPO) objective by computing token-level advantages based on group-normalized rewards.

During reinforcement learning,

the policy model $\pi_\theta(\cdot \mid \cdot)$ generates a batch of N rollouts $\{o_i\}_{i=1}^{G}$, and optimize towards the rollout that achieve higher scores.

Multi-Stage Cold-Start

Directly applying MLLMs to RL framework is suboptimal due to two key limitations in chart editing tasks. First, MLLMs struggle with aligning visual chart content with their corresponding plotting code, making it difficult to accurately map visual elements to code representations. Second, they exhibit weak associations between natural language editing instructions and the necessary code modifications, limiting their ability to precisely translate instructions into code changes. These misalignments lead to sparse

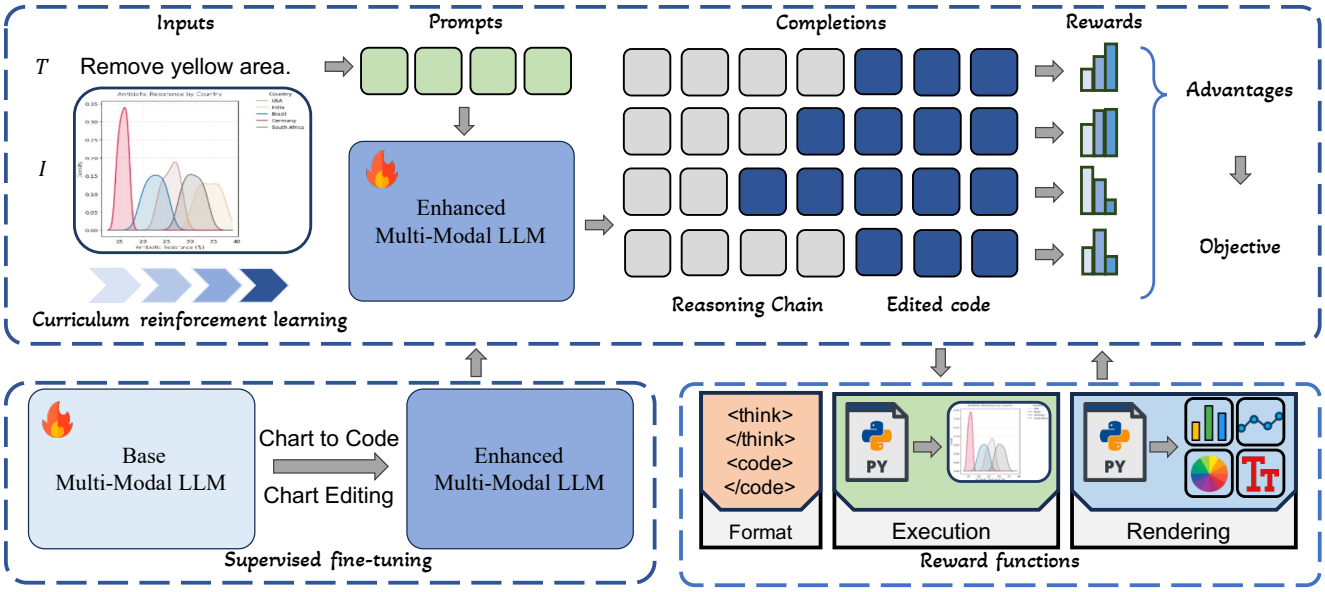

Figure 4: Overview of ChartEditor's training procedure. ChartEditor is optimized within a GRPO-based reinforcement learning framework, guided by format, execution, and rendering rewards. To further stabilize learning signals, ChartEditor employs a multi-stage training schema that incorporates supervised fine-tuning for cold-start and curriculum RL strategies.

and noisy reward signals during RL training, reducing both efficiency and downstream performance. To mitigate these issues, we propose a systematic two-stage supervised fine-tuning (SFT) pipeline using the Qwen-2.5-VL 3B model to improve cross-modal alignment.

Stage 1: Chart-to-Code SFT. The objective of this stage is to improve the correspondence between chart visual features and plotting code. The model is fine-tuned on a large-scale Chart-to-Code corpus, with the introduction of visual-to-code chain-of-thought (CoT) explanations to enable explicit reasoning over the mapping between image elements and code statements.

Stage 2: Chart Editing SFT. The goal of this stage is to strengthen the alignment between editing instructions and code modifications. The model undergoes further fine-tuning with chart editing data, incorporating editing-specific CoT demonstrations that decompose instructions, specify required code edits, and describe the expected visual outcomes, thereby reinforcing the model's capacity for instruction interpretation and code adaptation.

This two-stage SFT process facilitates integration chart visual representations, natural language editing instructions, and modifiable code components. By establishing these strengthened cross-modal alignments, the approach provides a robust foundation for subsequent reinforcement learning, ultimately leading to enhanced chart editing performance.

Reward Functions

Format Reward. We prompt the MLLM to place its reasoning trace inside `<think> ... </think>` tags and the final code inside `<code> ... </code>` tags. During training, we parse every rollout and attempt to extract both segments; if the reasoning trace and the code snippet are successfully recovered, the *Format Reward* is set to 1, otherwise 0.

Execution Reward. For every rollout produced by the MLLM, we run the extracted final code inside an isolated sandbox. Each sandbox is independent, ensuring that different executions cannot interfere with one another. If the code terminates successfully within a predefined time limit, the *Execution Reward* R_{E} is assigned a value of 1; if it times out or raises an error, the code is deemed non-executable and the reward is set to 0.

Rendering Reward. To faithfully capture the correspondence between code and visual output, we introduce a rule-based *Rendering Reward*. For each object type, we compute a pairwise similarity matrix $S \in \mathbb{R}^{M\times N}$ between M predicted and N ground truth objects:

$$S_{j,k} = \mathrm{sim}(obj_j^{\mathrm{pred}}, obj_k^{\mathrm{gt}}) \tag{4}$$

where $\mathrm{sim}(\cdot,\cdot)$ is a type-specific similarity metric (e.g. IoU for patch-like objects, L_2 distance for point objects). We then solve an optimal assignment problem using the Hungarian algorithm and normalize by the maximum object count:

$$S_{\mathrm{type}} = \frac{1}{\max(M,N)} \sum_{(j,k)\in\mathcal{M}^*} S_{j,k} \tag{5}$$

where $\mathcal{M}^*$ denotes the optimal matching. The final rendering reward aggregates the normalized scores across all object types, weighted as appropriate, and multiplied by the executability indicator R_{E}:

$$\mathcal{R}_{\mathrm{render}} = R_{\mathrm{E}} \sum_{t\in\mathcal{T}} w_t \, S_{\mathrm{type}=t} \tag{6}$$

where $R_{\mathrm{E}} \in 0, 1$ denotes code executability, more details are in Appendix C.

Model	# Params.	ChartEditVista						ChartEdit *w/o* Code			ChartMimic Customized			
		Exec.	$\mathrm{T_R}$	$\mathrm{L_R}$	Col.	Type	Avg.	Exec.	Code	Chart	Exec.	Low	High	Overall
Proprietary Models														
GPT-4o (Hurst et al. 2024)	–	69.3	31.5	48.9	52.3	41.2	43.5	91.5	60.0	79.9	**96.5**	**82.1**	**84.3**	**83.2**
Claude-3.7-Sonnet (Anthropic 2025)	–	**94.0**	46.2	55.9	64.1	75.8	60.5	**96.3**	73.4	87.5	80.6	67.4	78.1	72.3
Gemini-2.5-Pro (Comanici et al. 2025)	–	90.5	**55.1**	**63.4**	**69.0**	**76.3**	**66.0**	95.3	**82.9**	**89.2**	91.7	81.3	83.5	82.4
Open-Source Models														
Qwen2.5-VL-72B (Bai et al. 2025)	72B	80.6	11.3	39.4	38.5	30.0	29.8	89.8	57.7	71.0	85.3	67.6	69.1	68.4
Qwen2.5-VL-32B (Bai et al. 2025)	32B	74.7	9.1	35.6	33.9	26.6	26.3	88.4	61.7	72.5	85.6	66.4	69.5	68.0
Qwen3-VL-32B (QwenLM 2025)	32B	60.6	28.5	47.5	41.4	45.7	40.8	84.9	69.0	73.4	88.2	72.4	61.3	66.9
LLaVA-1.5-13B (Liu et al. 2023b)	13B	–	–	–	–	–	–	48.8	11.4	16.7	49.0	23.6	24.7	24.2
ChartLlama (Han et al. 2023)	13B	19.3	2.1	7.3	4.7	3.7	4.5	52.3	15.8	27.4	–	–	–	–
Qwen3-VL-8B (QwenLM 2025)	8B	61.9	26.4	46.3	37.7	45.4	39.0	84.9	59.6	69.3	78.1	61.4	60.8	61.1
Qwen2.5-VL-7B (Bai et al. 2025)	7B	57.1	6.1	14.4	18.9	22.3	15.4	65.1	43.4	44.0	68.4	46.0	70.0	58.0
Qwen3-VL-4B (QwenLM 2025)	4B	60.0	19.2	39.0	30.0	34.0	30.6	**79.5**	**55.6**	**63.2**	**66.0**	47.6	49.1	48.4
Qwen2.5-VL-3B (Bai et al. 2025)	3B	45.8	6.2	10.3	9.5	11.8	9.5	44.1	31.1	24.3	48.8	26.2	22.9	24.6
TinyChart (Zhang et al. 2024)	3B	30.1	6.2	13.8	13.8	13.0	12.2	36.3	18.3	25.1	–	–	–	–
Qwen3-VL-2B (QwenLM 2025)	2B	47.6	16.6	29.7	25.3	28.9	25.1	67.5	36.7	44.4	56.8	37.8	37.3	37.6
ChartEditor (**Ours**)	3B	**76.8**	**52.9**	**48.7**	**64.1**	**67.5**	**58.1**	66.5	40.2	55.3	61.6	**52.1**	**57.9**	**55.0**

Table 3: Performance on ChartEditVista (**in-domain**) and ChartEdit w/o code and ChartMimic Customized (**out-of-domain**).

Training

Curriculum Reinforcement Learning. In the initial training phase, the model rarely produces executable code, resulting in highly sparse reward signals. To facilitate the training, we implement a progressive curriculum reinforcement learning strategy: initial training on single-subplot, single-instruction cases; followed by gradual introduction of multi-subplot and multi-instruction examples. This approach maintains stable reward density throughout training.

Reward Computation. For each rollout, we compute two sub-rewards: a format reward, and a rendering reward depends on execution reward, reflecting code structure, executability, and visual fidelity, respectively. The final reward is defined as the summation of these sub-rewards:

$$r_i = r_i^{\text{format}} + r_i^{\text{render}} \tag{7}$$

Policy Update. The rewards $\{r_i\}_{i=1}^{G}$ are normalized into advantages via Z-score normalization, Finally, the model is optimized by minimizing the objective:

$$L = -\frac{1}{G}\sum_{i=1}^{G}\frac{\pi(o_i \mid I,T)}{\pi_{\text{old}}(o_i \mid I,T)}\cdot \hat{A}_{i,t} \tag{8}$$

where $\pi_{\text{old}}(\cdot \mid \cdot)$ denotes the old policy and $\hat{A}_{i,t}$ is the advantage associated with prediction o_i.

Experiments

Experimental Settings

Datasets. After filtering out examples containing stochastic code statements such as `random.choice`,we select 140k high-quality samples from the ChartCoder-160k (Zhao et al. 2025b) corpus for stage-1 supervised fine-tuning (SFT). We then perform stage-2 SFT with 20k instances constructed by our chart editing data generation pipeline, and further train the model with Generalized Reward Policy Optimization (GRPO) (Shao et al. 2024) on 6k additional ChartEditVista samples.We report results on two benchmark suites: (i) the *in-domain* ChartEditVista, and (ii) the *out-of-domain*, we select ChartEdit (Zhao et al. 2025a) and ChartMimic-Customized (Yang et al. 2024) test sets because they are designed with the absence of code input in mind.

Implementation Details. We adopt Qwen-2.5-VL-3B (Bai et al. 2025) as the default backbone. All SFT is conducted with a batch size of 16, an initial learning rate of 1e-5, and a weight decay of 0.01. For GRPO, we use a batch size of 2 and generate 8 rollout samples per training step. The execution-timeout threshold in the reward function is set to 30 s. Optimizer hyper-parameters remain identical to SFT (learning rate 1e-5, weight decay 0.01). We fully fine-tune all parameters of the MLLM throughout training.

Evaluation Metrics. We evaluate ChartEditVista using our proposed RARM layout $\mathrm{L_R}$ and text $\mathrm{T_R}$ metrics to provide fine-grained assessment. For overall chart type and color evaluation, we follow the same evaluation protocols as ChartMimic (Yang et al. 2024). On the ChartEdit and ChartMimic benchmarks, we report results using each benchmark's official metrics to ensure fair comparison.

Results

Main Results. As shown in Table 3, our 3B **ChartEditor** sets a new state-of-the-art for open-source models on the in-domain ChartEditVista benchmark with an average score of 58.1. It significantly surpasses both prior specialized models like TinyChart (12.2) and much larger general-purpose VLMs like Qwen2.5-VL-72B (29.8). Notably, ChartEditor also outperforms the proprietary GPT-4o, both in average score (58.1 vs. 43.5) and execution accuracy (76.8 vs. 69.3),

SFT		Metrics					
C2C	Edit	Exec.	T_R	L_R	Type	Col.	Avg.
–	–	47.2	14.2	11.2	25.7	22.5	18.5
–	✓	**77.8**	45.6	43.9	63.7	59.8	53.2
✓	–	73.4	45.9	46.1	64.3	60.7	54.3
✓	✓	76.8	**52.9**	**48.7**	**67.5**	**64.1**	**58.1**

Table 4: Ablation on the training strategy without curriculum learning.

Curric.	Exec.	T_R	L_R	Type	Col.	Avg.
–	73.6	**53.0**	46.8	65.1	61.3	56.6
✓	**76.8**	52.9	**48.7**	**67.5**	**64.1**	**58.1**

Table 5: Ablation on curriculum learning for Chart-to-Code and Chart-Editing data with RL.

demonstrating that a compact, domain-specialized model can outperform significantly larger systems. Despite being trained on synthetic data, ChartEditor exhibits strong out-of-domain generalization. It achieves competitive scores of 55.3 on ChartEdit w/o Code and 55.0 on ChartMimic Customized, outperforming many larger models like LLaVA-1.5-13B (16.7). On the latter benchmark, it also achieves the best low-level and high-level scores among similarly-sized open-source models. More detailed results and error analysis are available in Appendix D.

Ablation Study

In this section, we provide a set of comprehensive ablations to validate the efficiency of ChartEditor, furthermore, we provide significance-test and task ablation in Appendix D.

Impact of SFT Data Sources. As shown in Table 4. We ablate the SFT data while keeping curriculum learning and RL fixed. Without SFT, the model performs poorly (47.2 Exec, 18.5 Avg). Training only on *ChartEditing* data significantly boosts execution to 77.8 and average to 53.2. Using only *Chart2Code* yields comparable execution (73.4) and slightly better fine-grained metrics (54.3 Avg), showing its complementary effect. Combining both achieves the best results (76.8 Exec, 58.1 Avg), confirming that structural and edit-specific data provide additive benefits.

Effect of Curriculum Learning. With RL, Chart2Code, and ChartEditing data fixed, introducing curriculum learning increases execution accuracy from 73.6 to 76.8 and improves the overall score from 56.6 to 58.1, as shown in Table 5. The main improvements are observed in the visual-related metrics, including Layout (increase of 1.9), Type (increase of 2.4), and Color (increase of 2.8), while the Text score remains nearly unchanged. These results indicate that progressive curriculum learning stabilizes training and leads to more accurate and visually consistent chart editing.

Reward Design. As shown in Table 6, we analyze the effect of progressively adding reward components: format (**F**), execution (**E**), and rendering (**R**). Starting with only the format reward, the model achieves 50.8 average score and 66.3% execution accuracy, indicating limited capability without execution guidance. Adding the execution reward (E + F) significantly improves performance to 54.6 average and 73.8 execution, reflecting the crucial role of executable correctness. Incorporating the rendering reward instead of format (R + E) further boosts results to 57.2 average and 75.3 execution, mainly enhancing visual-related metrics. Finally, combining all three rewards yields the best overall results, with 58.1 average and 76.8 execution accuracy. The results indicate that the execution reward ensures code correctness, the rendering reward enhances visual fidelity, and the format reward enforces output consistency and parsability. Combining these rewards yields complementary improvements in overall model performance. To further evaluate the applicability of rendering reward, we apply it to chart-to-code task, which denotes Qwen2.5-VL-3B-Render, as shown in Table7, our rendering reward can also achieve comparable performance on ChartMimic Direct Mimic, demonstrate the benefits of rendering reward in chart-code related tasks.

Rewards			Metrics					
R	**E**	**F**	Exec.	T_R	L_R	Type	Col.	Avg.
–	–	✓	66.3	47.6	43.3	58.0	54.3	50.8
–	✓	✓	73.8	46.5	47.0	64.3	53.7	54.6
✓	✓	–	75.3	**53.4**	47.8	66.0	61.5	57.2
✓	✓	✓	**76.8**	52.9	**48.7**	**67.5**	**64.1**	**58.1**

Table 6: Ablation on reward components. R: rendering, E: execution, F: format.

Model	Exec.	Low	High	Overall
GPT-4o (Hurst et al. 2024)	**93.2**	**79.0**	**83.5**	**81.2**
GeminiProVision (Team et al. 2023)	68.2	53.8	53.3	53.6
Claude-3-opus (Anthropic 2024)	83.3	60.5	60.1	60.3
InternVL2-8B (Chen et al. 2024)	61.8	34.4	38.9	36.6
Qwen2-VL-7B (Wang et al. 2024)	67.0	32.9	35.0	34.0
LLaVA-Mistral-7B (Liu et al. 2023b)	59.7	20.7	21.3	21.0
InternVL2-4B (Chen et al. 2024)	66.2	33.8	38.4	36.1
Qwen2.5-VL-3B (Bai et al. 2025)	49.3	35.2	17.0	26.1
Qwen2.5-VL-3B-SFT	49.3	35.2	17.0	26.1
Qwen2.5-VL-3B-Render	**69.6**	**52.7**	**56.4**	**54.5**

Table 7: Results on ChartMimic Direct Mimic task.

Conclusion

We present ChartEditVista, a comprehensive chart editing benchmark with broad coverage and high-quality annotations, together with two rendering-aware evaluation metrics that robustly capture fine-grained visual and textual modifications. Leveraging these resources, our ChartEditor model, trained via GRPO with a novel rendering reward and curriculum learning, achieves state-of-the-art results on both in-domain and out-of-domain benchmarks. These contributions establish a new foundation for the rigorous evaluation and development of MLLMs for chart editing.

Acknowledgments

This work was supported by the Beijing Natural Science Foundation (No. L233008).

References

Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Anthropic. 2025. Claude 3.7 Sonnet. https://www.anthropic.com/news/claude-3-7-sonnet.

Anthropic, A. 2024. Introducing the next generation of claude. *https://www. anthropic. com/news/claude-3-family*.

Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. 2025. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Chen, X.; Fang, Y.; Li, J.; Xiao, Q.; Lin, J.; Tang, S.; and Zhuang, Y. 2025. Chart-HQA: A Benchmark for Hypothetical Question Answering in Charts. In *Proceedings of the 33rd ACM International Conference on Multimedia*, 13297–13303.

Chen, Z.; Wu, J.; Wang, W.; Su, W.; Chen, G.; Xing, S.; Zhong, M.; Zhang, Q.; Zhu, X.; Lu, L.; et al. 2024. Internvl: Scaling up vision foundation models and aligning for generic visual-linguistic tasks. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 24185–24198.

Comanici, G.; Bieber, E.; Schaekermann, M.; Pasupat, I.; Sachdeva, N.; Dhillon, I.; Blistein, M.; Ram, O.; Zhang, D.; Rosen, E.; et al. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Dong, X.; Zhang, P.; Zang, Y.; Cao, Y.; Wang, B.; Ouyang, L.; Wei, X.; Zhang, S.; Duan, H.; Cao, M.; et al. 2024a. Internlm-xcomposer2: Mastering free-form text-image composition and comprehension in vision-language large model. *arXiv preprint arXiv:2401.16420*.

Dong, X.; Zhang, P.; Zang, Y.; Cao, Y.; Wang, B.; Ouyang, L.; Zhang, S.; Duan, H.; Zhang, W.; Li, Y.; et al. 2024b. Internlm-xcomposer2-4khd: A pioneering large vision-language model handling resolutions from 336 pixels to 4k hd. *Advances in Neural Information Processing Systems*, 37: 42566–42592.

Guo, D.; Yang, D.; Zhang, H.; Song, J.; Zhang, R.; Xu, R.; Zhu, Q.; Ma, S.; Wang, P.; Bi, X.; et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Han, Y.; Zhang, C.; Chen, X.; Yang, X.; Wang, Z.; Yu, G.; Fu, B.; and Zhang, H. 2023. Chartllama: A multimodal llm for chart understanding and generation. *arXiv preprint arXiv:2311.16483*.

Huang, K.-H.; Chan, H. P.; Fung, Y. R.; Qiu, H.; Zhou, M.; Joty, S.; Chang, S.-F.; and Ji, H. 2024. From pixels to insights: A survey on automatic chart understanding in the era of large foundation models. *IEEE Transactions on Knowledge and Data Engineering*.

Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Laurençon, H.; Tronchon, L.; Cord, M.; and Sanh, V. 2024. What matters when building vision-language models? *Advances in Neural Information Processing Systems*, 37: 87874–87907.

Li, B.; Zhang, K.; Zhang, H.; Guo, D.; Zhang, R.; Li, F.; Zhang, Y.; Liu, Z.; and Li, C. 2024. LLaVA-NeXT: Stronger LLMs Supercharge Multimodal Capabilities in the Wild.

Lin, Z.; Liu, C.; Zhang, R.; Gao, P.; Qiu, L.; Xiao, H.; Qiu, H.; Lin, C.; Shao, W.; Chen, K.; et al. 2023. Sphinx: The joint mixing of weights, tasks, and visual embeddings for multi-modal large language models. *arXiv preprint arXiv:2311.07575*.

Liu, H.; Li, C.; Li, Y.; and Lee, Y. J. 2024. Improved baselines with visual instruction tuning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 26296–26306.

Liu, H.; Li, C.; Wu, Q.; and Lee, Y. J. 2023a. Visual instruction tuning. *Advances in neural information processing systems*, 36: 34892–34916.

Liu, H.; Li, C.; Wu, Q.; and Lee, Y. J. 2023b. Visual instruction tuning. *Advances in neural information processing systems*, 36: 34892–34916.

Masry, A.; Long, D. X.; Tan, J. Q.; Joty, S.; and Hoque, E. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244*.

OpenAI. 2025. GPT-4.1 Announcement. https://openai.com/index/gpt-4-1/. Accessed: 2025-08-03.

QwenLM. 2025. Qwen3-VL.

Shao, Z.; Wang, P.; Zhu, Q.; Xu, R.; Song, J.; Bi, X.; Zhang, H.; Zhang, M.; Li, Y.; Wu, Y.; et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. *URL https://arxiv.org/abs/2402.03300*, 2(3): 5.

Team, G.; Anil, R.; Borgeaud, S.; Alayrac, J.-B.; Yu, J.; Soricut, R.; Schalkwyk, J.; Dai, A. M.; Hauth, A.; Millican, K.; et al. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Wang, P.; Bai, S.; Tan, S.; Wang, S.; Fan, Z.; Bai, J.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; et al. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Xia, R.; Zhang, B.; Ye, H.; Yan, X.; Liu, Q.; Zhou, H.; Chen, Z.; Ye, P.; Dou, M.; Shi, B.; et al. 2024. Chartx & chartvlm: A versatile benchmark and foundation model for complicated chart reasoning. *arXiv preprint arXiv:2402.12185*.

Xu, Z.; Qu, B.; Qi, Y.; Du, S.; Xu, C.; Yuan, C.; and Guo, J. 2024. ChartMoE: Mixture of diversely aligned expert connector for chart understanding. *arXiv preprint arXiv:2409.03277*.

Yan, P.; Bhosale, M.; Lal, J.; Adhikari, B.; and Doermann, D. 2024a. Chartreformer: Natural language-driven chart image editing. In *International Conference on Document Analysis and Recognition*, 453–469. Springer.

Yan, P.; Bhosale, M.; Lal, J.; Adhikari, B.; and Doermann, D. 2024b. Chartreformer: Natural language-driven chart image editing. In *International Conference on Document Analysis and Recognition*, 453–469. Springer.

Yang, C.; Shi, C.; Liu, Y.; Shui, B.; Wang, J.; Jing, M.; Xu, L.; Zhu, X.; Li, S.; Zhang, Y.; et al. 2024. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *arXiv preprint arXiv:2406.09961*.

Ye, Q.; Xu, H.; Xu, G.; Ye, J.; Yan, M.; Zhou, Y.; Wang, J.; Hu, A.; Shi, P.; Shi, Y.; et al. 2023a. mplug-owl: Modularization empowers large language models with multimodality. *arXiv preprint arXiv:2304.14178*.

Ye, Q.; Xu, H.; Yan, M.; Zhao, C.; Wang, J.; Yang, X.; Zhang, J.; Huang, F.; Sang, J.; and Xu, C. 2023b. mPLUG-Octopus: The Versatile Assistant Empowered by A Modularized End-to-End Multimodal LLM. In *Proceedings of the 31st ACM International Conference on Multimedia*, 9365–9367.

Ye, Q.; Xu, H.; Ye, J.; Yan, M.; Hu, A.; Liu, H.; Qian, Q.; Zhang, J.; and Huang, F. 2024. mplug-owl2: Revolutionizing multi-modal large language model with modality collaboration. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition*, 13040–13051.

Zhang, L.; Hu, A.; Xu, H.; Yan, M.; Xu, Y.; Jin, Q.; Zhang, J.; and Huang, F. 2024. Tinychart: Efficient chart understanding with visual token merging and program-of-thoughts learning. *arXiv preprint arXiv:2404.16635*.

Zhang, P.; Dong, X.; Wang, B.; Cao, Y.; Xu, C.; Ouyang, L.; Zhao, Z.; Duan, H.; Zhang, S.; Ding, S.; et al. 2023. Internlm-xcomposer: A vision-language large model for advanced text-image comprehension and composition. *arXiv preprint arXiv:2309.15112*.

Zhang, Z.; Ma, W.; and Vosoughi, S. 2024. Is GPT-4V (ision) All You Need for Automating Academic Data Visualization? Exploring Vision-Language Models' Capability in Reproducing Academic Charts. In Al-Onaizan, Y.; Bansal, M.; and Chen, Y.-N., eds., *Findings of the Association for Computational Linguistics: EMNLP 2024*, 8271–8288. Miami, Florida, USA: Association for Computational Linguistics.

Zhao, X.; Liu, X.; Yang, H.; Luo, X.; Zeng, F.; Li, J.; Shi, Q.; and Chen, C. 2025a. ChartEdit: How Far Are MLLMs From Automating Chart Analysis? Evaluating MLLMs' Capability via Chart Editing. *arXiv preprint arXiv:2505.11935*.

Zhao, X.; Luo, X.; Shi, Q.; Chen, C.; Wang, S.; Liu, Z.; and Sun, M. 2025b. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *arXiv preprint arXiv:2501.06598*.