RECHART: REASONING-EFFICIENT CHART EDITING WITH LARGE REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Chart editing requires inferring and modifying visualization code from a reference chart image based on an editing instruction, challenging fine-grained visual reasoning, instruction following, and executable code synthesis capabilities of MLLMs. Large reasoning models (LRMs) with extended Chain-of-Thought (CoT) reasoning are suitable for tackling such complex multimodal tasks. However, our preliminary study reveals an "inverted-U" relationship between reasoning length and chart-editing performance: Excessive reasoning often leads to "overthinking," where models drift toward hallucinated visual details or get stuck in redundant reasoning loops. To address the gap, we introduce *REChart*, a two-stage training framework that provides process-level supervision over intermediate reasoning steps, improving both editing fidelity and reasoning efficiency. First, we synthesize 200k high-quality reasoning trajectories for supervised fine-tuning from a large image-instruction-code pool, using a role-specialized agentic Reason-Score-Refine workflow that iteratively refine the chart code toward higher quality. Second, we optimize the model via reinforcement learning with two complementary rewards: a *fidelity* reward evaluating code correctness, visual fidelity, and structural consistency, and an *efficiency* reward that assigns each rollout a random thinking budget, truncates the reasoning process, and credits the final reasoning segment according to its contribution to the output. On the ChartEdit and ChartMIMIC benchmarks, our model achieves state-of-the-art chart-editing performance among open-source models of comparable scale, while mitigating overthinking and reducing average reasoning token usage by 79.0% under a maximum thinking budget of 16,384 tokens compared with the base model.

1 INTRODUCTION

Refinement of visual content plays an important role in scientific research and business intelligence (Friendly, 2008). Chart editing, which involves iteratively adapting and refining visualization code by jointly interpreting source chart images and nuanced editing instructions, is a key aspect of this refinement process. Effective chart editing requires models to refine visual elements, adjust visual encodings, and adapt charts to new data, making it a challenging task for evaluating the reasoning and code generation capabilities of multimodal large language models (MLLMs) (Yang et al., 2025a; Zhao et al., 2025a).

The advent of large reasoning models (LRMs), characterized by extended Chain-of-Thought (CoT) processes, has shown immense potential in tackling complex reasoning tasks (Wei et al., 2022). While prior evidence suggests that scaling the length of reasoning tokens often correlates with improved performance (Guo et al., 2025; Team, 2025a; OpenAI, 2024), recent studies have identified a non-monotonic relationship (Aggarwal et al., 2026; Chen et al., 2025a; Sui et al., 2025). Specifically, excessive reasoning may trigger overthinking, in which models become preoccupied with trivial details or amplify initial errors, leading to a noticeable decline in output quality (Feng et al., 2025; Wu et al., 2026). To address this, several works have explored strategies such as length-penalized training, budget forcing, and adaptive reasoning depth to encourage more efficient use of reasoning tokens (Xiang et al., 2025; Muennighoff et al., 2025; Li et al., 2026b). Despite these efforts, it remains unclear whether extended CoT improves chart editing, where precise visual perception must be coordinated with complex code synthesis.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

We first conduct a preliminary study investigating the scaling behavior of LRMs on the ChartEdit benchmark (Zhao et al., 2025a) under varying thinking budgets. Our findings reveal a distinct inverted-U relationship between reasoning token length and model performance: while initial CoT expansion enhances accuracy, excessive reasoning eventually triggers performance degradation as the model hallucinates visual details and falls into circular reasoning. Besides, we observe that stronger models reach their peak performance with shorter reasoning chains, while harder tasks demand longer optimal budgets. Together, these findings expose interrelated weaknesses of current LRMs in chart editing: *reasoning efficiency* and *editing fidelity*, as models routinely overshoot the optimal thinking budget that may lead to visual hallucinations and inconsistent code generation. While recent advances in multimodal RL have introduced dense rewards for terminal code structures (Chen et al., 2026a; 2025b), they largely leave the intermediate reasoning process unsupervised, which may lead to correct code via inconsistent thinking.

To enhance the model's reasoning efficiency and editing fidelity, we propose a two-stage training framework. In the first stage, we construct a large-scale SFT dataset of 200k trajectories via a multi-agent collaborative generation framework. Our framework generates trajectories that instantiate a Reason-Score-Refine loop, progressively refining intermediate outputs toward high-quality code generation. In the second stage, we optimize the model via reinforcement learning with a hybrid reward that jointly guides the reasoning process and the final output. Specifically, the reward combines an efficiency reward that samples a thinking budget for each rollout and credits the final reasoning segment before truncation, and a fidelity reward evaluating the finally generated chart across three dimensions, including code correctness, visual fidelity, and structural consistency.

We evaluate our model on ChartEdit and ChartMimic benchmarks across multiple tasks, achieving state-of-the-art performance among models of comparable scale. Compared to the base model, our approach yields a 13.3-point average overall performance gain across ChartEdit, ChartMIMIC Customized Mimic, and ChartMIMIC Direct Mimic. Beyond accuracy, our method substantially improves reasoning efficiency. Under a maximum thinking budget of 16,384 tokens, average reasoning token consumption is reduced by 79.0% relative to the base model, with the largest reduction of 83.0% on ChartMIMIC Customized Mimic.

In summary, our contributions are as follows:

- We present a systematic empirical study revealing a distinct inverted-U relationship between reasoning token length and chart editing performance.
- We propose *REChart*, a two-stage training framework comprising SFT with a structured Reason-Score-Refine trajectory dataset and reinforcement learning with a hybrid reward guiding both the reasoning process and final output.
- Extensive evaluations demonstrate that *REChart* achieves state-of-the-art chart editing performance among open-source models of comparable scale, with a 13.3-point average overall improvement and 79.0% reduction in reasoning token cost over the base model.

2 Related Work

2.1 MLLMs for Chart Understanding

Chart understanding with MLLMs is a rapidly growing field with many directions (Huang et al., 2025; Davila et al., 2021). Some works focus on generating natural language outputs from charts, including chart question answering and summarization (Masry et al., 2022; 2023; Liu et al., 2024; Zeng et al., 2025; Rahman et al., 2023; Kantharaj et al., 2022; Zhang et al., 2024), and also table extraction from charts to facilitate reasoning (Liu et al., 2023a;b; Xu et al., 2025; Meng et al., 2024). Other works focus on chart-to-code generation, with dedicated benchmarks covering diverse chart types and evaluation dimensions (Yang et al., 2025a; Wu et al., 2025; Zhao et al., 2025b;a). To improve generation fidelity, recent methods leverage reinforcement learning with both textual and visual feedback. For example, Tan et al. (2025) proposed a chart similarity reward measuring both attribute and visual similarity, while Chen et al. (2026a) introduced a multi-granularity structured reward combining textual and visual evaluation. While these methods improve chart-to-code generation quality, they do not optimize the reasoning process itself, nor do they explore the applicability of their approaches to large reasoning models. It remains unclear how to harness stronger reasoning

capabilities efficiently for chart editing, a task that inherently demands long and complex reasoning to interpret visual elements, understand user intent, and synthesize precise code modifications.

2.2 Large Reasoning Models

Large reasoning models (LRMs) such as OpenAI o1 (OpenAI, 2024), DeepSeek-R1 (Guo et al., 2025), and Kimi k1.5 (Team, 2025b) extend the capabilities of language models by generating explicit chain-of-thought reasoning traces before producing a final answer. This paradigm has demonstrated substantial gains on tasks (Wei et al., 2022; Sprague et al., 2025; Chen et al., 2023) such as complex logical inference, mathematical reasoning, and structured code generation, where deliberate step-by-step thinking proves critical to correctness. However, the extended reasoning process also introduces significant token overhead, as models tend to generate verbose and sometimes redundant thinking traces regardless of task (Sui et al., 2025; Balachandran et al., 2025). This has motivated a growing line of work on reasoning efficiency, exploring strategies such as length-penalized training (Xiang et al., 2025; Liu et al., 2026; Aggarwal & Welleck, 2025; Hou et al., 2026), budget forcing (Muennighoff et al., 2025; Han et al., 2025; Yang et al., 2026), and adaptive reasoning depth (Li et al., 2026b; Zhang et al., 2025; Xiang et al., 2025; Wang et al., 2025) to reduce unnecessary computation while preserving output quality. Chart editing is a particularly representative case, as it demands both fine-grained visual perception and structured code synthesis, making efficient reasoning critical for both quality and computational cost.

3 Problem Definition and Preliminary Study

3.1 Problem Definition

We focus on the chart-editing task, where a model generates updated chart code given a reference chart image and a natural-language editing instruction. A chart is formally represented as a tuple (I, C), where I denotes the rendered image and C denotes the executable script (e.g., using Matplotlib or Seaborn in Python) that produces I. Given a reference chart image I_{ref} and an editing instruction T, a model $\mathcal{M}$ outputs an output script:

$$C_{out} = \mathcal{M}(I_{\text{ref}}, T). \tag{1}$$

The reference code C_{ref} remains hidden during inference, requiring the model to ground edits in visual perception rather than direct code manipulation. Our goal is to develop a model that can effectively grounds edits along three dimensions: *code correctness* and *structural consistency* of C_{out}, and *visual fidelity* of I_{out} generated from C_{out}.

Given the diversity of visual components and editing instructions, chart editing is inherently complex. To systematically address this complexity, we built a taxonomy of edit operations through a multi-stage process. First, we aggregated a diverse set of editing samples from prior chart-editing datasets (Zhao et al., 2025a; Yang et al., 2025b) and editing tasks documented in the visualization community (Vaithilingam et al., 2024; Xie et al., 2026). Next, two visualization experts independently labeled and clustered these samples based on the induced operation. The experts focused on the type of modification (e.g., adding a data series, changing a color palette, rotating axis labels, filtering data points) rather than the specific code implementation. Through iterative consolidation, this process yielded six non-overlapping edit categories formally defined in Table 1. Among these, *Direct Reproduction* serves as a no-edit baseline isolating raw image-to-code capability, and the remaining five categories encompass the diverse edit operations we distilled.

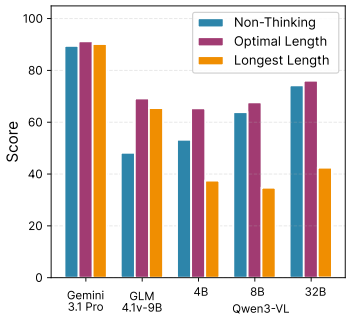
3.2 Reasoning Scaling Behavior of LRMs

We conduct an empirical study on the ChartEdit benchmark (Zhao et al., 2025a), evaluating the Qwen3-VL series (4B, 8B, 32B) (Bai et al., 2025a), GLM-4.1V-9B (Hong et al., 2025), and Gemini 3.1 Pro (Pichai et al., 2025) under varying thinking budgets.

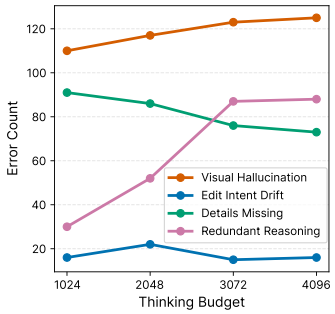
Figure 1a shows a consistent inverted-U pattern between reasoning length and chart-editing performance. Allowing LRMs to reason up to an appropriate budget improves accuracy over the non-thinking setting, indicating that explicit reasoning helps models coordinate visual perception, instruction following, and code synthesis. However, extending the budget to the longest setting often

Table 1: Taxonomy of chart-editing tasks across six functional categories.

Category	Key Editing Aspects
Direct Reproduction	No-edit baseline: reconstruction of the input chart from image to code.
Content Manipulation	Data filtering, aggregation, point/series addition or removal, and trend-line overlays.
Style & Aesthetics	Modifications to backgrounds, grids, markers, colormaps, and visual effects.
Layout Edits	Subplot composition, legend and colorbar positioning, spacing, and figure sizing.
Chart Transformation	Chart type swapping, orientation changes, and coordinate system transformations.
Axes & Annotations	Title and label content and typography, axis scaling, ticks, and data annotations.



(a) Model performance under non-thinking, optimal-budget, and longest-budget settings.



(b) Error patterns in Qwen3-VL-8B reasoning under increasing thinking budgets.

Figure 1: Reasoning scaling on ChartEdit. (a) Performance follows an inverted-U trend with reasoning length. (b) Error patterns shift toward hallucination and redundancy as the budget grows.

degrades performance, especially for open-source models. This suggests that additional reasoning is not uniformly beneficial: after a model passes its effective reasoning budget, the chain of thought can drift away from the visual evidence and reduce final execution or visual fidelity.

To understand why this degradation occurs, we further analyze Qwen3-VL-8B on the ChartEdit human set and categorize the errors appearing in its reasoning traces under different thinking budgets. As shown in Figure 1b, longer budgets do not simply add useful intermediate steps. Instead, visual hallucination steadily increases as the model spends more tokens describing chart elements that are not present or over-specifying visual details that later misguide the generated code. Redundant reasoning also grows sharply with the budget, showing that the model often revisits the same observations without producing new evidence for the edit. By contrast, details-missing errors decrease as the budget increases, indicating that longer reasoning initially helps the model inspect more visual components. The resulting trade-off explains the inverted-U behavior: moderate reasoning reduces under-analysis, but excessive reasoning introduces hallucinated or repetitive intermediate claims that eventually harm chart-editing performance.

4 RECHART

4.1 REASONING TRAJECTORY CONSTRUCTION FOR CHART EDITING

To train a model capable of efficient and high-fidelity chart-to-code generation, we build the supervised training data by first curating an image-instruction-code corpus from real-world scientific figures and then synthesizing structured reasoning trajectories from it.

4.1.1 INSTRUCTION CORPUS CONSTRUCTION

In this work, we construct a real-world dataset of 709k image-instruction-code triplets for fine-tuning. The corpus construction pipeline has three stages: 1) chart image collection, 2) code synthesis, and 3) editing augmentation (see Figure 2).

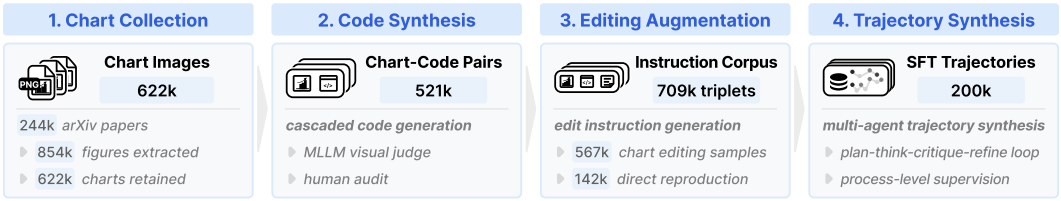


Figure 2: The data collection and annotation pipeline consists of three stages: (1) Chart Collection and Filtering; (2) Chart-Code Pair Generation; and (3) Chart Editing Augmentation.

Chart Image Collection. To cover diverse real-world chart styles and topics, we collect 2024 arXiv papers from computer science, bioinformatics, and mathematics, retaining roughly 100k CC BY 4.0 articles. From their LaTeX sources, we extract 854k embedded figures and use an MLLM classifier to filter out non-chart figures, yielding 622k chart images.

Code Synthesis. To balance efficiency and quality, we synthesize visualization code with a tiered set of generators (Qwen3-VL-32B, Qwen3-VL-235B-A22B, and Gemini 3 Flash). Rather than requiring pixel-level reproduction, we retain image-code pairs that pass a Qwen3-VL-32B audit on readability, completeness, visual integrity, and code-visual consistency; failed cases are forwarded to the next generator in the cascade. To verify audit reliability, we manually inspect a random sample of 1,000 pairs. This process yields 521k high-quality image-code pairs, with full prompts and filtering statistics provided in Appendix B.

Editing Augmentation. We then build the editing portion of the dataset by prompting Qwen3-VL-32B to generate instruction-code pairs over the five editing categories defined in Table 1. Since each of the 521k chart-code pairs is prompted three times, the accepted candidates across all pairs amount to 567k editing samples, while the 142k pairs for which no candidate passes the audit are repurposed as direct-reproduction samples. The final dataset comprises approximately 709k image-instruction-code triplets spanning the six functional dimensions of the taxonomy, covering both direct reproduction and chart editing tasks. Detailed pipeline statistics, model assignments, and prompts are provided in Appendix B.

4.1.2 Reasoning Trajectory Synthesis

Building on the preliminary study in Section 3, we synthesize SFT trajectories that provide explicit step-level supervision to address the overthinking problem of LRMs in chart editing. Naively prompting a single model to produce longer CoT traces does not resolve these issues, since the resulting traces are still optimized only by the final answer and may contain redundant or misleading reasoning steps. To address these issues at the data-construction stage, we use a role-specialized multi-agent workflow as illustrated in Figure 3. Inspired by MRT that estimate single reasoning episode's contribution to the final outcome to guide the reasoning process (Qu et al., 2025), the workflow uses an external scorer to measure episode-level progress, prune stagnant reasoning segments, and redirect the trajectory through critique when the current path stops improving.

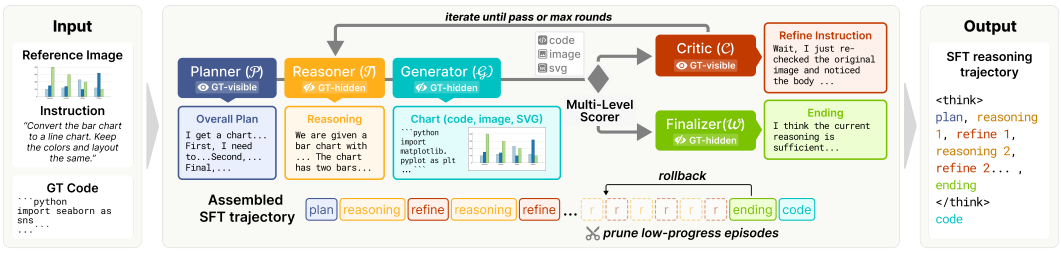


Figure 3: Multi-agent pipeline for generating the reasoning path.

(1) Agent Roles. The system comprises five role-specialized agents, each operating under a dedicated prompt. The *Planner Agent* ($\mathcal{P}$) receives the reference image, the editing instruction, and the ground-truth code, and produces a concise overall plan that decomposes the task into a sequence of

high-level steps. The *Reasoner Agent* ($\mathcal{T}$) operates only on the reference image, the instruction, and the overall plan, and at each iteration appends a single new reasoning episode to the running context, mimicking the step-wise self-talk of o1-style reasoning. The *Generator Agent* ($\mathcal{G}$) consumes the accumulated context and emits a candidate Python script. The *Critic Agent* ($\mathcal{C}$) is invoked when a candidate fails verification, and produces a first-person refine instruction (e.g., "Wait, I should re-examine the legend layout. . .") that pinpoints a concrete weakness in the current reasoning, framed as the model's own self-correction inside the trajectory. The *Finalizer Agent* ($\mathcal{F}$) is invoked once a candidate clears the quality bar, and composes a brief concluding thought that closes off the chain of thought before the final code block. The detailed implementation and setups of the pipeline are detailed in Appendix C.1.

(2) Episode-Level Reason-Score-Refine Loop. A trajectory begins with $\mathcal{P}$ producing the overall plan, after which the system enters an iterative loop in which each iteration constitutes one episode. At iteration t, $\mathcal{T}$ extends the context with a new reasoning episode, $\mathcal{G}$ produces candidate code from the updated context, and an external multi-level scorer $s_t \in [0, 100]$ evaluates the rendered chart against the reference image using the instruction, the code, and an intermediate SVG (see in Section 4.2.1). We treat the score change $\Delta_t = s_t - \max_{t'<t} s_{t'}$ as a measurable proxy for the per-episode *progress* in the sense of Qu et al. (2025): an episode that advances the running best contributes to eventual success, while one that fails to do so accrues regret without informational return.

If s_t reaches the acceptance threshold $\tau = 85$, the loop terminates and control passes to $\mathcal{F}$. Otherwise, the system applies a stagnation check before continuing. When three consecutive episodes have failed to surpass the running best ($\Delta_{t-2}, \Delta_{t-1}, \Delta_t \leq 0$), we interpret this as a high-regret excursion: the corresponding three episodes are removed from the context, and the context is rolled back to the snapshot that produced the running best. This explicit pruning prevents the trajectory from drifting along an unproductive path and bounds the regret accumulated within any single trajectory, which is the property our downstream SFT loss inherits from the data. $\mathcal{C}$ then generates a refine instruction conditioned on the current state, which is appended to the context to bias the next episode toward the deficiency it identifies, and control returns to $\mathcal{T}$. The loop is capped at 9 iterations, and trajectories that fail to converge within this budget are discarded.

(3) Trajectory Assembly. The accepted trajectory comprises the overall plan from $\mathcal{P}$, an alternating sequence of reasoning episodes from $\mathcal{T}$ and refine instructions from $\mathcal{C}$, the closing thought from $\mathcal{F}$, and the final code from $\mathcal{G}$. Following the standard o1-style format, the reasoning portion is wrapped in `<think>...</think>` tags and followed by the final output with executable code inside. By construction, the synthesized trajectories retain only reasoning episodes that collectively advance the fidelity score over successive ones, exposing the model to structured reasoning behavior where each retained sequence of steps contributes measurable progress toward the final output.

4.2 Reinforcement Finetuning

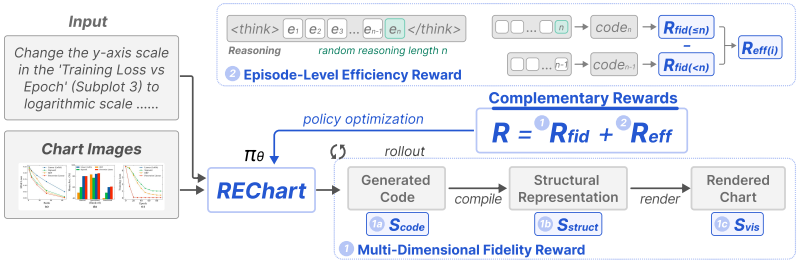


Figure 4: Overview of our RL framework guided by an episode-level efficiency reward and a multi-dimensional fidelity reward.

While SFT equips the model with structured reasoning behavior over synthesized trajectories, it may overfit to the synthetic data distribution and exhibit performance degradation on out-of-distribution editing instructions. To address this, we further optimize the model via reinforcement learning with two complementary rewards: a fidelity reward evaluating the final generated chart across code correctness, visual fidelity, and structural consistency, and an efficiency reward that randomly budgets

each rollout, truncates the reasoning process, and credits the final reasoning segment before the answer (see Figure 4).

4.2.1 FIDELITY REWARD

For each rollout, the code extracted from the model's final output is executed in a sandboxed environment to obtain both the rendered SVG and a rasterized image. We assess fidelity across the following three dimensions.

Code Similarity. We apply CodeBLEU (Ren et al., 2020) to measure the overall code similarity between the generated and ground-truth solution.

Structural Similarity. Rather than comparing raw SVG markup, we construct a lightweight structural representation of the chart. Specifically, we parse the SVG into its DOM tree, prune rendering-irrelevant nodes, and classify the `path` elements into geometric primitives based on their `d` attributes. Each node is then reduced to its essential attributes, including element type, text content, and color. The resulting simplified trees of the generated and ground-truth charts are compared to produce a structural similarity score, reflecting whether the chart's compositional layout is correctly reproduced.

Visual Fidelity. We employ a multimodal embedding model Qwen3-VL-Embedding (Li et al., 2026a) to evaluate instruction-following quality. Specifically, we measure the embedding similarity between the generated chart and the input context, which consists of the editing instruction paired with the reference chart image. This score captures whether the output faithfully reflects the required visual changes.

The fidelity reward R_{fidelity} is computed as a weighted average of the three dimension scores:

$$R_{\text{fid}} = w_{code} \cdot s_{\text{code}} + w_{struct} \cdot s_{\text{struct}} + w_{vis} \cdot s_{\text{vis}} \tag{2}$$

4.2.2 EFFICIENCY REWARD.

Optimizing solely on the outcome reward (fidelity reward $R_{fidelity}$) suffers from the credit-assignment problem (Qu et al., 2025), where a successful rollout reinforces all intermediate reasoning episodes equally, regardless of their actual contribution to the final outcome. Following Qu et al. (2025), we incorporate an episode-level efficiency reward that measures whether the final reasoning segment before the answer improves the fidelity reward under a sampled compute budget.

Concretely, for each rollout, we randomly sample a thinking budget B and allow the model to reason until it either emits the end-of-thinking token or reaches the budget. If the rollout reaches B before naturally stopping, we truncate the reasoning context and force the model to enter the answer phase. Let n denote the actual number of reasoning segments exposed under this procedure, which may be smaller than or equal to the sampled budget B. We denote by e_n the final reasoning segment immediately preceding the answer, by $h_{<n}$ the truncated context before e_n, and by $h_{\leq n} = h_{<n} \cup \{e_n\}$ the context that includes it. We decode answers from both contexts, evaluate them under the fidelity reward to obtain $R_{\text{fid}}^{(<n)}$ and $R_{\text{fid}}^{(\leq n)}$, and define the efficiency reward as:

$$R_{\text{eff}}(e_n) = \max(R_{\text{fid}}^{(\leq n)} - R_{\text{fid}}^{(<n)}, 0). \tag{3}$$

The reward therefore credits only the last reasoning segment actually exposed before answer generation, encouraging the model to make the reasoning immediately before generation concise and useful rather than rewarding arbitrarily selected spans from a completed trajectory.

4.2.3 POLICY OPTIMIZATION

The composite reward used for policy optimization is then:

$$R = w_{\text{fid}}\, R_{\text{fid}} + w_{\text{eff}}\, R_{\text{eff}}. \tag{4}$$

We optimize the model with this reward via GRPO (Shao et al., 2024), which samples a group of candidate responses per input and normalizes rewards within the group to compute relative advantages for policy updates.

Table 2: Performance on ChartEdit *w/o* Code and ChartMIMIC (customized mimic) benchmarks.

Model	# Params.	ChartEdit *w/o* Code				ChartMIMIC Customized			
		Exec.	Code	Chart	Overall	Exec.	Low	High	Overall
Proprietary Models									
GPT-4o (Hurst et al., 2024)	-	91.5	60.0	79.9	70.0	96.5	82.1	84.3	83.2
Claude-3.7-Sonnet (Anthropic, 2025)	-	96.3	73.4	87.5	80.5	80.6	67.4	78.1	72.3
Gemini-2.5-Pro (Comanici et al., 2025)	-	95.3	82.9	89.2	86.1	91.7	81.3	83.5	82.4
Open-Source Models									
Qwen2.5-VL-72B (Bai et al., 2025b)	72B	89.8	57.7	71.0	64.4	85.3	67.6	69.1	68.4
Qwen2.5-VL-32B (Bai et al., 2025b)	32B	88.4	61.7	72.5	67.1	85.6	66.4	69.5	68.0
Qwen3-VL-32B-Thinking (Bai et al., 2025a)	32B	85.5	72.9	76.3	74.6	84.0	67.0	72.3	69.6
Qwen3-VL-32B-Instruct (Bai et al., 2025a)	32B	84.9	69.0	73.4	71.2	88.2	72.4	61.3	66.9
Qwen3-VL-8B-Thinking (Bai et al., 2025a)	8B	79.1	66.6	68.0	67.3	77.6	59.6	63.4	61.5
GLM4.1v-9B-Thinking (Hong et al., 2025)	9B	78.9	68.4	68.7	68.6	80.7	63.7	65.7	64.7
Qwen3-VL-8B-Instruct (Bai et al., 2025a)	8B	84.9	59.6	69.3	64.5	78.1	61.4	60.8	61.1
LLaVA-1.5-13B (Liu et al., 2023c)	13B	48.8	11.4	16.7	14.1	49.0	23.6	24.7	24.2
ChartLlama (Han et al., 2023)	13B	52.3	15.8	27.4	21.6	-	-	-	-
ChartCoder (Zhao et al., 2025b)	7B	89.8	47.3	74.5	60.9	85.2	58.3	62.0	60.2
MSRL (Chen et al., 2026a)	7B	**97.2**	57.6	84.8	71.2	67.1	39.6	46.5	43.0
RRVF (Chen et al., 2025b)	7B	96.1	44.1	74.2	59.2	85.9	57.6	59.3	58.5
Qwen3-VL-4B-Instruct (Bai et al., 2025a)	4B	79.5	55.6	63.2	59.4	66.0	47.6	49.1	48.4
Qwen3-VL-4B-Thinking (Bai et al., 2025a)	4B	79.3	64.3	67.0	65.6	72.9	54.1	57.6	55.8
TinyChart (Zhang et al., 2024)	3B	36.3	18.3	25.1	21.7	-	-	-	-
ChartEditor (Chen et al., 2026b)	3B	66.5	40.2	55.3	47.8	61.6	52.1	57.9	55.0
REChart	8B	96.7	**73.3**	**86.1**	**79.7**	**88.6**	**67.7**	**72.5**	**70.1**

5 Experiment

5.1 Baselines and Benchmarks

We evaluate on ChartEdit *w/o* Code (Zhao et al., 2025a) and ChartMIMIC Customized and Direct Mimic (Yang et al., 2025a). Baselines comprise proprietary MLLMs (Hurst et al., 2024; Anthropic, 2025; Comanici et al., 2025), general-purpose open-source models (Liu et al., 2023c; Bai et al., 2025a; Hong et al., 2025), and chart-specialized models (Zhao et al., 2025b; Chen et al., 2026b; Zhang et al., 2024; Chen et al., 2025b). Tables 2 and 3 list the models; Appendix E provides evaluation protocols.

Table 3: Results on ChartMimic Direct Mimic task.

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
LLaVA-Mistral-7B (Liu et al., 2023c)	59.7	20.7	21.3	21.0
Qwen3-VL-32B-Thinking (Bai et al., 2025a)	82.8	65.2	72.2	68.7
Qwen3-VL-8B-Thinking (Bai et al., 2025a)	75.0	56.0	60.9	58.4
Qwen3-VL-4B-Thinking (Bai et al., 2025a)	69.8	51.5	56.0	53.7
GLM4.1v-9B-Thinking (Hong et al., 2025)	79.5	63.7	67.7	65.7
ChartEditor (Chen et al., 2026b)	69.6	52.7	56.4	54.5
ChartCoder (Zhao et al., 2025b)	91.4	72.5	74.0	73.3
RRVF (Chen et al., 2025b)	**97.8**	60.9	67.9	64.4
REChart	94.7	**72.8**	**81.5**	**77.2**

5.2 Training Setup

We fine-tune Qwen3-VL-8B-Thinking (Bai et al., 2025a) as our base model across two stages. In the SFT stage, we train on 200k synthesized trajectories for one epoch using LLaMA-Factory (Zheng et al., 2024), with a learning rate of 5e-5 and an effective batch size of 256 via gradient accumulation, completing in approximately 28 hours on 8 NVIDIA A800 80GB GPUs. In the RL stage, we further train on 82k samples for one epoch using Verl (Sheng et al., 2025), with a GRPO group size of 8. For the hybrid reward, the three fidelity dimensions are weighted equally ($w_{\text{code}} = w_{\text{struct}} =$

Table 4: Overall scores for cumulative training and reward ablations.

Variant	ChartEdit ↑	ChartMIMIC ↑	
		Custom.	Direct
Base	67.30	61.50	58.40
+ SFT	73.40	63.90	67.80
+ Base GRPO	79.60	**70.66**	77.59
+ Budgeted GRPO	**80.24**	68.76	**78.96**
+ Efficiency Reward	79.71	70.09	77.15

Table 5: Mean reasoning tokens for the same variants (maximum budget: 16,384).

Variant	ChartEdit ↓	ChartMIMIC ↓	
		Custom.	Direct
Base	6521.12	8889.43	9575.09
+ SFT	2790.66	2251.27	2267.25
+ Base GRPO	3184.46	2285.46	2320.49
+ Budgeted GRPO	3040.28	1995.39	1997.75
+ Efficiency Reward	**2005.18**	**1515.22**	**1727.69**

$w_{\text{vis}} = 0.33$), and the fidelity and efficiency rewards contribute equally to the final reward ($w_{\text{fid}} = w_{\text{eff}} = 0.5$). This stage requires approximately 65 hours on 8 NVIDIA H100 GPUs. Detailed hyperparameters are provided in Appendix E.1.

5.3 Main Results

Chart Editing. *REChart* achieves overall scores of 79.71 on ChartEdit and 70.09 on Customized Mimic, exceeding all evaluated open-source baselines (Table 2). These include the larger Qwen3-VL-32B-Thinking model, which scores 74.6 and 69.6, respectively, and chart-specialized models. Proprietary models retain the strongest overall scores, although our 8B model narrows the gap.

Direct Chart Reproduction. On Direct Mimic, *REChart* scores 77.15, compared with 73.3 for ChartCoder, the strongest open-source baseline (Table 3). This supports transfer from chart editing to direct image-to-code generation, while GPT-4o remains ahead.

5.4 Ablation Study

Training and Reward Ablation. SFT improves overall scores by 6.10, 2.40, and 9.40 points on ChartEdit, Customized Mimic, and Direct Mimic, respectively (Table 4). Base GRPO further improves all three settings. Budgeted GRPO achieves the highest ChartEdit and Direct Mimic scores (80.24 and 78.96), but reduces Customized Mimic performance relative to base GRPO. Adding the efficiency reward retains most of the GRPO gains while trading small score changes for lower reasoning cost.

Efficiency Reward Ablation. Under a maximum thinking budget of 16,384 tokens, SFT reduces mean reasoning length from 8,328.55 to 2,436.39 tokens (70.8%; Table 5). The full method lowers this to 1,749.36 tokens, a 79.0% reduction from the base model. Relative to base GRPO, token usage falls by 25.5–37.0% across the three settings with small changes in overall scores, improving the efficiency–accuracy trade-off.

6 Conclusion

We present *REChart*, a two-stage framework for improving chart-editing fidelity and reasoning efficiency. Our preliminary study shows that reasoning helps chart editing, but excessive reasoning introduces hallucination and redundancy that can degrade performance. To address this, we synthesize 200k Reason-Score-Refine trajectories for SFT, then apply budgeted RL with multi-dimensional fidelity and episode-level efficiency rewards. The resulting 8B model outperforms the evaluated open-source baselines across three benchmark settings while reducing mean reasoning token usage by 79.0%. These results suggest that rewarding useful reasoning, rather than encouraging longer traces, can improve the efficiency–accuracy trade-off in chart generation.

Limitations. Due to computational constraints, we fine-tune only an 8B backbone, so the benefits of scaling to larger LRMs remain unverified. Our evaluation also focuses on chart editing and direct reproduction. Extending the framework to broader visual code generation, such as HTML and SVG illustrations, remains an important direction for future work.

AI USE STATEMENT

LLMs serve as integral components in our data construction pipeline, including chart filtering, code synthesis, editing instruction generation, and multi-agent reasoning trajectory synthesis (Sections 4.1.1 and 4.1.2); the specific models assigned to each role (Qwen3-VL-32B, Qwen3-VL-235B-A22B, Gemini 3 Flash) are detailed in the corresponding sections and the Appendix. Beyond these methodological uses, LLMs were also used solely for grammar correction and language polishing of the manuscript, with no involvement in ideation, experimental design, or derivation of the scientific contributions. We have reviewed all AI-assisted work and take responsibility for the final content of this work, including text, claims, code, data, and artifacts produced with the aid of generative AI.

ETHICS STATEMENT

We acknowledge the broader ethical implications of generative AI. Our chart corpus is built exclusively from CC BY 4.0 licensed arXiv articles, ensuring compliance with content licensing requirements. The dataset contains no personally identifiable information (PII) and has been screened to exclude offensive content. For the human audit conducted during data filtering (Section 4.1.1), all annotators participated voluntarily with informed consent and were compensated fairly; the audit involved only assessing visual fidelity and code-image consistency of synthesized chart-code pairs, posing no foreseeable risks to participants. The release of REChart and its associated dataset poses no foreseeable harm to society and does not facilitate the generation of malicious content. All base models used in our pipeline are publicly available, and we will release our code, data, and model checkpoints upon publication.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide a representative subset of the dataset, source code, and scripts in the supplemental material, with the full 200k SFT trajectory dataset and model checkpoints to be released on Hugging Face upon publication. Detailed experimental configurations, including agent prompts, reward formulations, training hyperparameters, and evaluation protocols, are provided in the Appendix.

REFERENCES

Pranjal Aggarwal and Sean Welleck. L1: Controlling how long a reasoning model thinks with reinforcement learning. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=4jdIxXBNve.

Pranjal Aggarwal, Seungone Kim, Jack Lanchantin, Sean Welleck, Jason E Weston, Ilia Kulikov, and Swarnadeep Saha. Optimalthinkingbench: Evaluating over and underthinking in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=N5kWa3sRJt.

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

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

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Vidhisha Balachandran, Jingya Chen, Lingjiao Chen, Shivam Garg, Neel Joshi, Yash Lara, John Langford, Besmira Nushi, Vibhav Vineet, Yue Wu, et al. Inference-time scaling for complex tasks: Where we stand and what lies ahead. *arXiv preprint arXiv:2504.00294*, 2025.

Lei Chen, Xuanle Zhao, Zhixiong Zeng, Jing Huang, Liming Zheng, Yufeng Zhong, and Lin Ma. Breaking the SFT plateau: Multimodal structured reinforcement learning for chart-to-code generation. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=14qYxv2jki.

Liangyu Chen, Yichen Xu, Jianzhe Ma, Yuqi Liu, Donglu Yang, Liang Zhang, Zihao Yue, Wenxuan Wang, and Qin Jin. Charteditor: a reinforcement learning framework for robust chart editing. In *Proceedings of the Fortieth AAAI Conference on Artificial Intelligence and Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence and Sixteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'26/IAAI'26/EAAI'26. AAAI Press, 2026b. ISBN 978-1-57735-906-7. doi: 10.1609/aaai.v40i24.39107. URL https://doi.org/10.1609/aaai.v40i24.39107.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=YfZ4ZPt8zd.

Xingyu Chen, Jiahao Xu, Tian Liang, Zhiwei He, Jianhui Pang, Dian Yu, Linfeng Song, Qiuzhi Liu, Mengfei Zhou, Zhuosheng Zhang, Rui Wang, Zhaopeng Tu, Haitao Mi, and Dong Yu. Do NOT think that much for 2+3=? on the overthinking of long reasoning models. In *Forty-second International Conference on Machine Learning*, 2025a. URL https://openreview.net/forum?id=MSbU3L7V00.

Yang Chen, Yufan Shen, Wenxuan Huang, Sheng Zhou, Qunshu Lin, Xinyu Cai, Zhi Yu, Jiajun Bu, Botian Shi, and Yu Qiao. Learning only with images: Visual reinforcement learning with reasoning, rendering, and visual feedback. *CoRR*, abs/2507.20766, 2025b. doi: 10.48550/ARXIV.2507.20766. URL https://doi.org/10.48550/arXiv.2507.20766.

Zhe Chen, Jiannan Wu, Wenhai Wang, Weijie Su, Guo Chen, Sen Xing, Muyan Zhong, Qinglong Zhang, Xizhou Zhu, Lewei Lu, et al. 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*, pp. 24185–24198, 2024.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*, 2025.

Kenny Davila, Srirangaraj Setlur, David Doermann, Bhargava Urala Kota, and Venu Govindaraju. Chart mining: A survey of methods for automated chart analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(11):3799–3819, 2021. doi: 10.1109/TPAMI.2020.2992028.

Yunzhen Feng, Julia Kempe, Cheng Zhang, Parag Jain, and Anthony Hartshorn. What characterizes effective reasoning? revisiting length, review, and structure of cot. In *NeurIPS 2025 Workshop on Efficient Reasoning*, 2025. URL https://openreview.net/forum?id=8VNId3aihj.

Michael Friendly. *A Brief History of Data Visualization*, pp. 15–56. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-33037-0. doi: 10.1007/978-3-540-33037-0_2. URL https://doi.org/10.1007/978-3-540-33037-0_2.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nat.*, 645(8081):633–638, 2025. doi: 10.1038/S41586-025-09422-Z. URL https://doi.org/10.1038/s41586-025-09422-z.

Tingxu Han, Zhenting Wang, Chunrong Fang, Shiyu Zhao, Shiqing Ma, and Zhenyu Chen. Token-budget-aware LLM reasoning. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 24842–24855, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.1274. URL https://aclanthology.org/2025.findings-acl.1274/.

Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. Chartllama: A multimodal llm for chart understanding and generation. *arXiv preprint arXiv:2311.16483*, 2023.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, et al. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2025. URL https://arxiv.org/abs/2507.01006.

Bairu Hou, Yang Zhang, Jiabao Ji, Yujian Liu, Kaizhi Qian, Jacob Andreas, and Shiyu Chang. Thinkprune: Pruning long chain-of-thought of LLMs via reinforcement learning. *Transactions on Machine Learning Research*, 2026. ISSN 2835-8856. URL https://openreview.net/forum?id=V51gPu1uQD.

Kung-Hsiang Huang, Hou Pong Chan, May Fung, Haoyi Qiu, Mingyang Zhou, Shafiq Joty, Shih-Fu Chang, and Heng Ji. From pixels to insights: A survey on automatic chart understanding in the era of large foundation models. *IEEE Transactions on Knowledge and Data Engineering*, 37(5): 2550–2568, 2025. doi: 10.1109/TKDE.2024.3513320.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Shankar Kantharaj, Rixie Tiffany Leong, Xiang Lin, Ahmed Masry, Megh Thakkar, Enamul Hoque, and Shafiq Joty. Chart-to-text: A large-scale benchmark for chart summarization. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4005–4023, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.acl-long.277. URL https://aclanthology.org/2022.acl-long.277/.

Mingxin Li, Yanzhao Zhang, Dingkun Long, Keqin Chen, Sibo Song, Shuai Bai, Zhibo Yang, Pengjun Xie, An Yang, Dayiheng Liu, et al. Qwen3-vl-embedding and qwen3-vl-reranker: A unified framework for state-of-the-art multimodal retrieval and ranking. *arXiv preprint arXiv:2601.04720*, 2026a.

Zheng Li, Qingxiu Dong, Jingyuan Ma, Di Zhang, Kai Jia, and Zhifang Sui. SelfBudgeter: Adaptive token allocation for efficient LLM reasoning. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 21135–21156, San Diego, California, United States, July 2026b. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1063. URL https://aclanthology.org/2026.findings-acl.1063/.

Fangyu Liu, Julian Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. DePlot: One-shot visual language reasoning by plot-to-table translation. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 10381–10399, Toronto, Canada, July 2023a. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.660. URL https://aclanthology.org/2023.findings-acl.660/.

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Eisenschlos. MatCha: Enhancing visual language pretraining with math reasoning and chart derendering. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12756–12770, Toronto, Canada, July 2023b. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.714. URL https://aclanthology.org/2023.acl-long.714/.

Fuxiao Liu, Xiaoyang Wang, Wenlin Yao, Jianshu Chen, Kaiqiang Song, Sangwoo Cho, Yaser Yacoob, and Dong Yu. MMC: Advancing multimodal chart understanding with large-scale instruction tuning. In Kevin Duh, Helena Gomez, and Steven Bethard (eds.), *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1287–1310, Mexico City, Mexico, June 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.70. URL https://aclanthology.org/2024.naacl-long.70/.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023c. URL https://openreview.net/forum?id=w0H2xGHlkw.

Wei Liu, Ruochen Zhou, Yiyun Deng, Yuzhen Huang, Junteng Liu, Yuntian Deng, Yizhe Zhang, and Junxian He. Learn to reason efficiently with adaptive length-based reward shaping. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hj9eKpqxQl.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Ahmed Masry, Parsa Kavehzadeh, Xuan Long Do, Enamul Hoque, and Shafiq Joty. UniChart: A universal vision-language pretrained model for chart comprehension and reasoning. In Houda Bouamor, Juan Pino, and Kalika Bali (eds.), *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 14662–14684, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.906. URL https://aclanthology.org/2023.emnlp-main.906/.

Fanqing Meng, Wenqi Shao, Quanfeng Lu, Peng Gao, Kaipeng Zhang, Yu Qiao, and Ping Luo. ChartAssistant: A universal chart multimodal language model via chart-to-table pre-training and multitask instruction tuning. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 7775–7803, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.findings-acl.463. URL https://aclanthology.org/2024.findings-acl.463/.

Niklas Muennighoff, Zitong Yang, Weijia Shi, Xiang Lisa Li, Li Fei-Fei, Hannaneh Hajishirzi, Luke Zettlemoyer, Percy Liang, Emmanuel Candès, and Tatsunori Hashimoto. s1: Simple test-time scaling. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 20275–20321, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.1025. URL https://aclanthology.org/2025.emnlp-main.1025/.

OpenAI. Openai o1 system card. *CoRR*, abs/2412.16720, 2024. doi: 10.48550/ARXIV.2412.16720. URL https://doi.org/10.48550/arXiv.2412.16720.

Sundar Pichai, Demis Hassabis, and Koray Kavukcuoglu. A new era of intelligence with gemini 3, November 2025. URL https://blog.google/products-and-platforms/products/gemini/gemini-3/#gemini-3.

Yuxiao Qu, Matthew Y. R. Yang, Amrith Setlur, Lewis Tunstall, Edward Emanuel Beeching, Ruslan Salakhutdinov, and Aviral Kumar. Optimizing test-time compute via meta reinforcement finetuning. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=TqODUDsU4u.

Raian Rahman, Rizvi Hasan, Abdullah Al Farhad, Md. Tahmid Rahman Laskar, Md. Hamjajul Ashmafee, and Abu Raihan Mostofa Kamal. Chartsumm: A comprehensive benchmark for automatic chart summarization of long and short summaries. In Amílcar Soares, Farhana H. Zulkernine, Renata Dividino, Reihaneh Rabbany, Qiang Ye, David Beach, and Karim Ali (eds.), *36th*

Canadian Conference on Artificial Intelligence, Canadian AI 2023, Montreal, Canada, June 5-9, 2023, Proceedings. Canadian Artificial Intelligence Association, 2023. doi: 10.21428/594757DB.0B1F96F6. URL https://doi.org/10.21428/594757db.0b1f96f6.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. Codebleu: a method for automatic evaluation of code synthesis. *CoRR*, abs/2009.10297, 2020. URL https://arxiv.org/abs/2009.10297.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, EuroSys '25, pp. 1279–1297, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400711961. doi: 10.1145/3689031.3696075. URL https://doi.org/10.1145/3689031.3696075.

Chufan Shi, Haoran Yang, Deng Cai, Zhisong Zhang, Yifan Wang, Yujiu Yang, and Wai Lam. A thorough examination of decoding methods in the era of LLMs. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 8601–8629, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.489. URL https://aclanthology.org/2024.emnlp-main.489/.

Zayne Rea Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=w6nlcS8Kkn.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. Stop overthinking: A survey on efficient reasoning for large language models. *Transactions on Machine Learning Research*, 2025. ISSN 2835-8856. URL https://openreview.net/forum?id=HvoG8SxggZ.

Wentao Tan, Qiong Cao, Chao Xue, Yibing Zhan, Changxing Ding, and Xiaodong He. Chartmaster: Advancing chart-to-code generation with real-world charts and chart similarity reinforcement learning. *CoRR*, abs/2508.17608, 2025. doi: 10.48550/ARXIV.2508.17608. URL https://doi.org/10.48550/arXiv.2508.17608.

Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.

Kimi Team. Kimi K2: open agentic intelligence. *CoRR*, abs/2507.20534, 2025a. doi: 10.48550/ARXIV.2507.20534. URL https://doi.org/10.48550/arXiv.2507.20534.

Kimi Team. Kimi k1.5: Scaling reinforcement learning with llms. *CoRR*, abs/2501.12599, 2025b. doi: 10.48550/ARXIV.2501.12599. URL https://doi.org/10.48550/arXiv.2501.12599.

Qwen Team. Qwen3 technical report. *CoRR*, abs/2505.09388, 2025c. doi: 10.48550/ARXIV.2505.09388. URL https://doi.org/10.48550/arXiv.2505.09388.

Priyan Vaithilingam, Elena L. Glassman, Jeevana Priya Inala, and Chenglong Wang. Dynavis: Dynamically synthesized ui widgets for visualization editing. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, CHI '24, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642639. URL https://doi.org/10.1145/3613904.3642639.

Jiaqi Wang, Kevin Qinghong Lin, James Cheng, and Mike Zheng Shou. Think or not? selective reasoning via reinforcement learning for vision-language models. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/a168b27492ec2eb7aa184815fa0cd046-Abstract-Conference.html.

Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A. Smith, Iz Beltagy, and Hannaneh Hajishirzi. How far can camels go? exploring the state of instruction tuning on open resources. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=w4zZNC4ZaV.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi, Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=_VjQlMeSB_J.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. Plot2Code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. In Luis Chiruzzo, Alan Ritter, and Lu Wang (eds.), *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 3006–3028, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. ISBN 979-8-89176-195-7. doi: 10.18653/v1/2025.findings-naacl.164. URL https://aclanthology.org/2025.findings-naacl.164/.

Yuyang Wu, Yifei Wang, Ziyu Ye, Tianqi Du, Stefanie Jegelka, and Yisen Wang. When more is less: Understanding chain-of-thought length in LLMs. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=6QDFsYxtI1.

Violet Xiang, Chase Blagden, Rafael Rafailov, Nathan Lile, Sang T. Truong, Chelsea Finn, and Nick Haber. Just enough thinking: Efficient reasoning with adaptive length penalties reinforcement learning. *CoRR*, abs/2506.05256, 2025. doi: 10.48550/ARXIV.2506.05256. URL https://doi.org/10.48550/arXiv.2506.05256.

Liwenhan Xie, Yanna Lin, Can Liu, Huamin Qu, and Xinhuan Shu. Datawink: Reusing and adapting svg-based visualization examples with large multimodal models. *IEEE Transactions on Visualization and Computer Graphics*, 32(1):824–834, 2026. doi: 10.1109/TVCG.2025.3634635.

Zhengzhuo Xu, Bowen Qu, Yiyan Qi, SiNan Du, Chengjin Xu, Chun Yuan, and Jian Guo. Chartmoe: Mixture of diversely aligned expert connector for chart understanding. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=o5TsWTUSeF.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran XU, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. Chartmimic: Evaluating LMM's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations*, 2025a. URL https://openreview.net/forum?id=sGpCzsfd1K.

Chenxu Yang, Qingyi Si, Yongjie Duan, Zheliang Zhu, Chenyu Zhu, Qiaowei Li, Minghui Chen, Zheng Lin, and Weiping Wang. Dynamic early exit in reasoning models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=NpU7ZXafRi.

Donglu Yang, Liang Zhang, Zihao Yue, Liangyu Chen, Yichen Xu, Wenxuan Wang, and Qin Jin. ChartM^3: Benchmarking chart editing with multimodal instructions. In *Proceedings of the 33rd*

ACM International Conference on Multimedia, MM '25, pp. 5001–5009, New York, NY, USA, 2025b. Association for Computing Machinery. ISBN 9798400720352. doi: 10.1145/3746027.3755714. URL https://doi.org/10.1145/3746027.3755714.

Xingchen Zeng, Haichuan Lin, Yilin Ye, and Wei Zeng. Advancing multimodal large language models in chart question answering with visualization-referenced instruction tuning. *IEEE Transactions on Visualization and Computer Graphics*, 31(1):525–535, 2025. doi: 10.1109/TVCG.2024.3456159.

Jiajie Zhang, Nianyi Lin, Lei Hou, Ling Feng, and Juanzi Li. AdaptThink: Reasoning models can learn when to think. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 3716–3730, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-332-6. doi: 10.18653/v1/2025.emnlp-main.184. URL https://aclanthology.org/2025.emnlp-main.184/.

Liang Zhang, Anwen Hu, Haiyang Xu, Ming Yan, Yichen Xu, Qin Jin, Ji Zhang, and Fei Huang. TinyChart: Efficient chart understanding with program-of-thoughts learning and visual token merging. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 1882–1898, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.112. URL https://aclanthology.org/2024.emnlp-main.112/.

Xuanle Zhao, Xuexin Liu, Yang Haoyue, Xianzhen Luo, Fanhu Zeng, Jianling Li, Qi Shi, and Chi Chen. ChartEdit: How far are MLLMs from automating chart analysis? evaluating MLLMs' capability via chart editing. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3616–3630, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.185. URL https://aclanthology.org/2025.findings-acl.185/.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ChartCoder: Advancing multimodal large language model for chart-to-code generation. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7333–7348, Vienna, Austria, July 2025b. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.363. URL https://aclanthology.org/2025.acl-long.363/.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, and Zheyan Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In Yixin Cao, Yang Feng, and Deyi Xiong (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pp. 400–410, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-demos.38. URL https://aclanthology.org/2024.acl-demos.38/.

A Experimental Details of the Preliminary Study

A.1 Model Configuration

We follow the standard configuration adopted in prior work Wang et al. (2023); Shi et al. (2024) when querying the evaluated models. For the open-source models in our suite (the Qwen3-VL series and GLM-4.1v-9B), we set the sampling temperature to 0.1 and serve each model through the latest release of vLLM on a single NVIDIA A800 (80GB) GPU. For the proprietary Gemini 3.1 Pro, we set the temperature to 0 and query it via its official API endpoint. All other decoding parameters are left at their respective defaults.

A.2 Thinking Budget Implementation

Prompt for Stop Reasoning

Time is up.
I need to provide the final answer based on the existing considerations.

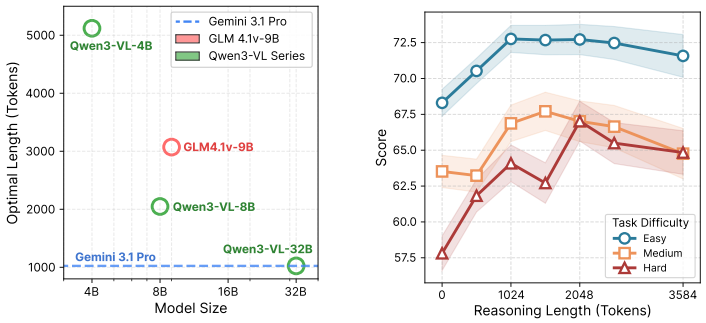
Figure 5: Prompt for terminating the reasoning phase within the given thinking budget and generating the final output.

We implement the thinking budget through a simple two-stage decoding scheme commonly used in prior work (Team, 2025c; Qu et al., 2025). In the first stage, the model is allowed to reason freely with the maximum generation length set to the target thinking budget B. Once generation reaches B tokens, decoding is interrupted, and we append (i) an end-of-thinking prompt that instructs the model to stop reasoning and produce the final code (Figure 5) and (ii) the special end-of-thinking token `</think>`. In the second stage, the model continues decoding from this augmented context with a fixed maximum generation length of $4{,}096$ tokens, and its output is taken as the final answer (i.e., the generated code). For the no-thinking baseline, we skip the first stage entirely and directly prompt the model with the end-of-thinking marker, so that decoding begins in the answer phase.

A.3 Evaluation Method

We adopt the official evaluation method of the ChartEdit benchmark (Zhao et al., 2025a). Each generated script is executed in a sandboxed Python environment, and its rendered output together with the source code is then scored by GPT-4o (Hurst et al., 2024) under the official ChartEdit evaluation prompt, which assesses correctness along multiple dimensions covering both code fidelity and visual fidelity, following ChartEdit (Zhao et al., 2025a). Samples whose generated code fails to execute are assigned a score of zero on all dimensions. The final score reported for each model and thinking budget is the average across all evaluation samples.

A.4 Additional Reasoning Scaling Analysis



(a) Optimal thinking budget across models of different scale and capability.

(b) Qwen3-VL-8B performance under different thinking budgets and task difficulties.

Figure 6: Additional analysis of reasoning budget sensitivity. (a) The optimal thinking budget differs across model scales and capabilities, suggesting that stronger models can often compress useful reasoning into shorter chains. (b) For Qwen3-VL-8B, the preferred budget also varies with task difficulty: harder editing tasks benefit from longer reasoning before overthinking begins.

Beyond the main preliminary analysis in Section 3, we further examine two factors that modulate the effective reasoning budget. Figure 6a compares the optimal thinking budget across models and shows that the best budget is not fixed across model families or scales. Figure 6b analyzes Qwen3-VL-8B across task difficulties and shows that harder examples can benefit from longer reasoning, although performance still declines once the model exceeds a useful budget. These observations

support our choice to train models toward budget-aware and contribution-aware reasoning rather than simply encouraging longer chains of thought.

A.5 Training-Induced Reasoning Error Shift

Table 6: Reasoning error counts on the ChartEdit human set before and after training.

Variant	Visual Hallucination	Edit Intent Drift	Details Missing
Base	125	16	73
+ SFT	105	13	**66**
+ RL	**84**	**9**	69

SFT already reduces all three error types relative to the base model. RL further lowers hallucination and intent drift, while slightly increasing details-missing errors relative to SFT. This suggests that later-stage training mainly suppresses spurious reasoning rather than uniformly improving every error category.

B Dataset Construction Details

(1) Chart Collection and Filtering.

To cover the diversity of chart types and usage scenarios in real-world charts, we begin with roughly 244k arXiv papers published in 2024, of which around 100k carry a CC BY 4.0 license and form our source corpus. From their LaTeX sources, we extract about 854k figures spanning PNG, JPG, JPEG, and PDF formats. PDF figures are rasterized to PNG, and the remaining formats are normalized to PNG so that every input shares a uniform modality. We then apply Qwen3-VL-Flash to classify each figure as chart or non-chart, leaving about 622k chart images for downstream processing.

(2) Chart-Code Pair Generation.

To generate image-code pairs at scale, we adopt a three-pass cascaded strategy that progressively assigns stronger generators to harder cases, balancing cost and quality. In each pass, a generator is prompted with the chart image (full prompt in Figure 7) to produce visualization code. The rendered output is then evaluated against the reference chart by a Qwen3-VL-32B-Instruct judge along four dimensions (readability, completeness, visual integrity, and consistency), following the rubric in Figure 8. Charts whose rendered code passes the judge are retained, while those that fail are forwarded to the next pass with a stronger generator. We started with Qwen3-VL-32B-Instruct, then handed over the failed cases to Qwen3-VL-235B-A22B. The most challenging cases that are still unresolved are then forwarded to Gemini 3 Flash. This process yields around 521k high-quality chart-code pairs in total.

(3) Chart Editing Augmentation.

To extend the dataset to chart editing, we augment each chart-code pair through an editing-instruction synthesis pipeline driven by Qwen3-VL-32B-Instruct. For each pair, the model is prompted (Figure 9) to select an appropriate editing operation from the five editing categories defined in Section 3.1, compose a natural-language editing instruction, and produce the corresponding edited code. We repeat this process three times per pair to encourage operation and instruction diversity. Each candidate edit is then evaluated against the criteria in Figure 10, which rates the rendered edit on four dimensions (alignment with the instruction, integrity of unmodified regions, linguistic quality, and visual legibility), and is accepted only if it meets the required standards.

This step yields about 567k accepted editing samples. The remaining roughly 142k chart-code pairs, for which no editing candidate is accepted, are repurposed as direct-reproduction samples, where the instruction asks the model to reconstruct the reference chart and the ground-truth code generates that reference. The final dataset comprises around 700k image-instruction-code triplets spanning the six functional dimensions of Table 1, covering both direct generation and editing tasks. Detailed per-stage statistics, including the per-model contribution to the cascade and the distribution across editing categories, are reported in Appendix C.

C Dataset Statistic

Table 7: Distribution of chart types in the Chart Edit dataset (by percentage).

Chart Type	Ratio (%)	Chart Type	Ratio (%)
line	39.74	area	4.70
scatters	14.97	graph	3.52
bar	8.70	contour	3.19
heatmap	8.50	hist	3.15
errorbar	6.99	density	2.12
box	1.43	quiver	0.77
scatter	0.66	violin	0.43
radar	0.37	pie	0.28
surface	0.17	other	0.25
errorpoint	0.08		

Table 8: Statistics of visualization code generation models and their respective contribution to the final dataset. The table shows the number of chart-code pairs generated by each model and their proportion in the total dataset (N=521,338).

Model	Number of Charts	Proportion
Qwen3-VL-32B-Instruct	118,577	22.74%
Qwen3-VL-253B-A22B	187,907	36.04%
Gemini 3 Flash	214,854	41.21%
Total	**521,338**	**100%**

Table 9: Distribution of edit types in the Chart Edit dataset (by percentage).

Task Type	Ratio (%)
Axes & Annotations	27.11
Content Manipulation & Data Logic	20.68
Chart Transformation	11.61
Layout & Spatial Organization	10.02
Style & Aesthetics	12.68
direct	17.91

Prompt for Code Generation

Act as an expert data scientist. Convert the provided chart image into high-fidelity Python code (using Matplotlib or Seaborn) that perfectly replicates its visual style, data trends, and layout. You must strictly initialize the plot using `plt.figure(figsize=({width}, {height}))` to match the required dimensions. Ensure all labels, legends, and LaTeX scientific notations are preserved, and provide only the executable code block.

Figure 7: Prompt for code reproduction based on the given image from the arXiv papers.

Prompt for Image-Code Pair Evaluation

You are a data visualization expert and a QA engineer.

Task:
Analyze the provided chart image and its corresponding code. Identify any quality issues that make

Table 10: Pipeline statistics for image-instruction-code triplet construction. Each row reports the count of items remaining after the corresponding stage.

Stage	Count
arXiv papers (2024)	244,029
Papers under CC BY 4.0 license	100,180
Figures extracted from LaTeX sources	854,484
Chart images after classification	622,672
Chart-code pairs after cascade filtering	521,338
Editing samples accepted by audit	567,049
Direct-reproduction samples (no edit accepted)	142,735
Total image-instruction-code triplets	**709,784**

Prompt for Image-Code Pair Evaluation (continued)

this data pair unsuitable for a high-quality machine learning dataset.

Evaluation Criteria:
Readability: Are labels, titles, or legends overlapping or cut off?
Completeness: Are the axes, labels, and legends present and correct?
Visual Integrity: Is the scaling appropriate? Is there any glitching in the rendering?
Consistency: Does the visual representation in the image accurately reflect the logic in the code?

Output Format:
You should first analyze the entire chart, then respond in the following JSON format, evaluating based on the four dimensions mentioned above:

```
{
 "Readability": { "Issues": [ ... ], "Score": int,
            "Status": bool },
 "Completeness": ...,
 "Visual": ...,
 "Consistency": ...
}
```

Among the evaluation of a specific dimension:
- Issues: List specific issues, or an empty array [].
- Score: 1-10, where 10 is production-ready.
- Status: true represents Pass, while false represents Fail.

Figure 8: Prompt for evaluating chart-code pairs during dataset construction. The prompt instructs the MLLM (Multimodal Large Language Model) assessor to check quality across four key dimensions: readability, completeness, visual integrity, and consistency. Each dimension is scored 1-10, with a binary pass/fail status. This quality filtering ensures the dataset contains only high-fidelity image-code pairs.

Prompt for Chart Editing Instruction Generation

You are an expert Data Visualization Engineer and AI Training Specialist. Your task is to generate high-quality "Instruction-to-Code" pairs for fine-tuning a Multimodal LLM.

Task
Analyze the provided image and code, then generate a natural language instruction that either requests a single atomic modification or a composite request involving 2-3 interrelated modifications. The changes must be logically synergistic (e.g., changing the data density while simultaneously adjusting the axis scale to fit the new range).

Prompt for Chart Editing Instruction Generation (continued)

Task-Specific Guidelines:
Choose some task from these categories:
- Content Manipulation & Data Logic: Focus on the underlying narrative of the data. This involves statistical abstraction (e.g., adding mean/median lines), data density control (e.g., sampling high-frequency signals or binning continuous data), or re-filtering data series to emphasize specific trends, outliers, or comparisons.
- Style & Aesthetics: Focus on the 'Data-to-Ink' ratio and aesthetic appeal. Adjust non-data components such as color mapping (sequential, diverging, or categorical), transparency (alpha) for handling overplotting, grid line visibility, background themes (clean vs. decorative), and the refined properties of data markers (edge colors, shapes, and sizes).
- Layout & Spatial Organization: Focus on the topology of the canvas. This includes the strategic placement of legends and color bars to avoid occlusion, adjusting the figure's aspect ratio for better readability, managing subplot spacing (padding/margins), and re-arranging multiple panels into a more logical grid or combined structure.
- Chart Transformation (Structural Change): Focus on changing the fundamental visual metaphor. This involves switching between geometric marks (e.g., from Line to Area charts), re-configuring data alignment (e.g., from Grouped Bar to Stacked or 100% Stacked Bar), or performing coordinate system transformations (e.g., Cartesian to Polar or Linear to Logarithmic).
- Axes & Annotations (Contextual Layers): Focus on precision and semantic guidance. This includes fine-tuning axis limits and tick granularity, adding localized text callouts to explain anomalies, drawing directional arrows to guide the viewer's eye, or highlighting specific threshold regions and 'safe zones' with shaded areas.

Other Requirements:
- **Tone Strategy**: Use a **varied** tone. Synthesize a natural, human-like request.
- **Code Logic**: The modified code must be complete, executable, and preserve the original data structure unless the instruction explicitly asks for a data update.

Output Format (JSON)
1. **Visual & Code Analysis**: Briefly describe the chart and identify the code segment to be modified.
2. **Metadata & Instruction** (in `json`):

```
{
 "edit_categories": [ "Category name" ],
 "instruction": "Modification instruction",
 "change_log": [ "Detailed list of what was modified" ],
}
```

3. **Refined Code** (in `python`):

```
# Fully functional code reflecting the changes
```

Figure 9: Prompt for generating chart editing instructions, which defines the role, inputs, task, domain-specific categories, and structured output format. This prompt is used to guide a multimodal LLM in producing high-quality instruction-code pairs for fine-tuning.

System Prompt for Visual Editing Quality Audit

You are a Senior Data Visualization Auditor and Quality Control Expert for AI training datasets. Your task is to evaluate the quality of a "Chart Edit" pair consisting of an instruction and the resulting visual transformation.

Evaluation Criteria
Please rate the following dimensions on a scale of 1 to 25 (1 = Poor, 25 = Excellent):
1. **Alignment (Faithfulness)**: Does the Modified Image accurately reflect the Edit Instruction?
    - 25: Perfectly executed as requested.
    - 1: The change is unrelated to the instruction or factually wrong.
2. **Integrity (Minimal Disturbance)**: Were non-requested elements preserved?
    - 25: Only the requested parts were changed; all other data and styles remain consistent.
    - 1: Significant, unintended changes occurred (e.g., data points shifted, titles vanished).

System Prompt for Visual Editing Quality Audit (continued)

3. **Linguistic Quality**: Is the instruction natural, clear, and human-like?
   - 25: Sounds like a real user; professional and clear.
   - 1: Robotic, repetitive, or contains logical contradictions.
4. **Visual Legibility**: Is the resulting chart professional and readable?
   - 25: Clear, no overlapping text, appropriate contrast, and professional aesthetics.
   - 1: Broken layout, overlapping elements, or illegible fonts.

Output Format (Strict JSON)
Analyze the visual differences between the two images first, then provide the scores and brief comments in `json` format.

```
{
 "alignment": { "comment": ${comment}, "score": ${subscore} },
 "integrity": { "comment": ${comment}, "score": ${subscore} },
 "linguistic": { "comment": ${comment}, "score": ${subscore} },
 "legibility": { "comment": ${comment}, "score": ${subscore} },
}
```

Figure 10: System prompt for the quality audit of a visual editing process. The prompt defines the auditor's role, required inputs, evaluation criteria, with a detailed 1-25 scoring scale, and a strict JSON output format. The auditor must assess faithfulness, integrity, linguistic quality, and visual legibility of the edit.

C.1 Multi-Agent System Configuration

Table 11 lists the backbone model assigned to each agent in the trajectory synthesis pipeline. The Reasoner and Generator agents share the backbone we use for downstream SFT (Qwen3-VL-8B-Thinking), so that the synthesized reasoning episodes lie within the distribution the student model is best able to internalize during fine-tuning. The remaining three agents serve only as data-construction supervisors and are powered by the stronger Qwen3-VL-32B-Instruct, which provides more reliable planning, critique, and trajectory finalization.

Table 11: Backbone model assignment for each agent in the trajectory synthesis pipeline.

Agent	Symbol	Backbone
Planner	$\mathcal{P}$	Qwen3-VL-32B-Instruct
Reasoner	$\mathcal{T}$	Qwen3-VL-8B-Thinking
Generator	$\mathcal{G}$	Qwen3-VL-8B-Thinking
Critic	$\mathcal{C}$	Qwen3-VL-32B-Instruct
Finalizer	$\mathcal{F}$	Qwen3-VL-32B-Instruct

D SFT Trajectory Synthesis Analysis

D.1 Trajectory Component Ablation

The planner and critic are complementary. Keeping both modules yields the strongest pass@k and the lowest token cost, while removing either module hurts trajectory quality. In particular, the planner is most important for reaching high pass@k, whereas the critic mainly helps the trajectory recover from unproductive reasoning.

D.2 Synthesis Cost

The synthesis pipeline is compute-intensive but manageable with two coordinated machines. The planner, critic, and finalizer run on A800-class hardware, while the reasoner and generator occupy H100-class GPUs, and the scorer dominates the throughput budget.

Table 12: Ablation of the Reason-Score-Refine synthesis pipeline. We evaluate whether the planner and critic are both needed for effective trajectory construction.

Planner	Critic	Pass@3 ↑	Pass@6 ↑	Pass@9 ↑	Avg. Token ↓
✓	✓	**24.75%**	35.17%	**42.58%**	**2086.33**
	✓	24.00%	**35.75%**	**42.58%**	2416.21
✓		16.58%	22.00%	24.75%	2200.22
		15.33%	19.75%	23.42%	2300.86

Table 13: Estimated throughput and compute footprint of the SFT synthesis pipeline.

Module	Hardware	Throughput
Planner	1×A800	3500/h
Reasoner	1×H100	4000/h
Generator	1×H100	4000/h
Scorer	64 Core GPU + 512GB Memory	9000/h
Critic	1×A800	18000/h
Finalizer	1×A800	9000/h
Total	Running on two machines (8×A800+64C CPU + 512GB Mem; 8×H100+64C CPU + 512GB Mem)	About 2 weeks (380 hours)

D.3 SFT Filtering Statistics

Table 14: Statistics of the SFT data synthesis pipeline after filtering.

Task Type	Total Input	Filtered	Final Output	Pass Rate	Avg. Round Num
Edit Task	562,485	383,867	178,618	31.76%	5.39
Direct Task	75,275	45,957	29,318	38.95%	5.42
Total	**637,760**	**429,824**	**207,936**	**32.60%**	**5.40**

These statistics show that the editing portion is harder to retain through the cascade, while direct-reproduction samples have a slightly higher pass rate. Across both splits, the synthesis loop averages about 5.4 rounds before producing an accepted trajectory.

D.4 Effect of SFT Data Scale

The full SFT set gives the strongest downstream results on both benchmarks. Performance improves steadily as more synthesized trajectories are added, indicating that the data scaling effect is still positive even after 150k samples, although the gains become smaller at the largest scale.

E Experimental Details

E.1 Training Configuration

Tables 16 and 17 summarize the SFT and RL settings, respectively, as specified by our launch scripts. For SFT, the effective batch size is $8 \times 2 \times 16 = 256$ sequences, accounting for the number of GPUs, per-device batch size, and gradient accumulation. The value of `max_samples` is a loading cap, rather than the number of training trajectories retained after filtering. The RL configuration uses the defaults of the 8B wrapper together with the base launcher it invokes.

During RL training, we uniformly sample an integer from 1 to 10 and multiply it by the 512-token chunk size, giving a thinking budget in $\{512, 1024, \ldots, 5120\}$ tokens. Each final-answer generation is capped at 4,096 tokens. The 8,192-token response limit applies to the concatenation of reasoning,

Table 15: Effect of SFT dataset size on downstream performance.

SFT Dataset Size	ChartEdit w/o Code		ChartMIMIC Direct	
	Exec. Rate	Overall	Exec. Rate	Overall
base	79.15	67.30	75.00	58.45
50k	83.56	68.76	79.06	61.75
100k	84.48	69.12	80.72	63.58
150k	86.98	71.01	82.11	65.05
full	**87.76**	**73.43**	**84.06**	**67.78**

Table 16: SFT training configuration specified by the launch script.

Parameter	**Value**
Framework	LLaMA-Factory
Initialization	Qwen3-VL-8B-Thinking
Number of GPUs	8
Training epochs	1
Learning rate	5×10^{-5}
Training batch size	256 sequences
Per-device batch size	2
Gradient accumulation steps	16
Sequence cutoff (tokens)	16,384
Sample cap (`max_samples`)	210,000
Chat template	`qwen3_vl`
Checkpoint interval	100 steps
Validation	Disabled

the injected stop-thinking prompt, and the final answer; longer training responses are truncated to this limit. The injected prompt is masked out of the policy loss. KL regularization is applied as an actor loss using the `low_var_kl` estimator, rather than added to the reward. Online validation is performed every five training steps, with validation before training disabled.

E.2 Evaluation Configuration

Following prior work (Wang et al., 2023; Shi et al., 2024), we adopt the following decoding configurations for all evaluations. For open-source models, we set the temperature to 0.1 and top-p to 1. For proprietary models, we set the temperature to 0 and top-p to 1, with a maximum output length of 4096 tokens. For all open-source reasoning models, we set the thinking budget to 4096 tokens and the maximum output length for final code generation to 4096 tokens; the implementation of thinking budget control follows the protocol described in Appendix A.2. All open-source models are served via the vLLM inference library on NVIDIA A800 80GB GPUs.

E.3 Benchmark Descriptions

ChartEdit (Zhao et al., 2025a). ChartEdit is a chart editing benchmark that requires models to modify an existing chart according to a natural language instruction, without access to the original source code. Each sample provides a reference chart image and an editing instruction, and the model is expected to generate executable visualization code that reflects the requested changes. Performance is assessed across three complementary dimensions: code-level correctness, which measures the syntactic and structural validity of the generated code; chart-level fidelity, which evaluates the visual alignment between the generated and target charts; and execution-level success, which checks whether the code runs without errors. An official overall score aggregates these dimensions into a single metric.

ChartMIMIC (Yang et al., 2025a). ChartMIMIC evaluates models on two distinct tasks. The Customized Mimic task requires instruction-conditioned chart editing, where the model must adapt

Table 17: RL training configuration. Values include the 8B wrapper's overrides of the base launcher.

Parameter	**Value**
Framework	verl
Initialization	SFT checkpoint
Number of GPUs	8 (one node)
Training epochs	1
Learning rate	1×10^{-6}
Training batch size	256 prompts
Checkpoint interval	5 steps
Validation interval	5 steps
Optimization and budgeted rollout	
Advantage estimator	GRPO
PPO mini-batch size	128 prompts
Rollouts per prompt	8
Rollout temperature	1.0
KL loss coefficient	0.01
Entropy coefficient	0
Thinking chunk size (tokens)	512
Number of thinking chunks	Uniform in $\{1, \ldots, 10\}$
Final-answer limit (tokens)	4,096
Prompt / response limits (tokens)	4,096 / 8,192
Execution settings	
Actor strategy / rollout backend	FSDP2 / vLLM
Rollout tensor parallelism	2
Rollout context limit (tokens)	12,288
Actor token limit per GPU	24,576 (dynamic batching)
Image pixel range	3,136–1,048,576
Gradient checkpointing	Enabled
Fused kernels	Disabled

a reference chart according to a given instruction; samples are divided into Low and High difficulty tiers based on the complexity of the required edits. The Direct Mimic task measures chart-to-code reproduction ability without any editing instruction, providing a complementary assessment of the model's direct chart generation capability. Both tasks report execution success rate and an overall score computed from visual and structural similarity between the generated and reference charts.

F Training Corpus Examples

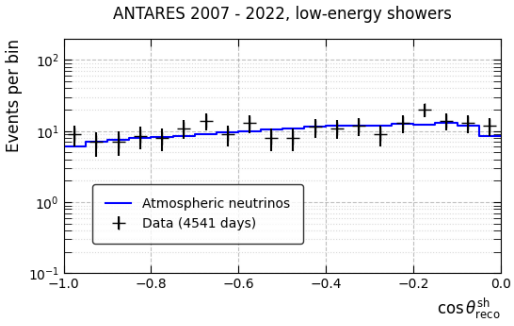
Figures 11–16 present 28 selected training examples covering the six categories in Table 1. Figure 11 shows eight input charts for direct reproduction, where the task is to generate code that recreates the given image. The remaining figures show four editing examples per category, pairing each source chart (left) with its target chart (right) and an excerpt of the original instruction. These targets are corpus supervision images, not predictions from *REChart*. Shared prompt boilerplate is omitted, and ellipses mark omitted instruction text. These examples were selected for clear instruction–target correspondence and task diversity, rather than to estimate corpus-wide quality. Images retain their original content and aspect ratios; only exterior blank margins are trimmed for compact display.

Direct Reproduction

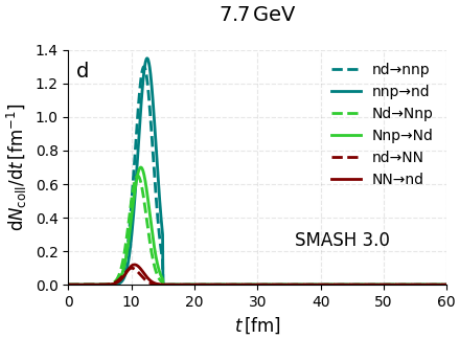
Input charts only

Task: Generate executable Matplotlib code to reproduce each input chart.

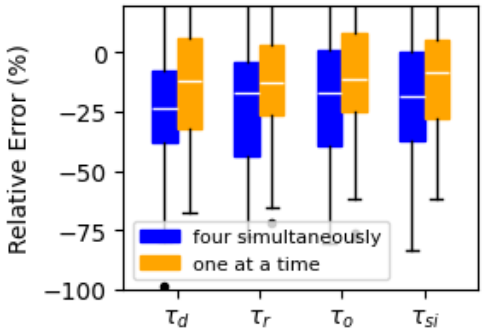
a Histogram and error bars



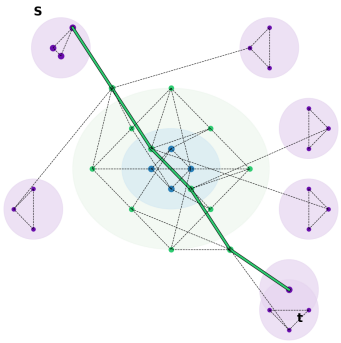
b Reaction-rate curves



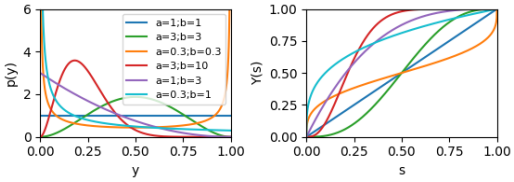
c Grouped box plots



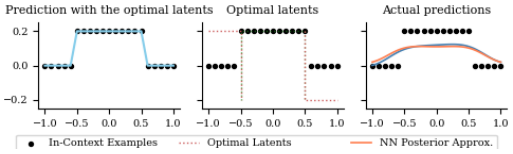
d Network topology



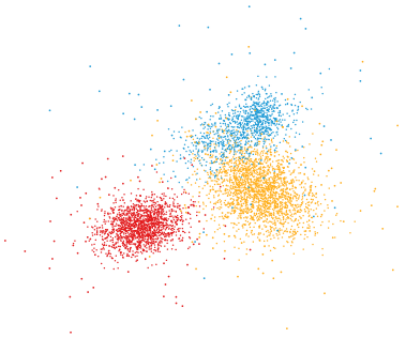
e Probability density and CDF



f Step-function approximation



g Clustered scatter plot



h Multi-panel comparison

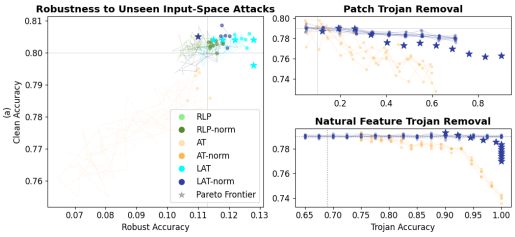
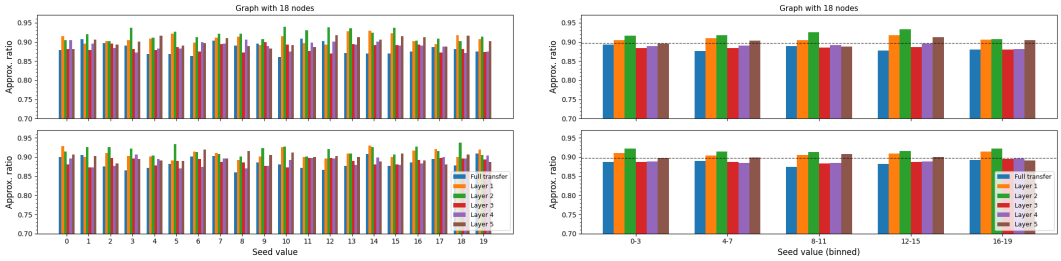


Figure 11: Eight input examples for *Direct Reproduction*. The task is to generate executable code that recreates each chart. Each image is shown once because this task requires no visual edits.

Content Manipulation

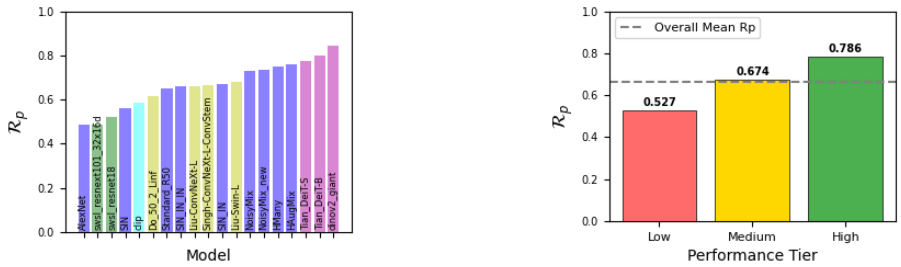
Source chart → Target chart

a Seed-wise aggregation



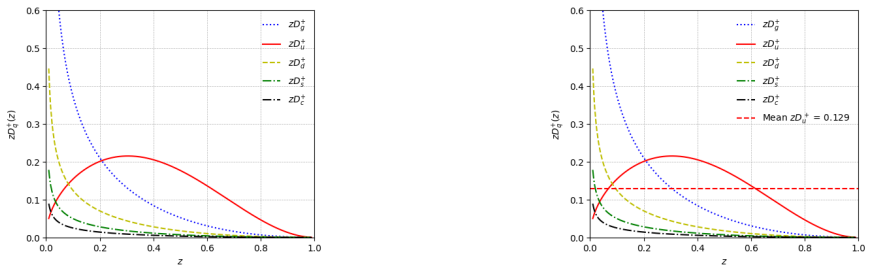
Instruction: binning the seed values into 5 groups (0-3, 4-7, 8-11, 12-15, 16-19), then compute the mean approx. ratio for each group and category. ... add a horizontal line at the overall mean approx. ratio across all data points

b Performance-tier averages



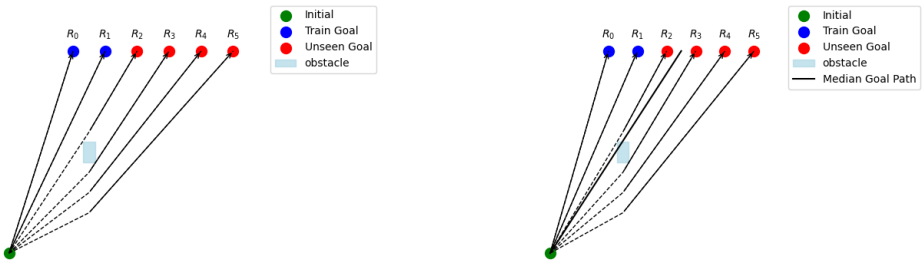
Instruction: binning the models into three performance tiers (Low: Rp < 0.6, Medium: 0.6 ≤ Rp < 0.75, High: Rp ≥ 0.75) ... add a horizontal line at the overall mean Rp

c Mean reference line



Instruction: Add a horizontal line at the mean value of zD_u^+ across the z range to emphasize the central tendency of the most prominent distribution.

d Median goal trajectory



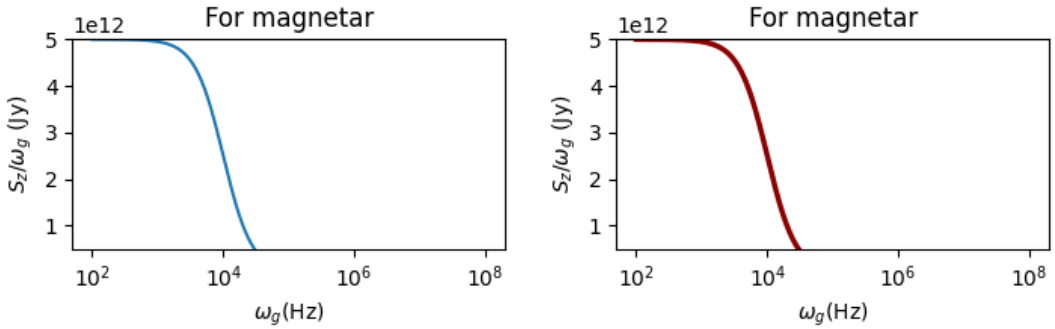
Instruction: Add a median trajectory line that connects the initial position to the median of all goal positions, using a solid black line with a label 'Median Goal Path'.

Figure 12: Training corpus examples for *Content Manipulation*: seed-wise aggregation, performance-tier averaging, a mean reference line, and a median goal trajectory.

Style & Aesthetics

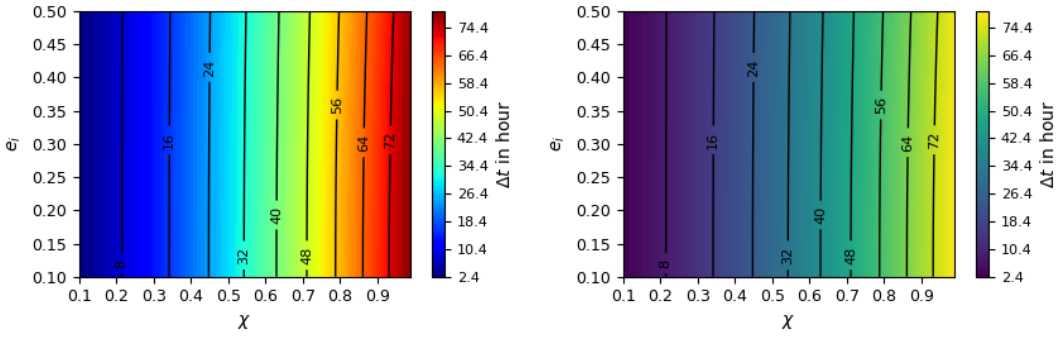
Source chart → Target chart

a Line emphasis



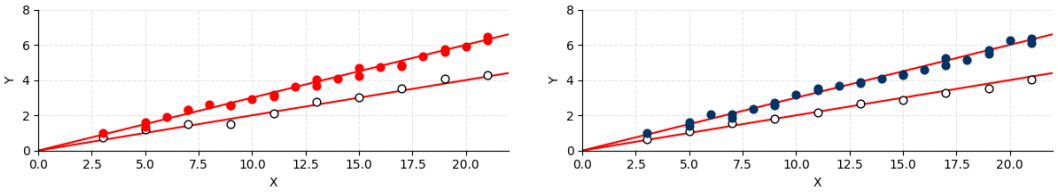
Instruction: Change the line color to dark red and increase the line width to 2.5 for better visual prominence.

b Perceptual colormap



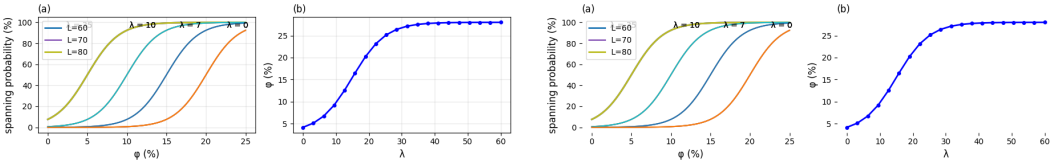
Instruction: Replace the 'jet' colormap with 'viridis' ... maintaining the same contour levels and structure.

c Marker recoloring



Instruction: Change the color of the solid red circles to a dark blue (#003366) ... distinguish them more clearly from the red trend lines.

d Clean axes and background



Instruction: removing grid lines from both subplots and applying a clean white background with no borders or spines

Figure 13: Training corpus examples for *Style & Aesthetics*: line emphasis, colormap replacement, marker recoloring, and grid/spine removal.

Layout Edits

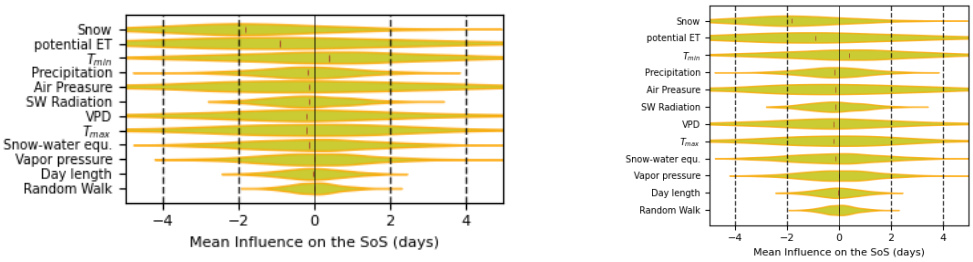
Source chart → Target chart

a Legend relocation



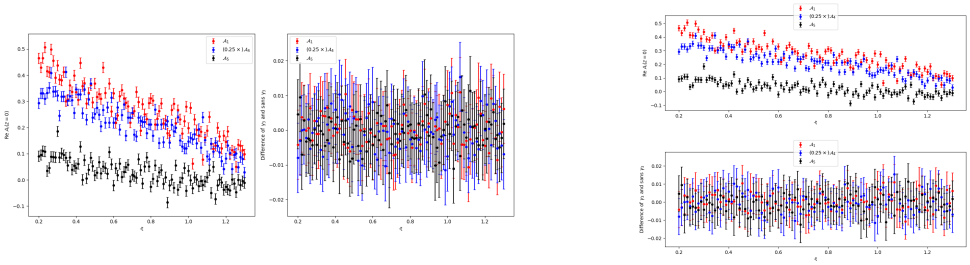
Instruction: Move the legend to the lower right corner of the plot to prevent it from overlapping with the data curves and improve readability.

b Violin spacing



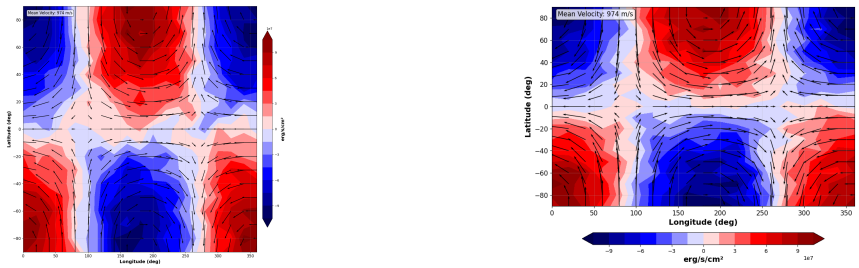
Instruction: Increase the vertical spacing between the violin plots and adjust the figure height to prevent label overlap and improve readability.

c Subplot reorganization



Instruction: Reorganize the subplot layout to a 2x1 grid ... adjust the figure height to 8.0 ... reposition the legends to the top of each subplot

d Colorbar and aspect ratio



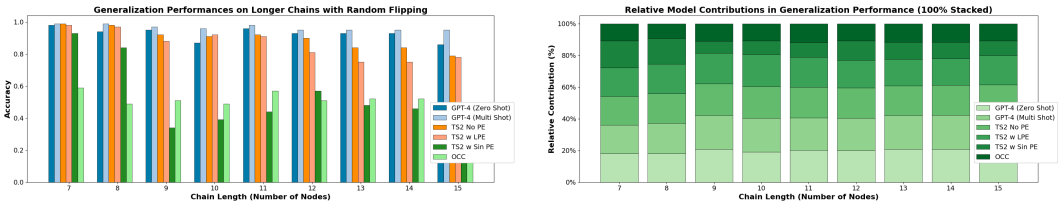
Instruction: reposition the colorbar to the bottom of the plot, adjust the figure's aspect ratio to be more square-like ... increase the spacing between the plot and the colorbar

Figure 14: Training corpus examples for *Layout Edits*: legend relocation, violin spacing, subplot reorganization, and colorbar/aspect-ratio adjustment.

Chart Transformation

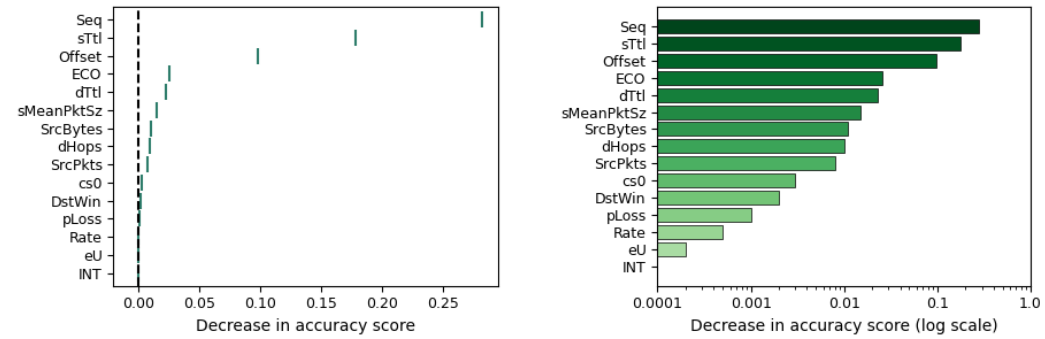
Source chart → Target chart

a Normalized stacked bars



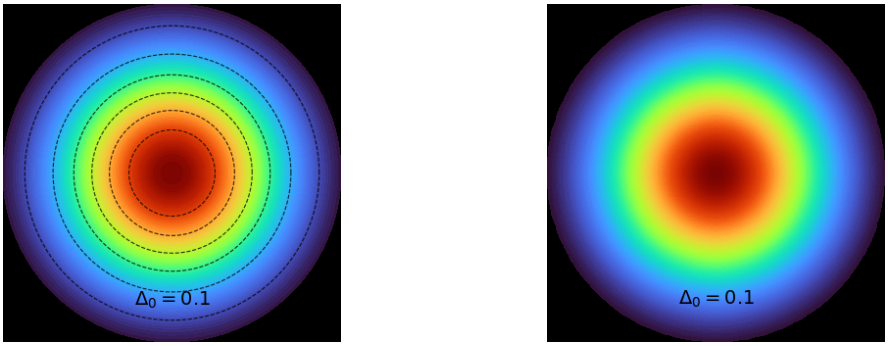
Instruction: Transform the current grouped bar chart into a 100% stacked bar chart ... switch the y-axis to a percentage scale (0% to 100%) ... adjust the color palette to use a sequential gradient

b Log-scale horizontal bars



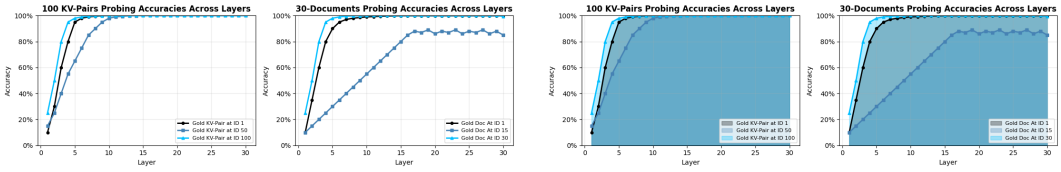
Instruction: Transform the chart from a vertical tick plot to a horizontal bar chart with logarithmic x-axis scaling ... adjust the color scheme to use a gradient from light to dark green

c Polar heatmap



Instruction: Transform the current polar contour plot into a polar heatmap by switching from contourf to pcolormesh in polar coordinates, using the same radial intensity function and color map.

d Line-to-area conversion



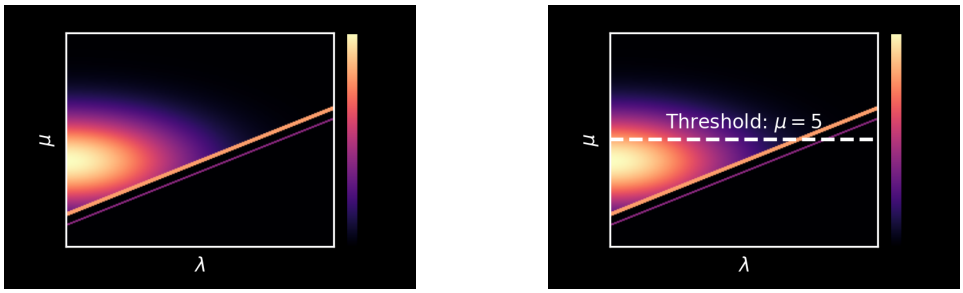
Instruction: Transform the line charts in both subplots into area charts by filling the area under each line with a semi-transparent color corresponding to the line's color.

Figure 15: Training corpus examples for *Chart Transformation*: normalized stacked bars, log-scale horizontal bars, a polar heatmap, and line-to-area conversion.

Axes & Annotations

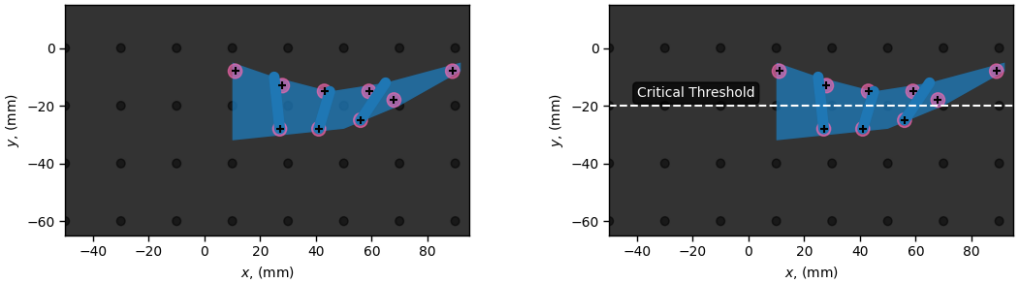
Source chart → Target chart

a Threshold annotation



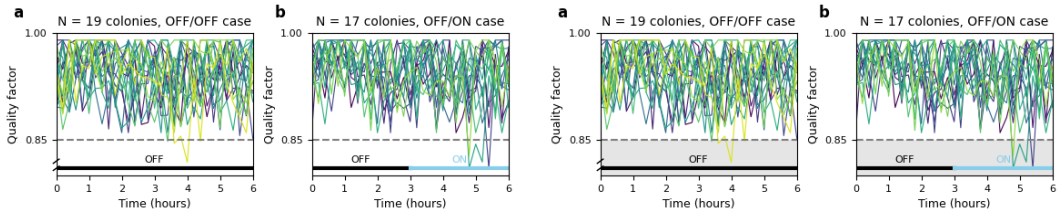
Instruction: Add a white dashed horizontal line at y=5 ... annotate it with the text 'Threshold: μ=5' positioned slightly above the line

b Spatial reference line



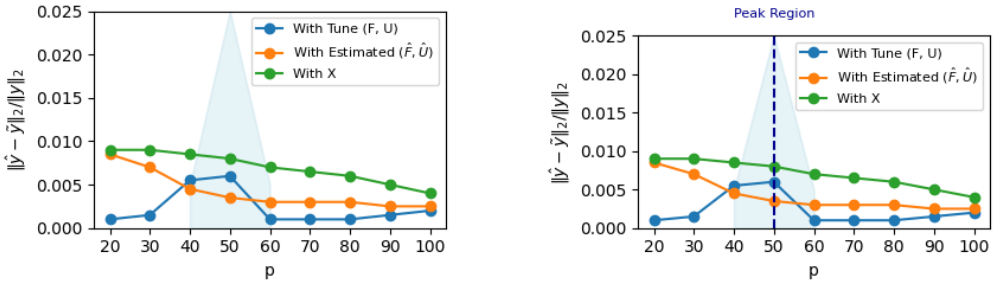
Instruction: Add a horizontal dashed line at y = -20 mm ... annotate it with the text 'Critical Threshold' positioned slightly above the line, aligned to the left of the plot.

c Low-quality region



Instruction: Add a shaded region below the 0.85 threshold line in both panels ... using a semi-transparent gray color

d Vertical reference



Instruction: Add a vertical dashed line at p=50 ... annotate it with the text 'Peak Region' positioned above the line, aligned to the center, with a font size of 8 and color 'darkblue'.

Figure 16: Training corpus examples for *Axes & Annotations*: labeled thresholds, a shaded low-quality region, and a vertical reference line.