LEARNING PROGRAMMATIC CHART EDITING THROUGH PRIVILEGED ON-POLICY SELF-DISTILLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Chart editing requires a multimodal large language model (MLLM) to translate a reference chart and a natural-language instruction into executable plotting code. Existing approaches often rely on powerful proprietary models to synthesize reasoning data for supervised fine-tuning (SFT) and evaluate rollouts during group relative policy optimization (GRPO), making supervision costly to scale. To reduce this reliance, we introduce a two-stage fine-tuning framework that combines SFT on verified reasoning and code pairs with on-policy self-distillation (OPSD), where a frozen copy of the fine-tuned model receives privileged information (PI) to supervise a student model given only the original input. First, for SFT data construction, we design a reasoning trajectory generation pipeline guided by privileged information (PI). Specifically, we annotate the reference chart to obtain a structured textual description and provide it, together with the reference chart and editing instruction, to a low-cost MLLM of comparable capability to Qwen3.5-9B (the base model) to obtain the reasoning trajectory. This yields a training set of 266k verified trajectories, which, after SFT of the base model, brings improvements of over 10% on both ChartEdit and ChartMIMIC. Next, after SFT, OPSD further improves the model through dense token-level feedback, without requiring a separate larger teacher or repeated calls to an external reward evaluator during optimization. We conduct experiments to systematically study how the choice of PI and SFT initialization affect OPSD. On ChartEdit and ChartMIMIC, our model achieves relative improvements of 17% and 32%, respectively, over the base model and delivers the best overall performance among the open-source models of comparable size. Our comparisons favor precise PI aligned with the desired output, with target code outperforming both reference information and combinations containing additional context. Direct OPSD degrades performance and exhibits substantial PI leakage, whereas SFT initialization reduces the leakage rate from over 40% to approximately 2% and enables further performance gains.

1 INTRODUCTION

Programmatic chart editing requires a multimodal large language model (MLLM) to generate executable plotting code from a reference chart image and a natural-language editing instruction, without access to the original source code (Zhao et al., 2025a; Chen et al., 2026b). The model needs to infer the chart's content and visual structure from the image, apply the requested edit, and leave other chart properties unchanged. Existing approaches obtain supervision for this task through synthetic demonstrations and execution-based feedback. Some works rely on supervised fine-tuning (SFT), which constructs reliable examples by generating chart programs and, in some cases, intermediate reasoning or coding trajectories, followed by execution and visual verification (Zhao et al., 2025b; Jia et al., 2025; Tang et al., 2026). Subsequent studies have explored group relative policy optimization (GRPO) (Shao et al., 2024), which introduces an online cost by executing and rendering multiple sampled programs and scoring their outputs with external visual evaluators (Chen et al., 2026a;b; 2025; Tang et al., 2026). For example, MSRL and RRVF use 72B-parameter evaluators to score training rollouts (Chen et al., 2026a; 2025).

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.

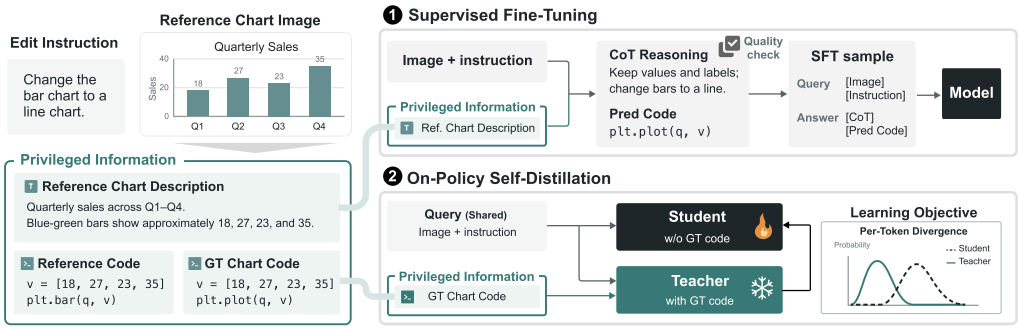


Figure 1: Overview of our two-stage chart-editing framework: description-guided SFT followed by privileged on-policy self-distillation.

However, these approaches rely on extensive supervision that requires powerful proprietary models, making supervision costly to scale. To reduce the supervision costs, training-time chart descriptions and programs offer a promising alternative by making chart content and plotting operations explicit to a generator or teacher. Figure 1 illustrates this opportunity with a bar-to-line edit: a reference description summarizes the visible chart, while target code specifies the requested transformation. However, such privileged information (PI) can also introduce dependence on details unavailable in the student's image-and-instruction input (Yang et al., 2026; Kaur et al., 2026; Shrestha & Tessier, 2026). Therefore, there is a need to study how PI representation, target alignment, and SFT initialization affect the effectiveness of privileged supervision.

We introduce a two-stage framework (Figure 1) to address this need. First, we leverage structured reference descriptions to guide a low-cost generator in producing reasoning and code for SFT. The description separates observed, inferred, and uncertain properties to reduce the trajectory generator's reliance on details available only in the source code. We separately verify reasoning for visual grounding and code consistency, and programs for executability, instruction compliance, and visual agreement with the target. We use the resulting 266k examples to fine-tune another model.

Next, we apply on-policy self-distillation (OPSD) using a frozen copy of the SFT model as the teacher (Zhao et al., 2026; Penaloza et al., 2026). The teacher receives PI and supplies next-token distributions on student-generated prefixes; the student sees only the image and instruction throughout training and inference. This provides dense token-level feedback without a separate larger teacher and avoids online execution, rendering, and external reward evaluation during optimization.

Our experiments show that, after SFT, target code outperforms reference descriptions and reference code, while adding either reference input weakens its benefits. Direct OPSD degrades base-model performance despite reducing teacher–student divergence and induces substantial leakage with code-based PI. SFT initialization substantially mitigates this leakage and enables further gains. The resulting 9B model achieves the highest overall scores among the comparable open-weight models we evaluate on ChartEdit and both ChartMIMIC tasks. For ChartMIMIC Customized, OPSD improves overall score from 69.81 after SFT to 76.40, a 9.4% improvement. In a second-stage comparison of 100 optimization updates on four H100 GPUs, GRPO takes 5.77 times as long as OPSD and additionally incurs external-judge API costs.

Our contributions are:

- A description-guided pipeline that uses a low-cost MLLM and separate reasoning and program verification to construct 266k training examples.
- A two-stage framework combining SFT and privileged OPSD to improve a 9B chart-editing model without external reward evaluation during policy optimization.
- An analysis of PI representation, target alignment, and SFT initialization, showing that reduced distillation divergence alone does not ensure task improvement.

2 Related Work

Visual Coding LLMs. Visual coding LLMs generate programs from visual specifications, supporting webpage reconstruction (Si et al., 2025), chart reproduction (Wu et al., 2025; Yang et al., 2025), and editing (Zhao et al., 2025a; Li et al., 2026b; Kapadnis et al., 2026). Web2Code and VisCodex develop large multimodal coding corpora (Yun et al., 2024; Jiang et al., 2026), while ChartCoder and ChartReasoner construct intermediate reasoning supervision (Zhao et al., 2025b; Jia et al., 2025). Such supervision requires high-quality trajectories whose visual content, reasoning, and code agree. Producing these trajectories is difficult and costly, as synthesis requires advanced generators and checkers. ChartReasoner uses DeepSeek-R1 with code-based validation, and MM-ReCoder curates multi-turn coding demonstrations (Jia et al., 2025; Tang et al., 2026). GRPO is also widely used to improve visual coding through rewards derived from execution and rendering feedback (Chen et al., 2026a;b; 2025; Tang et al., 2026). Although structural rewards can be rule-based (He et al., 2026), visual assessment often requires a strong external evaluator (Chen et al., 2026a; 2025). Repeated scoring adds to the cost of high-quality supervision. Our framework addresses both costs through verified, description-assisted reasoning synthesis with a low-cost MLLM and subsequent OPSD without a separate larger teacher or external reward evaluator.

On-Policy Self-Distillation. OPSD derives token-level supervision from the same model conditioned on training-only privileged information (PI) (Zhao et al., 2026; Penaloza et al., 2026; Agarwal et al., 2024). The information supplied to this self-teacher varies substantially across tasks. OmniOPSD uses generated rationales for affective computing (Cheng et al., 2026), Vision-OPD focuses on informative image crops, while Video-OPSD and Clue-OPSD select evidence frames or temporal intervals (Yuan et al., 2026; Wang et al., 2026b;a). Visual-OPSD supplies visual reasoning traces, OPD-V uses modality-balance signals, and PTD-PO combines spatial guidance with intermediate textual hints (Li et al., 2026a; Aniri et al., 2026; Xiang et al., 2026). These approaches show that PI can change either the available evidence or its representation. However, recent studies report PI leakage, degradation with richer privileged contexts, and gains that depend on the reference form and student generation mode (Yang et al., 2026; Kaur et al., 2026; Shrestha & Tessier, 2026; Zhang et al., 2026). Chart editing brings together visual observations, natural-language instructions, and executable programs, making PI selection a cross-representation problem. A reference description makes visual content explicit, reference code specifies its implementation, and target code additionally encodes the requested transformation. We therefore compare PI by representation precision and target alignment, including combinations of these sources, and examine when SFT initialization enables this information to improve editing without inducing privileged-context dependence.

3 Method

3.1 Task formulation

Let I_{ref} denote a reference chart image and u an editing instruction. The student input is $x = (I_{\text{ref}}, u)$. A model with parameters θ generates a complete plotting program

$$\hat{c} \sim p_\theta(c \mid x), \qquad \hat{I} = \mathcal{R}(\hat{c}), \tag{1}$$

where $\mathcal{R}$ executes the program and renders the chart. Given a target program c^* and its rendering $I^* = \mathcal{R}(c^*)$, the objective is to generate executable code that implements u and matches the target chart. Chart descriptions and programs can provide supervision during training, but the student receives only x at both training and inference. Figure 1 shows how we use this additional information to construct SFT targets and guide subsequent OPSD.

3.2 Supervised fine tuning

3.2.1 Corpus construction

We collected chart images from the compiled LaTeX sources of arXiv papers submitted in 2024 under CC BY 4.0. For each image, an MLLM generated a reference program c_{ref}, an editing or reconstruction instruction u, and a target program c^*. For reconstruction, $c^* = c_{\text{ref}}$. For editing, c^* implements the requested change. The resulting corpus contains 665,451 tuples $\mathcal{D}_{\text{pair}} = \{(I_{\text{ref}}, c_{\text{ref}}, u, c^*, I^*)\}$. Appendix A.1.1 describes source processing and pair construction.

3.2.2 Reasoning synthesis and verification

Additional chart information can help a generator recover details that are difficult to infer from pixels alone. However, the resulting reasoning must remain suitable for a student that receives only x. Reference plotting programs can contain quantities that are not uniquely recoverable from the rendered image (Niu et al., 2026), while providing code directly can yield explanations that assume access to the supplied program. Figure 2 shows a trace that cites reference code to justify an exact figure size. The final program may be correct even when its explanation relies on context absent from the student's input.

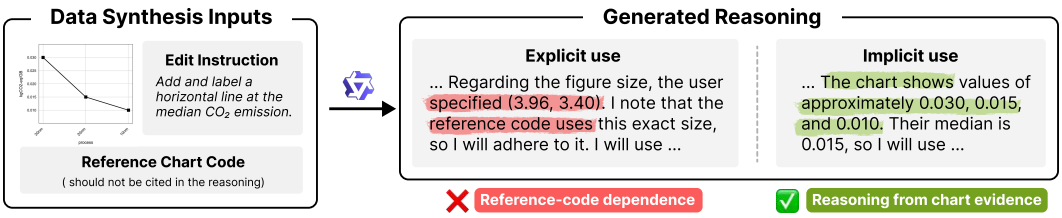


Figure 2: Additional information leaking into synthesized reasoning. The red highlight marks explicit reliance on reference code to justify an exact figure size. The green highlight expresses reasoning in terms of chart evidence. The student receives only the chart image and instruction.

We use a structured reference description to assist generation while limiting this dependence. Given the reference image and program, a description generator produces

$$d_{\text{ref}} \sim p_{\text{desc}}(\cdot \mid I_{\text{ref}}, c_{\text{ref}}). \tag{2}$$

The prompt distinguishes directly observable properties, inferences supported by visual evidence, and uncertain properties. Visible text and legend placement are observations, whereas values estimated from graphical marks are inferences. We instruct the generator to use reference code only to check or disambiguate visible content and to exclude hidden implementation details from the description. This constraint aims to keep the additional information grounded in the image.

The trajectory generator then produces reasoning r and a program c from the original input and description,

$$(r, c) \sim p_{\text{gen}}(\cdot \mid x, d_{\text{ref}}). \tag{3}$$

The description assists synthesis but is excluded from the student's input. We instruct the generator to explain how visual evidence supports the requested edit and its implementation, without referring to the additional description.

We check the reasoning and program separately. The reasoning judge assesses instruction understanding, visual grounding, reasoning quality, and agreement between r and c. These checks address errors that successful execution alone would miss. For the program, we require successful execution and assess compliance with u using the reference program. We also compare $\mathcal{R}(c)$ with I^* in chart content and presentation. Only examples passing both reasoning and program quality filters are retained. Appendices A.1.3 and A.1.6 provide the scoring rules, thresholds, and prompts.

Qwen3.7 Flash performs description and trajectory generation as well as automated quality assessment. Its API cost and task performance relative to the student backbone are reported in Appendix A.1.4. The retained dataset contains approximately 266k examples with input x and target (r, c). SFT on these examples adapts the base model, Qwen3.5-9B, to chart editing and reproduction (Qwen Team, 2026a). We denote the base and SFT parameters by θ_{base} and θ_{SFT}, respectively.

3.3 On policy self distillation

3.3.1 Privileged information

We sample 26k examples from the portion of $\mathcal{D}_{\text{pair}}$ not selected for SFT, without synthesized reasoning targets. Each example provides the original input x, reference description d_{ref}, reference program c_{ref}, and target program c^*. We use these materials as candidate PI for the teacher, denoted by z, while keeping the student's input restricted to x.

The reference description expresses chart content in natural language. Reference code provides a programmatic representation with explicit data and plotting operations. Comparing them examines how representation and the precision of available details affect supervision. Since descriptions and programs differ in both form and information coverage, this comparison reflects the combined effect of these two factors rather than representation alone.

Reference and target programs share a code representation but serve different purposes for editing. Reference code specifies the original chart, leaving the teacher to infer the required modification. Target code specifies the intended result and supplies an implementation of the edit. Comparing them tests the value of guidance aligned with the requested output. This distinction applies to editing examples, since reference and target code coincide for direct reconstruction. We also combine target code with either the reference description or reference code to test whether additional reference context improves supervision. The five PI settings are

$$z \in \{d_{\mathrm{ref}}, c_{\mathrm{ref}}, c^*, (d_{\mathrm{ref}}, c^*), (c_{\mathrm{ref}}, c^*)\}\,. \tag{4}$$

Programs can reveal exact values or implementation choices that the image does not uniquely determine. Their connection to the same chart therefore does not make all teacher information available to the student.

3.3.2 Self-distillation and initialization

We apply OPSD to obtain supervision from a copy of the student model conditioned on PI (Zhao et al., 2026; Penaloza et al., 2026). Let θ_0 denote their common initialization. The main method uses $\theta_0 = \theta_{\mathrm{SFT}}$. For the direct OPSD comparison, both teacher and student start from $\theta_0 = \theta_{\mathrm{base}}$. The teacher remains fixed at θ_0 within each condition, while only the student parameters θ are updated. This comparison examines the effect of SFT initialization on the complete supervision process.

The student samples $\hat{c} \sim p_\theta(\cdot \mid x)$. At position t, the student and teacher evaluate the same generated prefix,

$$p_S^t = p_\theta(\cdot \mid x, \hat{c}_{<t}), \qquad p_T^t = p_{\theta_0}(\cdot \mid x, z, \hat{c}_{<t}). \tag{5}$$

The teacher supplies a distribution over the next token at each prefix visited by the student. We minimize the divergence along the student's own generated programs,

$$\mathcal{L}_{\mathrm{OPSD}}(\theta) = \mathbb{E}_{(x,z),\,\hat{c}\sim p_\theta(\cdot|x)}\left[\sum_t D_{\mathrm{JSD}}(p_T^t, p_S^t)\right]. \tag{6}$$

We use symmetric Jensen Shannon divergence (JSD), with $m^t = (p_T^t + p_S^t)/2$,

$$D_{\mathrm{JSD}}(p_T^t, p_S^t) = \tfrac{1}{2}\,\mathrm{KL}(p_T^t \| m^t) + \tfrac{1}{2}\,\mathrm{KL}(p_S^t \| m^t). \tag{7}$$

Appendix A.2.2 gives implementation details and the alternative objectives evaluated in Section 4.5.2. OPSD requires no online program execution, chart rendering, or external reward evaluation in our implementation. At inference, the teacher and PI are removed.

We assess the usefulness of PI through complementary observations. Teacher task scores with PI indicate whether the additional context helps the teacher solve the problem. Divergence measures distribution matching, while student task scores measure performance using only x. We also examine leakage, understood here as outputs that assume access to descriptions or programs absent from the student's input. Agreement with the target program alone does not establish such dependence. Section 4.5.3 reports these diagnostics, and Appendix A.2.4 specifies the generation and divergence evaluation protocols.

4 Experiments

4.1 Training details

We fine-tuned Qwen3.5-9B with ms-swift in both stages. Full-parameter SFT used approximately 266k examples on four NVIDIA A800 GPUs for 32 hours. OPSD used 26k examples on four NVIDIA H100 GPUs, updating the student with LoRA while freezing the teacher. Both stages used an effective batch size of 256. Appendix A provides the remaining settings.

4.2 BENCHMARKS AND BASELINES

We evaluated chart editing on ChartEdit (Zhao et al., 2025a) and reproduction and customized editing on ChartMIMIC (Yang et al., 2025). Both benchmarks report execution rate; ChartEdit additionally measures Code Level and Chart Level, while ChartMIMIC measures Low-Level and High-Level similarity. Overall is the mean of the two non-execution metrics. Table 1 compares proprietary GPT (OpenAI, a;b;c) and Gemini models (Google DeepMind, 2026a;b), general-purpose open-source MLLMs with up to 106B parameters (Bai et al., 2025b;a; Qwen Team, 2026a;b; Team et al., 2026; Wang et al., 2025), and chart-specific models, including ChartCoder (Zhao et al., 2025b), MSRL (Chen et al., 2026a), RRVF (Chen et al., 2025), and ChartEditor (Chen et al., 2026b).

4.3 MAIN RESULTS

Table 1: Main comparison on ChartEdit and ChartMIMIC.

Model	ChartEdit			ChartMIMIC Direct			ChartMIMIC Customized		
	Exec.	Code	Chart	Exec.	Low	High	Exec.	Low	High
Proprietary MLLMs									
GPT-5.6 Terra	98.9	89.3	94.7	**98.5**	**78.2**	**96.2**	**98.0**	83.2	**91.8**
GPT-5.6 Luna	98.2	88.3	93.9	96.7	75.1	93.5	**98.0**	82.1	91.4
GPT-5	**99.1**	84.4	92.4	94.2	78.0	88.0	93.8	79.6	85.9
Gemini 3.8 Flash	98.3	**89.3**	**95.0**	71.0	58.1	67.2	84.5	73.8	78.1
Gemini 3.1 Pro	98.8	86.1	94.2	78.8	66.8	73.9	95.5	**84.7**	90.7
Open Sourced MLLMs									
Qwen2.5-VL-72B	89.8	57.7	71.0	-	-	-	85.3	67.6	69.1
Qwen2.5-VL-32B	88.4	61.7	72.5	-	-	-	85.6	66.4	69.5
Qwen3-VL-32B-Thinking	85.5	72.9	76.3	82.8	65.2	72.2	84.0	67.0	72.3
Qwen3.5-9B	78.1	75.0	69.1	71.9	56.0	61.3	74.7	59.8	62.9
Qwen3.7-Flash	81.7	67.2	67.4	82.5	63.6	67.9	80.2	65.5	69.8
InternVL3.5-38B	85.1	62.2	71.6	77.2	54.5	59.0	83.0	59.8	65.0
InternVL3.5-8B	74.7	58.0	61.2	69.2	46.3	48.5	72.2	49.0	52.9
GLM-4.6v	91.4	74.2	83.2	84.5	69.0	75.8	84.7	66.2	70.6
GLM-4.6v-Flash	86.1	69.4	77.0	80.2	64.7	68.8	72.8	52.5	54.3
ChartCoder	89.8	47.3	74.5	91.4	72.5	74.0	85.2	58.3	62.0
MSRL	**97.2**	57.6	84.8	96.5	**78.6**	83.8	67.1	39.6	46.5
RRVF	96.1	44.1	74.2	**97.8**	60.9	67.9	85.9	57.6	59.3
ChartEditor	66.5	40.2	55.3	69.6	52.7	56.4	61.6	52.1	57.9
Ours	94.9	**80.5**	**88.0**	93.4	**78.6**	**85.7**	**89.3**	**74.4**	**78.4**

Our 9B model led the evaluated general-purpose open-source MLLMs on all nine metrics (Table 1). Including chart-specific systems, it was best or tied for best on seven metrics; MSRL and RRVF retained higher execution rates on ChartEdit and Direct, respectively. Over Qwen3.5-9B, High Level improved by 24.4 points on Direct and 15.5 points on Customized. On Direct, our model also exceeded Gemini 3.8 Flash and Gemini 3.1 Pro on all three metrics.

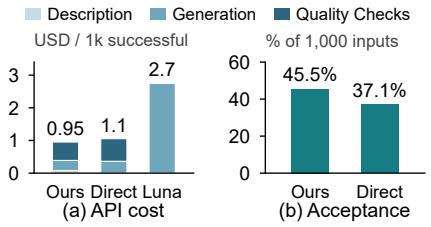


Figure 3: SFT reasoning data synthesis cost and acceptance. Costs include failed attempts. Luna denotes GPT5.6 Luna, which omits quality checks.

4.4 SFT ABLATION

We tested using different types of data as guidance during SFT data synthesis. Figure 4 shows that reference descriptions produced substantially less leakage than

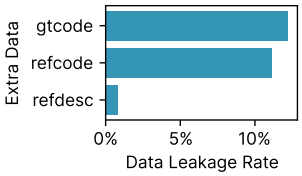


Figure 4: Data leakage after SFT CoT data construction with different extra data.

Table 2: SFT data ablation. Performance of the base model and models fine-tuned with or without CoT across ChartEdit and ChartMIMIC benchmarks and evaluation metrics.

	ChartEdit			ChartMIMIC Direct			ChartMIMIC Customized		
Training	Exec.	Code	Chart	Exec.	Low	High	Exec.	Low	High
Base	78.15	74.96	69.13	71.94	56.04	61.26	74.72	59.75	62.90
w/o CoT	**94.59**	78.49	**88.02**	89.72	75.84	82.54	47.33	37.05	38.96
w/ CoT	94.31	**79.50**	87.80	**91.50**	**76.12**	**84.45**	**82.72**	**68.32**	**71.31**

reference or target code, demonstrating that using descriptions of reference images can indeed mitigate the data leakage problem.

We also compared our method against the direct generation method without additional guidance and reverse synthesis from target code using an advanced model. Figure 3a shows our method's advantage in API cost, with 9.7% reduction of direct generation cost, which is even below GPT5.6 Luna's reverse generation cost. Figure 3b shows that our method improves the acceptance rate, enabling more efficient use of the available data.

SFT with CoT improved every reported metric over the base model (Table 2). Code-only SFT improved ChartEdit and Direct but degraded all three Customized metrics, including a 27.39-point decrease in execution rate. With CoT, Customized execution instead exceeded the base model by 8.00 points. Thus, the generated reasoning supported improvements across all evaluated tasks and avoided the Customized degradation observed with code-only training.

4.5 OPSD ABLATION

4.5.1 TRAINING COST

We compared 100 OPSD and GRPO updates on four H100 GPUs (Table 3). GRPO took 20 hours 34 minutes, versus 3 hours 33 minutes 46 seconds for OPSD, a 5.77-fold difference in wall time. The GRPO run used four rollouts per prompt and Qwen3.7-Flash for remote reward evaluation. Its judge API calls added USD 18.99 over 100 steps. OPSD used a local teacher and incurred no reward-judge API charge during this stage. The cost gap therefore persists with a small GRPO rollout count and a low-cost judge.

Table 3: Per-update cost over 100 steps.

	OPSD	GRPO
Wall time/step (m:s)	2:08.3	12:20.4
Judge API/step (USD)	0	0.1899

4.5.2 DISTILLATION OBJECTIVE AND TOP-K LOGITS

We compared objectives at $K = 64$ and logit budgets under JSD, using target-code PI after SFT.

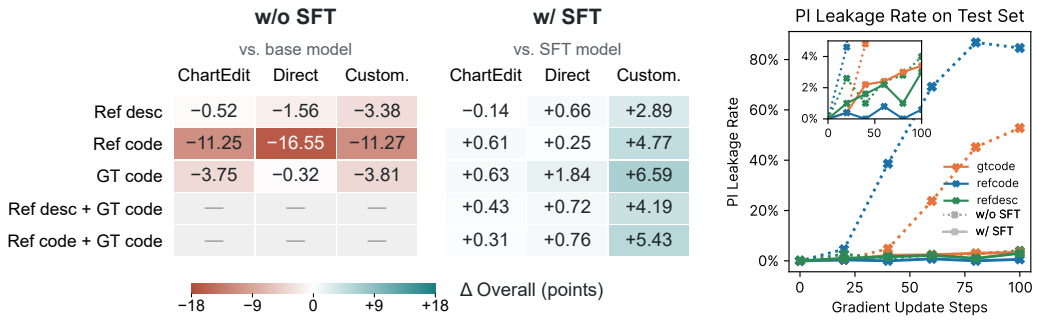
Table 4: Distillation objective ablation. All values are Overall scores. Bold marks the best score in each column.

Objective	ChartEdit	ChartMIMIC	
		Direct	Customized
Forward KL	**84.41**	80.69	72.90
Reverse KL	83.92	80.97	73.80
JSD	84.28	**82.12**	**76.40**

Table 5: Top-K logit ablation. All values are Overall scores. Bold marks the best score in each column.

Top-K	ChartEdit	ChartMIMIC	
		Direct	Customized
32	83.91	81.43	73.60
64	84.28	**82.12**	**76.40**
128	**84.70**	81.27	74.89

JSD led both ChartMIMIC tasks, exceeding forward and reverse KL by 3.50 and 2.60 points on Customized; forward KL led ChartEdit by 0.13 points (Table 4). $K = 64$ also led ChartMIMIC, while $K = 128$ improved ChartEdit by 0.42 points but reduced Customized by 1.51 points (Table 5).



(a) Overall changes after OPSD.

(b) PI leakage during OPSD.

Figure 5: Performance and privileged-information (PI) leakage during OPSD. (a) Changes in Overall (points) relative to the corresponding base or SFT checkpoint. "Direct" and "Custom" denote the two ChartMIMIC sets. (b) Under direct OPSD, leakage from code-based PI grows with the number of gradient-update steps, whereas after SFT, leakage remains low across all PI types.

These comparisons support JSD with $K = 64$ as our default, with task-dependent trade-offs, consistent with the objective and logit-budget ablations in Vision-OPD and Visual-OPSD (Yuan et al., 2026; Li et al., 2026a).

4.5.3 Privileged information and SFT initialization

We interpret student performance and leakage (Figure 5) alongside held-out on-policy JSD and the task scores of checkpoints prompted with PI (Tables 6 and 7). The prompting diagnostic uses benchmark subsets and no OPSD update, with the details given in Appendix A.2.4.

Table 6: Teacher–student JSD ($\times 10^{-3}$) before and after OPSD. Lower values indicate closer token distributions.

PI	Base		SFT	
	Before	After	Before	After
ref desc	11.22	5.06	7.37	5.94
ref code	17.55	11.04	8.49	6.74
gt code	16.47	11.05	8.33	6.66
ref code + gt code	–	–	10.77	8.69
ref desc + gt code	–	–	11.93	9.78

Table 7: PI prompting diagnostic. Each model generates with the indicated PI.

PI	Base model			SFT model		
	CE	Dir.	Cust.	CE	Dir.	Cust.
None	64.21	34.24	56.55	83.86	80.49	67.58
ref desc	64.28	34.10	55.65	84.58	80.77	69.11
ref code	64.50	33.46	58.12	84.56	81.46	69.23
gt code	63.77	33.46	57.90	84.47	81.46	71.26
ref code + gt code	–	–	–	84.32	81.46	67.66
ref desc + gt code	–	–	–	84.55	81.35	70.40

CE: ChartEdit; Dir./Cust.: ChartMIMIC Direct/Customized. Dashes indicate unreported conditions. Evaluation protocols are given in Appendix A.2.4.

After SFT, reference code improved Customized Overall over descriptions by 1.88 points, although it reduced Direct by 0.41 points. Target code outperformed both reference forms on all three tasks and also gave the strongest SFT-teacher scores on average across all benchmarks and sets, tying reference code on Direct. Descriptions primarily support chart perception, code additionally specifies plotting operations, and target code specifies the intended edit. The teacher and student results thus favor precise PI aligned with the output, with task-dependent benefits from reference code.

Direct OPSD degraded base-model performance despite reducing JSD in every tested condition. Both code-based PI forms also lowered the base teacher's scores on all three tasks, while descriptions helped only marginally on chart edit. After SFT, all three single-source PI forms improved teacher scores, and student leakage remained low. Together, these observations suggest that SFT makes PI more useful for task supervision, and that reducing distribution mismatch alone is insufficient. This qualifies the distribution-gap interpretation of Visual-OPSD (Li et al., 2026a) and is consistent with RLSD's account of information asymmetry (Yang et al., 2026). SFT enhances transfer in this

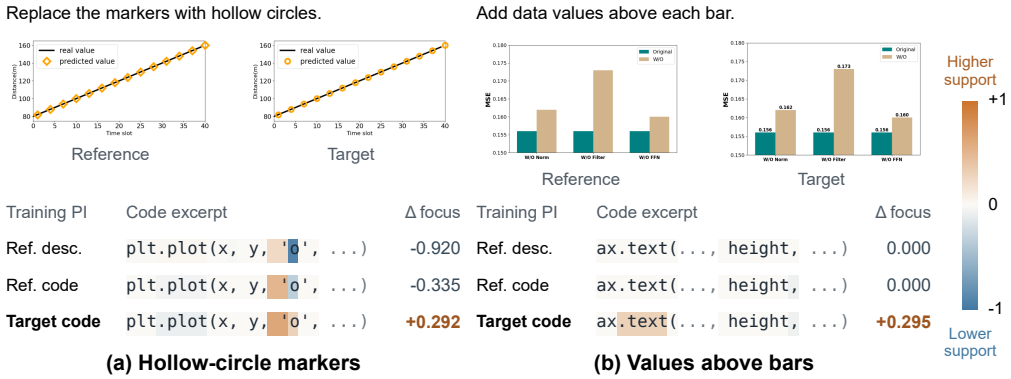


Figure 6: Token-level changes under different PI training conditions. (a) Replacing markers with hollow circles; (b) adding values above bars. Reference and target charts are shown above the corresponding code excerpts. Colours encode Δ_z; the right column reports its value for the focus token (`o` or `.text`). Ellipses omit code from the display only.

asymmetric setting, but it does not eliminate the teacher's access to information the student cannot obtain.

Adding reference information to target code lowered student performance on every task without improving teacher scores. Both combinations also had higher on-policy JSD before and after OPSD, and with descriptions, post-OPSD loss was 46.8% higher than with target code alone. Thus, extra context increased the mismatch on student rollouts without improving teacher guidance.

4.5.4 Token-level qualitative analysis

Figure 6 visualizes the token-level log-probability change $\Delta_z = \log p_{\text{OPSD-}z} - \log p_{\text{SFT}}$ for SFT-initialized variants (z denotes the training PI) under the same generated responses as prefixes. Target-code PI strengthens support for code choices aligned with the requested edits. In these examples, reference-based PI either suppresses or barely changes the relevant marker (`o`) and annotation (`.text`) tokens, whereas target-code PI increases their probability. This contrast suggests that the target program supplies guidance more directly aligned with edit implementation than reference information alone.

5 Conclusion

We presented a framework for programmatic chart editing that combines SFT on verified reasoning and code with OPSD. Reference descriptions guide an affordable generator to produce 266k training examples, while a frozen copy of the SFT model provides privileged supervision during subsequent optimization. This approach improves chart editing and reproduction with lower measured synthesis and optimization costs in our comparisons, without requiring external reward evaluation during OPSD. The resulting 9B model achieves the highest Overall scores among the evaluated open-sourced models of comparable size on ChartEdit and both ChartMIMIC tasks. Our PI comparisons favor target code, with additional reference information reducing its benefits. SFT initialization substantially mitigates PI leakage and enables further gains, while direct OPSD reduces distribution divergence without improving task performance. These findings support careful PI selection and show that successful distribution matching alone does not guarantee better chart editing.

Limitations. Our experiments are subject to several constraints. Due to computational limits, we restricted training to Qwen3.5-9B and did not evaluate our dataset or training procedure on other model families or sizes. Larger models may interpret chart descriptions and programs more effectively and thus obtain greater gains from OPSD. Our scope is limited to chart generation and editing. Extending the approach to webpage and geometric diagram would help assess its applicability to other forms of visual code generation. Finally, our budget constrained the scale of verified SFT data construction, so we have not built a larger dataset to determine how performance scales beyond the current 266k examples or where the benefits of SFT begin to saturate.

AI USE STATEMENT

We used Qwen3.7 Flash to generate reference chart descriptions, reasoning trajectories, and plotting programs, and to assess generated reasoning, code, and rendered charts during dataset construction. This model also provided reward scores for the GRPO comparison. The generation and evaluation prompts, filtering procedures, and model settings are described in Appendix A. OpenAI Codex was used to assist with polishing the writing. The execution checks and automated assessments used to filter synthetic data are described in Appendix A.1.3. The authors take responsibility for the study design, experimental evidence, and final text, code, figures, and conclusions, including material produced with generative AI assistance.

ETHICS STATEMENT

The source corpus is derived from arXiv papers submitted in 2024 under the CC BY 4.0 license, as described in Appendix A.1.1. Any redistribution of derived materials should retain source attribution and respect applicable licenses and notices for third party content. Generated programs are executed in isolated environments with time limits to reduce execution risks. Synthetic reasoning and automated judgments can contain errors, and generated charts may distort data values, trends, or uncertainty. The corpus reflects the disciplines and visualization practices of its source publications, so the reported results should not be interpreted as evidence of reliability across all chart domains.

REPRODUCIBILITY STATEMENT

Sections 3 and 4 describe the task, data construction, OPSD objective, and experimental comparisons. Appendix A.1.1 documents the source corpus and dataset sizes, while Appendix A.1.3 specifies the generation settings and filtering criteria. The complete generation and evaluation prompts are provided in Appendix A.1.6. Tables 11, 12, and 13 report the training configurations for SFT, OPSD, and the GRPO comparison. Appendix A.2.4 specifies the evaluation subsets, fixed teacher checkpoints, and procedure for measuring divergence on student responses. The reported estimates are based on individual runs, and Section 5 discusses the remaining limitations.

REFERENCES

Rishabh Agarwal, Nino Vieillard, Yongchao Zhou, Piotr Stanczyk, Sabela Ramos Garea, Matthieu Geist, and Olivier Bachem. On-policy distillation of language models: Learning from self-generated mistakes. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=3zKtaqxLhW.

Aniri, Jinhe Bi, Peng Liao, Zengjie Jin, Volker Tresp, Fei Shen, Yunpu Ma, and Tat-Seng Chua. OPD-V: Visual on-policy self-distillation with modality balance, 2026. URL https://arxiv.org/abs/2608.05131.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-vl technical report, 2025a. URL https://arxiv.org/abs/2511.21631.

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, 2025b. URL https://arxiv.org/abs/2502.13923.

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 *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. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(24):20199–20207, 2026b. doi: 10.1609/aaai.v40i24.39107. URL https://ojs.aaai.org/index.php/AAAI/article/view/39107.

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, 2025. URL https://arxiv.org/abs/2507.20766.

Zebang Cheng, Shuimu Chen, Boxue Yang, Yuanshen Guan, Jingyi Chen, Zheng Lian, Xiaojiang Peng, Fei Ma, Laizhong Cui, and Qi Tian. OmniOPSD: Rationale-privileged on-policy self-distillation for affective computing, 2026. URL https://arxiv.org/abs/2606.15920.

Google DeepMind. Gemini 3.1 Pro. Model card, February 2026a. URL https://deepmind.google/models/model-cards/gemini-3-1-pro/. Accessed: 2026-09-26.

Google DeepMind. Gemini 3.8 Flash. Model card, September 2026b. URL https://deepmind.google/models/model-cards/gemini-3-8-flash/. Accessed: 2026-09-26.

Minggui He, Mingchen Dai, Jian Zhang, Yilun Liu, Shimin Tao, Pufan Zeng, Osamu Yoshie, and Yuya Ieiri. Chart specification: Structural representations for incentivizing vlm reasoning in chart-to-code generation. *Neurocomputing*, 702:134706, 2026. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2026.134706. URL https://www.sciencedirect.com/science/article/pii/S0925231226021041.

Caijun Jia, Nan Xu, Jingxuan Wei, Qingli Wang, Lei Wang, Bihui Yu, and Junnan Zhu. ChartReasoner: Code-driven modality bridging for long-chain reasoning in chart question answering, 2025. URL https://arxiv.org/abs/2506.10116.

Lingjie Jiang, Shaohan Huang, Xun Wu, Yixia Li, Guanhua Chen, Dongdong Zhang, and Furu Wei. Viscodex: Unified multimodal code generation via merging vision and coding models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=hUXzPauNEM.

Manav Nitin Kapadnis, Lawanya Baghel, Atharva Naik, and Carolyn Rosé. ChartEditBench: Evaluating grounded multi-turn chart editing in multimodal language models, 2026. URL https://arxiv.org/abs/2602.15758.

Simran Kaur, Narutatsu Ri, Yinghui He, Liam Fowl, and Sanjeev Arora. Rethinking on-policy self-distillation for thinking models, 2026. URL https://arxiv.org/abs/2607.05184.

Pengyu Li, Zhitao Gao, Lingling Zhang, Muye Huang, Yuanming Li, Zesheng Yang, Fangzhi Xu, and Jun Liu. Visual-OPSD: Cross-modal on-policy self-distillation for efficient unified multimodal reasoning, 2026a. URL https://arxiv.org/abs/2606.18974.

Shuo Li, Jiajun Sun, Zhekai Wang, Xiaoran Fan, Hui Li, Dingwen Yang, Zhiheng Xi, Yijun Wang, Zifei Shan, Tao Gui, Qi Zhang, and Xuanjing Huang. Charte$ˆ{3}$: A comprehensive benchmark for end-to-end chart editing. In *Forty-third International Conference on Machine Learning*, 2026b. URL https://openreview.net/forum?id=7tAon7qQQX.

Tianhao Niu, Qingfu Zhu, and Wanxiang Che. What you see is what you get: Observation-aligned supervision for chart-to-code generation, 2026. URL https://arxiv.org/abs/2607.04726.

OpenAI. GPT-5. OpenAI API documentation, a. URL https://developers.openai.com/api/docs/models/gpt-5. Accessed: 2026-09-26.

OpenAI. GPT-5.6 luna. OpenAI API documentation, b. URL https://developers.openai.com/api/docs/models/gpt-5.6-luna. Accessed: 2026-09-26.

OpenAI. GPT-5.6 terra. OpenAI API documentation, c. URL https://developers.openai.com/api/docs/models/gpt-5.6-terra. Accessed: 2026-09-26.

Emiliano Penaloza, Dheeraj Vattikonda, Nicolas Gontier, Alexandre Lacoste, Laurent Charlin, and Massimo Caccia. Privileged information distillation for language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=ebZcMQImhG.

Qwen Team. Qwen3.5-9B. Hugging Face model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-9B. Accessed September 2, 2026.

Qwen Team. Qwen3.7: The agent frontier, May 2026b. URL https://qwen.ai/blog?id=qwen3.7.

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.

Samyak Shrestha and Alexander Tessier. Rethinking privileged information in on-policy self-distillation, 2026. URL https://arxiv.org/abs/2608.18271.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3956–3974, Albuquerque, New Mexico, April 2025. Association for Computational Linguistics. doi: 10.18653/v1/2025.naacl-long.199. URL https://aclanthology.org/2025.naacl-long.199/.

Zitian Tang, Xu Zhang, Jianbo Yuan, Yang Zou, Varad Gunjal, Songyao Jiang, and Davide Modolo. Mm-recoder: Advancing chart-to-code generation with reinforcement learning and self-correction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22164–22173, June 2026.

V Team, Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, Baoxu Wang, Bin Chen, Boyan Shi, Changyu Pang, Chenhui Zhang, Da Yin, Fan Yang, Guoqing Chen, Haochen Li, Jiale Zhu, Jiali Chen, Jiaxing Xu, Jiazheng Xu, Jing Chen, Jinghao Lin, Jinhao Chen, Jinjiang Wang, Junjie Chen, Leqi Lei, Letian Gong, Leyi Pan, Mingdao Liu, Mingde Xu, Mingzhi Zhang, Qinkai Zheng, Ruiliang Lyu, Shangqin Tu, Sheng Yang, Shengbiao Meng, Shi Zhong, Shiyu Huang, Shuyuan Zhao, Siyan Xue, Tianshu Zhang, Tianwei Luo, Tianxiang Hao, Tianyu Tong, Wei Jia, Wenkai Li, Xiao Liu, Xiaohan Zhang, Xin Lyu, Xinyu Zhang, Xinyue Fan, Xuancheng Huang, Yadong Xue, Yanfeng Wang, Yanling Wang, Yanzi Wang, Yifan An, Yifan Du, Yiheng Huang, Yilin Niu, Yiming Shi, Yu Wang, Yuan Wang, Yuanchang Yue, Yuchen Li, Yusen Liu, Yutao Zhang, Yuting Wang, Yuxuan Zhang, Zhao Xue, Zhengxiao Du, Zhenyu Hou, Zihan Wang, Peng Zhang, Debing Liu, Bin Xu, Juanzi Li, Minlie Huang, Yuxiao Dong, and Jie Tang. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning, 2026. URL https://arxiv.org/abs/2507.01006.

Kaishen Wang, Dongdi Zhao, Yijun Liang, Dingqiang Ye, Ruibo Chen, Heng Huang, and Di Fu. Where to look matters: On-policy self-distillation for long-video understanding, 2026a. URL https://arxiv.org/abs/2608.25356.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, Zhaokai Wang, Zhe Chen, Hongjie Zhang, Ganlin Yang, Haomin Wang, Qi Wei, Jinhui Yin, Wenhao Li, Erfei Cui, Guanzhou Chen, Zichen Ding, Changyao Tian, Zhenyu Wu, Jingjing Xie, Zehao Li, Bowen Yang, Yuchen Duan, Xuehui Wang,

Zhi Hou, Haoran Hao, Tianyi Zhang, Songze Li, Xiangyu Zhao, Haodong Duan, Nianchen Deng, Bin Fu, Yinan He, Yi Wang, Conghui He, Botian Shi, Junjun He, Yingtong Xiong, Han Lv, Lijun Wu, Wenqi Shao, Kaipeng Zhang, Huipeng Deng, Biqing Qi, Jiaye Ge, Qipeng Guo, Wenwei Zhang, Songyang Zhang, Maosong Cao, Junyao Lin, Kexian Tang, Jianfei Gao, Haian Huang, Yuzhe Gu, Chengqi Lyu, Huanze Tang, Rui Wang, Haijun Lv, Wanli Ouyang, Limin Wang, Min Dou, Xizhou Zhu, Tong Lu, Dahua Lin, Jifeng Dai, Weijie Su, Bowen Zhou, Kai Chen, Yu Qiao, Wenhai Wang, and Gen Luo. Internvl3.5: Advancing open-source multimodal models in versatility, reasoning, and efficiency, 2025. URL https://arxiv.org/abs/2508.18265.

Ziyue Wang, Shiqi Huang, Weiwen Xu, Bihan Wen, and Xudong Jiang. Video-OPSD: Exploiting privileged visual evidence for on-policy self-distillation in video large language models, 2026b. URL https://arxiv.org/abs/2608.27065.

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/.

Shizhe Xiang, Ke An, Wenlong Yu, Yue Liu, Jian Luan, Pei Fu, and Qilong Wang. Teaching the way, not the answer: Privileged tutoring distillation for multimodal policy optimization, 2026. URL https://arxiv.org/abs/2606.07000.

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 *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sGpCzsfd1K.

Chenxu Yang, Chuanyu Qin, Qingyi Si, Minghui Chen, Naibin Gu, Dingyu Yao, Zheng Lin, Weiping Wang, Jiaqi Wang, and Nan Duan. Self-distilled RLVR, 2026. URL https://arxiv.org/abs/2604.03128.

Qianhao Yuan, Jie Lou, Xing Yu, Hongyu Lin, Le Sun, Xianpei Han, and Yaojie Lu. Vision-OPD: Learning to see fine details for multimodal LLMs via on-policy self-distillation, 2026. URL https://arxiv.org/abs/2605.18740.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: a large-scale webpage-to-code dataset and evaluation framework for multimodal llms. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Xiuyu Zhang, Wei Chow, Junfeng Fang, Xingyu Zhu, Zhenkai Liang, and Tat-Seng Chua. What does privileged information add to on-policy self-distillation?, 2026. URL https://arxiv.org/abs/2609.20612.

Siyan Zhao, Zhihui Xie, Mengchen Liu, Jing Huang, Guan Pang, Feiyu Chen, and Aditya Grover. Self-distilled reasoner: On-policy self-distillation for large language models. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Jpxfof0EaS.

Xuanle Zhao, Xuexin Liu, Haoyue Yang, 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 *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 3616–3630, Vienna, Austria, July 2025a. Association for Computational Linguistics. 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 *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. doi: 10.18653/v1/2025.acl-long.363. URL https://aclanthology.org/2025.acl-long.363/.

A Implementation Details

A.1 Supervised fine-tuning

A.1.1 Source corpus and dataset size

We collected LaTeX source packages for arXiv submissions from 2024 released under CC BY 4.0. The sources were compiled in isolated environments, and chart images were selected using their surrounding LaTeX structure and visual content. For each image, an MLLM generated a reference Matplotlib program, a reconstruction or editing instruction, and the corresponding target program. Samples with program-execution failures or missing renderings were removed. For reconstruction, the reference and target programs are identical; for editing, the target program implements the instruction. Table 8 gives the corpus size before reasoning synthesis and after synthesis and verification.

Table 8: Dataset size before reasoning-data construction and after synthesis and verification.

Stage	Examples
Before reasoning synthesis	665,451
After synthesis and verification	266,454

A.1.2 REPRESENTATIVE CORPUS EXAMPLES

Figure 7 illustrates the range of visual forms in the reference charts. Figure 8 presents eleven reference–target pairs grouped by editing operation, with condensed instructions. The cases were selected from a sampled subset of the chart–instruction–code corpus for visual variety and clarity.

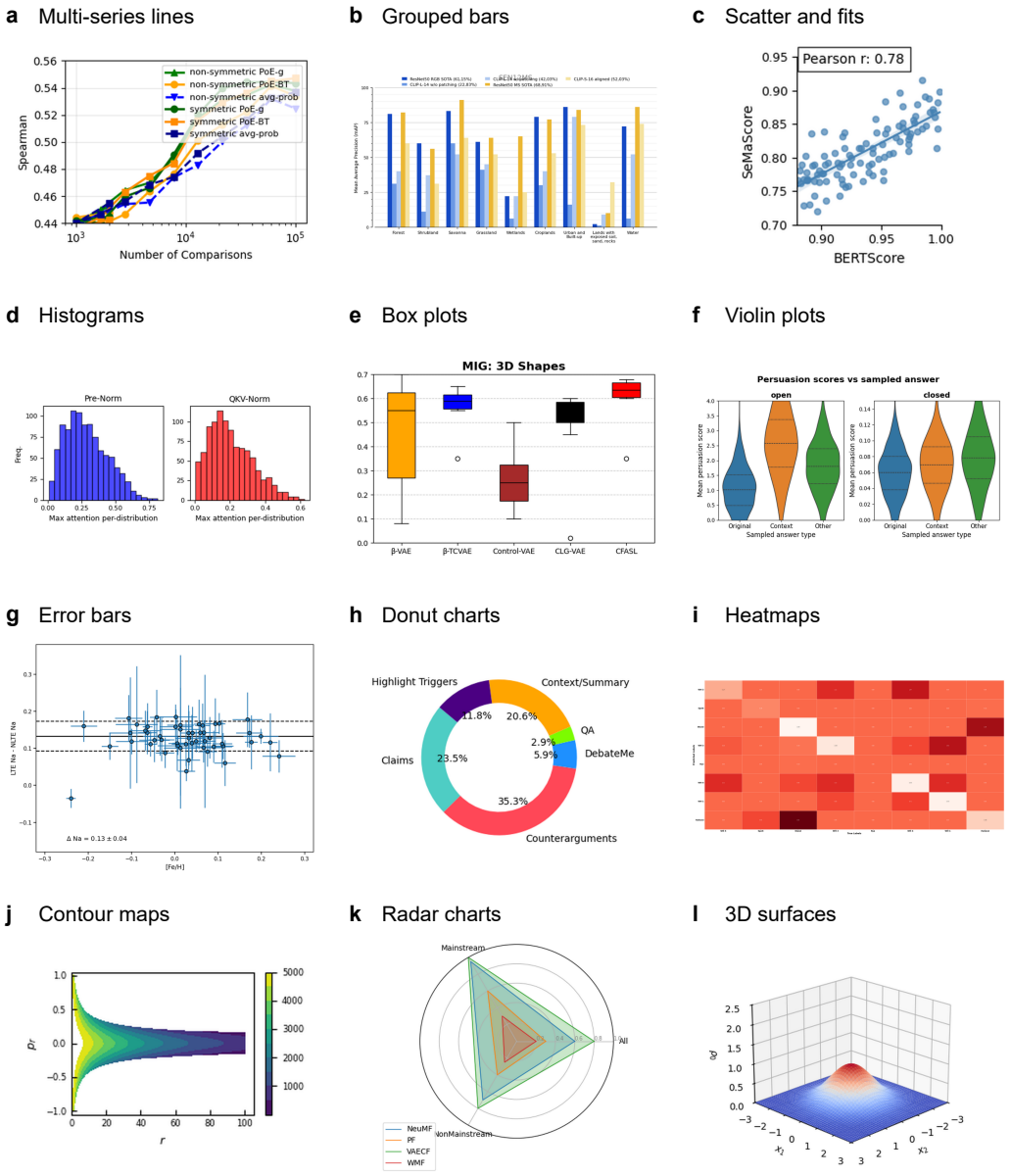


Figure 7: Selected reference charts illustrating twelve visual forms in the corpus, including different marks, coordinate systems, and uncertainty displays. These are illustrative examples; the selection does not indicate the frequency of each form in the full corpus. Images retain their original content and aspect ratios.

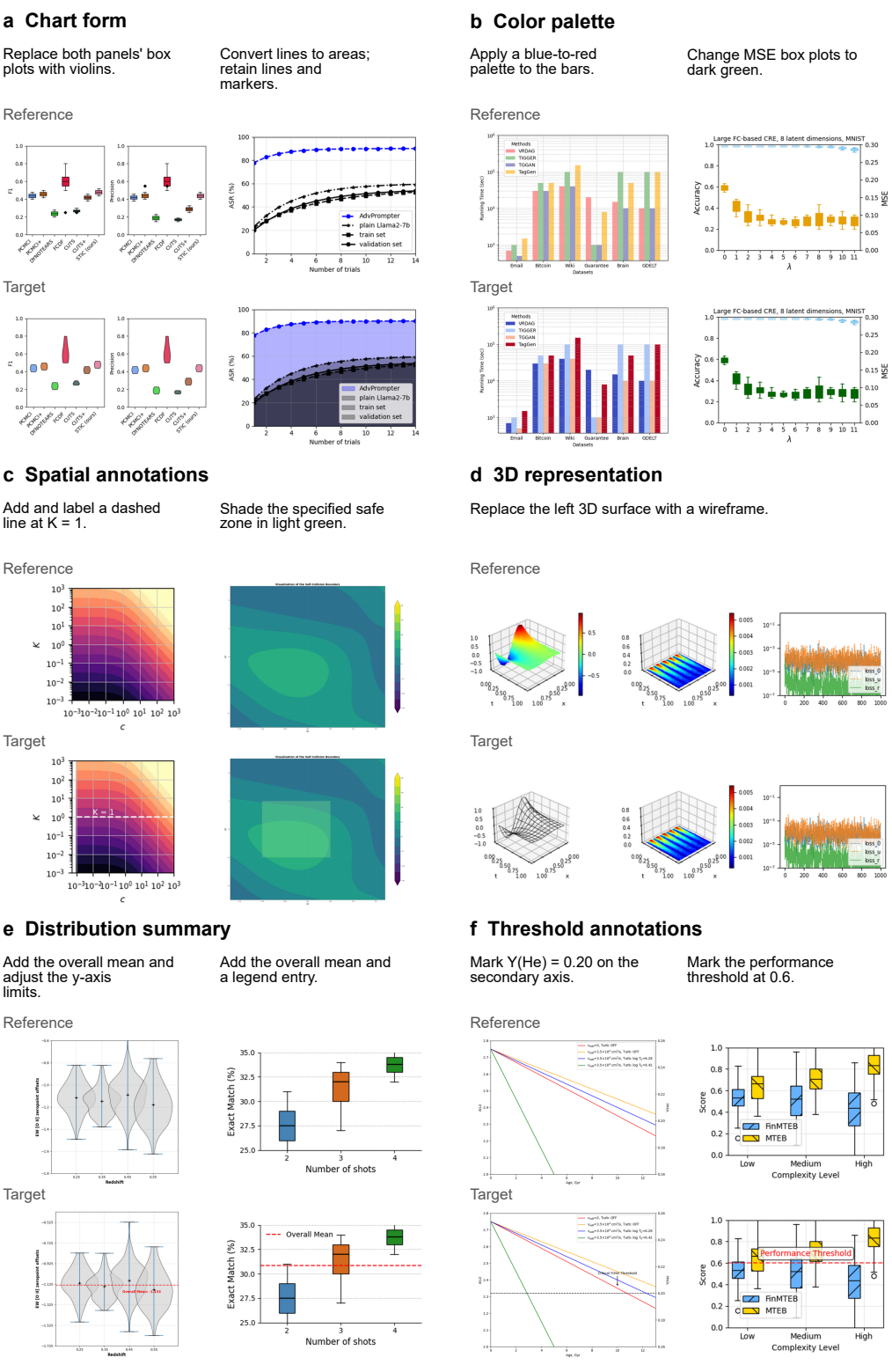


Figure 8: Eleven editing examples grouped into six categories. Categories (a), (b), (c), (e), and (f) each contain two cases side by side; (d) contains one wider 3D case. Within each case, the reference chart appears above its target, beneath a condensed instruction. Images are uniformly resized without content edits.

A.1.3 REASONING GENERATION AND VERIFICATION

Reference descriptions separate directly observed, inferred, and uncertain chart properties. The reasoning generator receives this description together with the original reference image and instruction, but the description is excluded from the student's SFT input. Description generation uses Qwen3.7-Flash with temperature 0.2, top-p 0.9, a 4,096-token response limit, and thinking disabled. Reasoning–code generation uses temperature 1.0, top-p 0.9, a 16,384-token response limit, and thinking enabled.

Generated programs are executed with a 60-second timeout. Qwen3.7-Flash judges program and reasoning quality at temperature 0. Let s_c be the mean of the code-level and image-level scores and s_r the mean of the three reasoning scores. The reasoning judge is called only when $s_c \geq 70$, and final selection requires $s_c > 70$ and $s_r > 90$. The retained 266,454 trajectories form the SFT dataset. The complete generation and evaluation prompts appear in Appendix A.1.6.

A.1.4 GENERATION MODEL AND API COST

The construction pipeline requires multiple generation and evaluation calls per candidate. Running Qwen3.5-9B locally throughout this pipeline was infeasible under our compute budget, so we used the low-cost Qwen3.7-Flash API. Table 9 includes the Batch Chat mode used in our pipeline.

Table 9: API prices in USD per one million tokens. Qwen3.7-Flash uses the official USD-equivalent price for the China (Beijing) $\leq$ 32K-token tier. Batch Chat applies the 50% batch-inference discount used in our pipeline. Prices were accessed on September 18, 2026.

Model / mode	Input (USD / 1M tokens)	Output (USD / 1M tokens)
Qwen3.7-Flash (Standard Chat)	0.028	0.110
Qwen3.7-Flash (Batch Chat)	0.014	0.055
GPT-5.6 Luna	0.20	1.20
Gemini 3 Flash	0.50	3.00
GPT-5.6 Terra	2.00	12.00
Gemini 3.1 Pro	2.00	12.00

Qwen prices and the 50% batch discount follow Alibaba Cloud Model Studio.[1] GPT prices follow the OpenAI API documentation.[2] Gemini prices follow the Google AI documentation.[3]

Table 10: Comparison of the data-construction model and the deployed backbone.

	ChartEdit		Direct		Customized	
Model	Exec.	Overall	Exec.	Overall	Exec.	Overall
Qwen3.5-9B	78.1	72.1	71.9	58.7	74.7	61.4
Qwen3.7-Flash	81.7	67.3	82.5	65.8	80.2	67.7

Table 10 uses the results in Table 1, with Overall defined as the mean of the two non-execution metrics. Qwen3.7-Flash has higher execution rates on all three tasks and higher Overall scores on both ChartMIMIC tasks, but lower ChartEdit Overall. It is therefore a competitive, low-cost construction model.

A.1.5 TRAINING CONFIGURATION

SFT updates the full Qwen3.5-9B model, including its vision encoder and aligner, on the direct-reproduction and editing data. Table 11 summarizes the launch-script configuration. Four workers, one example per device, and 64 accumulation steps give an effective batch size of 256. The reported run used four A800 GPUs and took approximately 32 hours.

[1] Alibaba Cloud: model pricing and OpenAI-compatible Batch API.

[2] OpenAI: model pricing and specifications.

[3] Google AI: Gemini API pricing.

Table 11: SFT launcher defaults. Dataset size, hardware, and runtime describe the reported run.

Parameter	Value	Parameter	Value
Base model	Qwen3.5-9B	Training examples	266,454
Tuning method	Full parameters	Epochs	1
GPUs	$4\times$ A800	Training time	$\sim$32 h
Effective batch size	256	Learning rate	5×10^{-5}
Gradient accum. steps	64	Weight decay	0.1
LR scheduler	Cosine	Warmup ratio	0.01
Max sequence tokens	8,192	Max image pixels	1,003,520
DeepSpeed	ZeRO-3	Precision	BFloat16
Vision encoder	Trainable	Aligner	Trainable
Attention	FlashAttention	Padding-free	Enabled

A.1.6 PROMPT TEMPLATES

The following templates are reproduced from the generation and evaluation scripts used in the data pipeline. Placeholders in braces are filled with the corresponding instruction, reasoning trace, or program.

Reference-chart Description Generation

```
You are carefully inspecting a rendered chart image.

You are also given the source code as private verification context.
    Use it only to
check or disambiguate what the rendered image supports. Never expose
    code, variable
names, libraries, APIs, hidden parameters, random seeds, or
    implementation details.

Separate direct observations from inferences. A direct observation is
    readable or
visually apparent in the image, such as visible text, chart marks,
    approximate layout,
color families, a legend, or gridlines. An inference is an estimate
    from visual
evidence, such as a bar value estimated from ticks, an approximate
    font size, an exact
shade inferred from a visible color, or an occluded label inferred
    from context.

Return only one valid JSON object with this exact schema:
{
  "direct_observations": [
    {
      "statement": "I can see ...",
      "confidence": "high" | "medium"
    }
  ],
  "inferred_properties": [
    {
      "statement": "From ..., I infer that ...",
      "confidence": "medium" | "low"
    }
  ],
  "uncertainties": [
    "I cannot reliably determine ..."
  ]
```

Reference-chart Description Generation (continued)

```
}

Keep statements concise, information-dense, and grounded in the
    rendered image.
Do not place any implementation fact in direct_observations. Do not
    claim an exact
numeric value, font size, or hex color unless the image itself
    visibly supports it;
otherwise state it as an estimate in inferred_properties.
```

Figure 9: Prompt for constructing a grounded description of the reference chart while preventing source-code details from entering the description.

Reasoning and Code Generation

```
You are generating a high-quality first-person reasoning trace for
the chart-to-code task. The task may require either reconstructing
    the reference chart
faithfully or editing it according to a user request.

The downstream student receives only the reference chart image and
    the user's instruction.
The instruction may request a specific edit, or it may simply ask for
    faithful
reconstruction. During trace generation, you additionally receive a
    private intermediate
visual representation produced from the reference image. Use it only
    as silent auxiliary
evidence. Never mention, quote, or allude to this representation, its
    JSON format, its
generation process, variables, libraries, APIs, hidden parameters, or
    random seeds.

The trace is not a short caption and not a generic description of a
    chart. It should
reconstruct the visual evidence in an organized way and explain how
    that evidence leads
to the decisions needed for the edit. Every statement must be written
    in the first person.
Use phrases such as "I can see...", "I notice...", "I observe...", "I
    infer from...",
"This suggests to me...", "I cannot determine exactly...", and "I
    will...". Do not write
impersonal notes such as "the chart has..." when you can write "I can
    see that the chart
has...".

Separate observation from inference carefully:

1. Directly observable evidence:
   Describe what I can actually see or read in the image before
    interpreting it. Include
   the position, size, orientation, alignment, and spatial
    relationship of important
   elements. Describe the canvas and margins, the overall layout, the
    number and
   arrangement of visible panels, titles, axes, tick labels, legends,
    annotations, marks,
   and whitespace. For visual styles, describe visible color
    families, contrast, line
```

Reasoning and Code Generation (continued)

```
   weight, marker shape, bar borders, fills, gridlines, spines, text
    hierarchy, and
   relationships between elements. Connect observations when useful,
    for example:
   "I can see the legend in the upper-right area, aligned with the
    plotting region, and
   its entries correspond to the differently colored lines."

2. Evidence-based inference:
   When something is not directly readable, explain how I infer it
    from visible evidence.
   For example, estimate data values from the positions of marks
    relative to ticks; infer
   the number of series from repeated colors or markers; infer a
    subplot's chart type from
   the geometry and arrangement of its marks; infer approximate font
    scale from its size
   relative to the canvas and other labels; or infer which visual
    element an annotation
   refers to from its position and connector. State the evidence
    first, then
   the inference:
   "I can see the tops of the bars at approximately these tick
    positions, so I infer
   that the values are roughly...". Use uncertainty words such as
    "approximately",
   "likely", or "I cannot determine exactly" whenever the image does
    not justify
   precision. Never present an invented value, exact hex color, exact
    font size, hidden
   datum, or implementation fact as directly observed.

3. Reasoning discipline:
   Do not reverse-rationalize from the code you intend to write.
    First establish the
   image evidence, then explain the inference, then decide how to
    implement the requested
   edit. The private intermediate representation may help resolve an
    ambiguity internally,
   but the trace must remain grounded in what a model could visually
    inspect and must not
   reveal that auxiliary representation was consulted. Do not invent
    panels, series,
   annotations, labels, or style details that are absent. If an
    element is unclear or
   occluded, say so explicitly.

Organize the reasoning around the image, but do not turn it into a
    rigid checklist,
numbered outline, table, JSON object, or formal report. The trace
    should read like
natural o1-style internal reasoning in the first person: I look at
    one aspect of the
figure, connect it to nearby or corresponding elements, draw a
    cautious conclusion,
and then continue to the next relevant aspect. It is acceptable to
    move between global
layout and local details when that is how the visual evidence
    supports the conclusion.

For a reconstruction task, explain how I will recover the chart's
    structure, visual
```

Reasoning and Code Generation (continued)

````
encodings, approximate data, text, styles, and layout from the
    reference image. For an
edit task, explain the requested change and which existing properties
    I will preserve.
In both cases, I must first analyze the whole reference image at fine
    granularity and
then determine which details, elements, and data need to be changed
    or reproduced.
The trace should make the evidence-to-decision chain clear without
announcing artificial stages such as "Step 1" or "Step 2".

The reasoning should be detailed enough to teach another model how
    visual evidence
supports a decision, but it should not become repetitive narration or
    a polished
post-hoc report. It must remain a coherent, natural first-person
    account of what I
see, what I can read from the image, what I infer from those
    observations, what remains
uncertain, and what I will implement.

Return exactly two Markdown fenced blocks:
```trace
A substantive natural-language first-person reasoning trace. Do not
    use numbered
headings, bullet points, JSON, or a rigid template. Follow the visual
    evidence in a
clear but conversational chain from observation to inference to
    implementation.
```
```python
one complete runnable Python script
```

Do not add text after the code block.
"""
````

Figure 10: Prompt for generating a first-person reasoning trace and a complete executable plotting program from the reference chart, instruction, and private description.

Reasoning Evaluation

```
You are an expert evaluator of reasoning traces for a chart-to-code
    task. The attached
image is the only visual evidence available to the model. Evaluate
    the reasoning trace
itself, not merely whether the final chart or code happens to be
    successful.

User instruction:
<instruction>
{instruction}
</instruction>

Reasoning trace:
<reasoning_trace>
{reasoning_trace}
</reasoning_trace>

Final generated code:
```

Reasoning Evaluation (continued)

```
```python
{generated_code}
```

Score exactly three dimensions from 0 to 100:

1. Task Understanding and Grounding: Does the trace correctly
    understand every material
part of the instruction and ground its observations and inferences in
    the attached image?
Penalize missed requirements, incorrect visual identification,
    unsupported exact claims,
and invented chart elements. Appropriate uncertainty should not be
    penalized.

2. Reasoning Quality: Is the reasoning correct, coherent, and
    sufficiently complete to
support the proposed implementation? Penalize faulty technical
    claims, contradictions,
missing critical decisions, and irrelevant repetition. Do not reward
    verbosity by itself.

3. Trace-Code Alignment: Does the final code implement the decisions
    stated in the trace?
Penalize planned changes that are absent, important code choices that
    contradict
the trace,
and substantial implementation choices left unexplained. Judge
    alignment rather than the
standalone visual quality of the rendered chart.

The overall Score must be the arithmetic mean of the three dimension
    scores, rounded to
the nearest integer. Keep each comment concise and evidence-based.
    Return only valid JSON
inside one fenced JSON block, with exactly this structure:

```json
{{
  "Task Understanding and Grounding": {{"score": 0, "comment":
    "..."}},
  "Reasoning Quality": {{"score": 0, "comment": "..."}},
  "Trace-Code Alignment": {{"score": 0, "comment": "..."}},
  "Score": 0
}}
```

Figure 11: Prompt for evaluating task grounding, reasoning quality, and agreement between the reasoning trace and generated program.

Image-Level Program Evaluation

```
You are an excellent judge at evaluating visualization chart plots.
    The first
image is the reference chart produced by ground-truth matplotlib
    code, and the
second image is produced by model-generated
matplotlib code. Score how well the generated plot matches the
    reference plot.

Scoring criteria, totaling 100 points:
```
```

Image-Level Program Evaluation (continued)

```
1. Chart Types (20): all chart types are present.
2. Layout (10): subplot arrangement and overall geometry match.
3. Text Content (20): titles, annotations, axis labels, and legends
    match,
   excluding tick labels.
4. Data (20): trends, data groups, and relative values match.
5. Style (20): colors, markers, lines, grids, legends, and other
    styling match.
6. Clarity (10): clear rendering without severe overlaps or artifacts.

Return only:
'''json
{
  "Chart Types": {"score": 0, "comment": "..."},
  "Layout": {"score": 0, "comment": "..."},
  "Text Content": {"score": 0, "comment": "..."},
  "Data": {"score": 0, "comment": "..."},
  "Style": {"score": 0, "comment": "..."},
  "Clarity": {"score": 0, "comment": "..."},
  "Score": 0
}
'''
```

Figure 12: Prompt for comparing the rendered generated chart with the target chart across type, layout, text, data, style, and clarity.

Code-Level Program Evaluation

```
You are an expert evaluator for a Python chart-code generation task.
You will be provided with the original Python code and the user's
edit/reproduction instruction,
and the code generated by the model.

Original code:
'''python
{source_code}
'''

Instructions:
{instruction}

Generated code:
'''python
{generated_code}
'''

Scoring criteria, totaling 100 points:
1. Modification Accuracy (30): the code makes accurate and
    comprehensive modifications.
2. Code Completeness (30): the code is complete, detailed, precise,
    and runnable.
3. Instruction Following Performance (40): the generated code follows
    the
instruction with no missed or misinterpreted elements.

Return only:
'''json
{{
  "Modification Accuracy": {{"score": 0, "comment": "..."}},
  "Code Completeness": {{"score": 0, "comment": "..."}},
```

Code-Level Program Evaluation (continued)

```
  "Instruction Following Performance": {{"score": 0, "comment":
    "..."}},
  "Score": 0
}}
```

Figure 13: Prompt for evaluating modification accuracy, code completeness, and instruction following using the source program and generated program.

A.2 On-policy self-distillation

A.2.1 Training data and privileged inputs

We sampled 26k pairs not selected for SFT, without synthesized reasoning targets. Student messages contain only the reference image and instruction; a separate `teacher_prompt` contains the selected PI. The plugin preserves this field and checks image-placeholder consistency. The target-code launcher selects its PI dataset and inherits the common training settings.

A.2.2 Training configuration

OPSD uses ms-swift GKD with LoRA updates to the student and a fixed teacher at the initial checkpoint. The vision encoder and aligner remain frozen. Table 12 reports the launcher settings. Four workers, one example per device, and 64 accumulation steps give an effective batch of 256. Objective and top-K ablations change the corresponding settings while keeping the remaining configuration fixed.

Using the distributions defined in Section 3.3.2, forward KL uses $\mathrm{KL}(p_T^t \| p_S^t)$ and reverse KL uses $\mathrm{KL}(p_S^t \| p_T^t)$. The implementation approximates the vocabulary distribution loss using top-K logits. The default is symmetric JSD with $K = 64$. The objective comparison fixes $K = 64$, while the logit budget comparison uses JSD with $K \in \{32, 64, 128\}$.

Table 12: OPSD launcher defaults. The target-code launcher selects its PI dataset and inherits the common GKD settings.

Parameter	Value	Parameter	Value
Initial model	Qwen3.5-9B SFT	Training examples	26k
Student tuning	LoRA	Teacher mode	Fixed initial model
GPUs	4× H100	Epochs	1
Effective batch size	256	Learning rate	2×10^{-5}
Gradient accum. steps	64	Generation batch size	256
LoRA rank	64	LoRA alpha	128
JSD β	0.5	Logit top-K	64
On-policy λ	1.0	Temperature	1.0
Auxiliary SFT weight	0	Thinking mode	Disabled
Max prompt tokens	4,096	Max completion tokens	4,096
Max image pixels	602,112	DeepSpeed	ZeRO-2
Vision encoder	Frozen	Aligner	Frozen
Attention	SDPA	Precision	BFloat16

A.2.3 GRPO comparison configuration

The GRPO comparison starts from the same SFT model and uses four H100 GPUs. Its verl launcher samples four responses per prompt, with Qwen3.7-Flash providing remote reward scores (Table 13).

Table 13: GRPO comparison configuration on four H100 GPUs.

Parameter	Value	Parameter	Value
Initial model	Qwen3.5-9B SFT	Method	verl GRPO
GPUs	$4\times$ H100	Epochs	1
Train batch size	256	Rollouts per prompt	4
Actor tuning	LoRA	LoRA rank / alpha	64 / 128
Actor learning rate	1×10^{-5}	KL-loss coefficient	0.01
Max prompt tokens	4,096	Max response tokens	4,096
PPO mini-batch	256	Reward judge	Qwen3.7-Flash

A.2.4 Evaluation protocols

Held-out GKD loss. The evaluation-only script `eval_chartedit_gkd_loss.py` calls `GKDTrainer.evaluate()` without optimization. It loads full or merged student checkpoints. The Base comparison uses the original Qwen3.5-9B teacher, and the SFT comparison uses the SFT teacher; each remains fixed before and after OPSD.

On held-out ChartEdit and ChartMIMIC inputs, each checkpoint generates fresh student responses with $\lambda = 1$. The teacher scores these responses with the corresponding PI. Table 6 therefore measures on-policy held-out loss, not the fixed-completion distribution gap used in Visual-OPSD (Li et al., 2026a).

Table 14: Default settings of the evaluation-only GKD runner. Selected effective settings are saved with each run's metrics.

Parameter	Value	Parameter	Value
Student loading	Full/merged model	Parameter updates	None
JSD β	0.5	Logit top-K	64
On-policy λ	1.0	Auxiliary SFT weight	0
Temperature	1.0	Random seed	42
Max sequence tokens	16,384	Max completion tokens	2,048
vLLM model length	8,192	Max image pixels	602,112
Eval. batch per GPU	8	Thinking mode	Disabled

Table 14 gives evaluation defaults separately from training. Top-64 loss computation, token masking, and reduction are delegated to the installed ms-swift trainer; the wrapper reports its aggregate `eval_loss` without additional averaging. The complete input JSONL is evaluated without further sampling. Checkpoint paths, input-file SHA-256, and selected effective settings are saved with the metrics. Relative loss reduction is $\rho_z = 100(1 - J_z^{\text{after}}/J_z^{\text{before}})$. Since rollouts change across checkpoints, this is not fixed-prefix gap closure or a measure of task correctness.

PI-conditioned generation. Table 7 tests the Base and SFT checkpoints with PI, without further training. ChartEdit uses the human subset; each ChartMIMIC task uses 300 randomly sampled examples. Comparisons use the corresponding no-PI checkpoint on the same subset, not the full-benchmark scores.