Visual-ERM: Reward Modeling for Visual Equivalence

Ziyu Liu[*1,2], **Shengyuan Ding**[*2,3], **Xinyu Fang**[2], **Xuanlang Dai**[2], **Penghui Yang**[1,2], **Jiaqi Wang**[2,5], **Kai Chen**[2], **Dahua Lin**[2,4], **Yuhang Zang**[2✉]
[1]Shanghai Jiao Tong University [2]Shanghai Artificial Intelligence Laboratory
[3]Fudan University [4]The Chinese University of Hong Kong [5]Shanghai Innovation Institute

Abstract

Vision-to-code tasks require models to reconstruct structured visual inputs, such as charts, tables, and SVGs, into executable or structured representations with high visual fidelity. While recent Large Vision Language Models (LVLMs) achieve strong results via supervised fine-tuning, reinforcement learning remains challenging due to misaligned reward signals. Existing rewards either rely on textual rules or coarse visual embedding similarity, both of which fail to capture fine-grained visual discrepancies and are vulnerable to reward hacking. We propose the Visual Equivalence Reward Model (Visual-ERM), a multimodal generative reward model that provides fine-grained, interpretable, and task-agnostic feedback to evaluate vision-to-code quality directly in the rendered visual space. Integrated into Reinforcement Learning (RL), Visual-ERM improves Qwen3-VL-8B-Instruct by +8.4 on chart-to-code and yields consistent gains on table and SVG parsing (+2.7, +4.1 on average), and further strengthens test-time scaling via reflection and revision. We also introduce VisualCritic-RewardBench (VC-RewardBench), a benchmark for judging fine-grained image-to-image discrepancies on structured visual data, where Visual-ERM at 8B decisively outperforms Qwen3-VL-235B-Instruct and approaches leading closed-source models. Our results suggest that fine-grained visual reward supervision is both necessary and sufficient for vision-to-code RL, regardless of task specificity.

1 Introduction

Large Vision Language Models (LVLMs) have made rapid progress in multimodal understanding [12, 25, 3]. Among the resulting capabilities, **vision-to-code** (Fig. 1 top) stands out as a particularly important primitive: it converts structured visual inputs such as charts, tables, and SVGs into executable code [47] or markdown [24]. Vision-to-code has become a key primitive for downstream uses including UI-to-code generation [33], scientific document parsing, and knowledge management.

Most existing work improves vision-to-code with supervised fine-tuning (SFT) [47, 48], which is data-intensive and often *generalizes poorly* across tasks. Reinforcement learning (RL) is a natural alternative [18, 37], but its effectiveness depends on the **reward signal**. A reliable reward for vision-to-code should be *visual* (judge the rendered output rather than the source code), *fine-grained* (penalize element-level errors in layout, axes, and content), *interpretable* (localize what is wrong rather than emit an opaque scalar), and *task-agnostic* (cover charts, tables, and SVGs within a single model). As shown at the bottom of Fig. 1, **current rewards fall short** on these axes. Text-based rules such as edit distance and Tree-Edit-Distance Similarity (TEDS) operate on predicted code or markup, so they are not *visual*: layout, alignment, and other rendering errors are invisible to them. Vision-encoder similarities such as DINO [34] are *visual* but not *fine-grained* or *interpretable*: their encoders are pre-trained for semantic invariance and tolerate small spatial or structural shifts that change the meaning of a chart or table, while collapsing evidence into a single scalar (Sec. 2). Neither

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

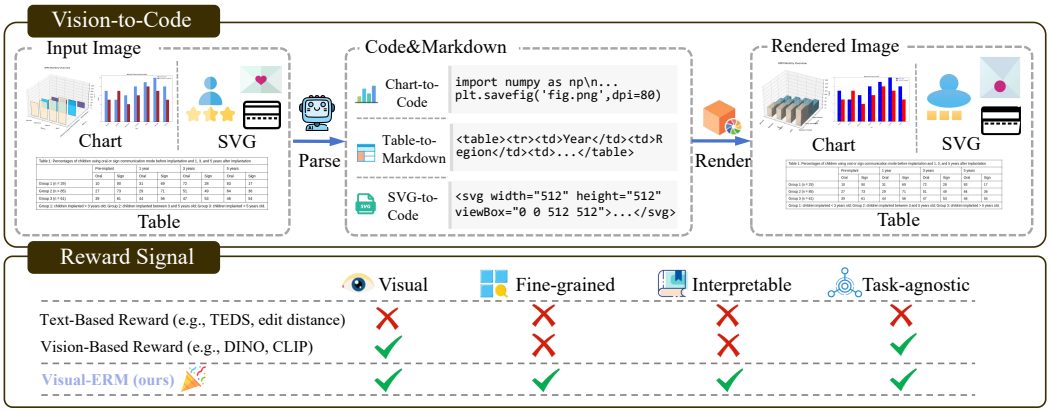


Figure 1: **Overview of vision-to-code reward modeling and Visual-ERM. Top:** vision-to-code parses a structured visual input (chart, table, or SVG) into code or markup, which is rendered back to an image so that quality can be judged in the visual space. **Bottom:** we compare reward signals along four properties: *visual*, *fine-grained*, *interpretable*, and *task-agnostic*. Text-based rewards (e.g., TEDS, edit distance) ignore visual cues; vision-encoder similarities (e.g., DINO [34], CLIP [29]) capture visual content but remain coarse and opaque. Visual-ERM combines all four properties, providing a reliable supervisor across vision-to-code tasks.

family is *task-agnostic* across chart, table, and SVG within a single model, leaving a **gap** between the reward and the underlying notion of *visual equivalence*. We present the detailed analysis in Sec. 2.

We address this gap with the Visual Equivalence Reward Model (Visual-ERM), a multimodal generative reward model that scores vision-to-code outputs by *reasoning over the rendered image directly*. We train Visual-ERM on a discrepancy-annotated corpus of reference–prediction image pairs assembled from two complementary sources: targeted edits that inject pre-defined error types and natural inferences from weaker LVLMs, with fine-grained annotations distilled from a stronger proprietary model. At deployment, the same model serves two roles: it converts per-error severities into a bounded scalar that, paired with a render-success term, supplies the reward for GRPO-based RL, and it returns the structured discrepancy list as natural-language feedback for test-time scaling.

We evaluate Visual-ERM along three axes. On RL, we integrate it into vision-to-code pipelines and observe that Visual-ERM boosts Qwen3-VL-8B-Instruct [3] by $+8.4$ points on chart-to-code and yields consistent gains on table-to-markdown ($+2.7$) and SVG-to-code ($+4.1$). To assess the reward signal itself, we introduce VisualCritic-RewardBench (VC-RewardBench), a benchmark for fine-grained image-to-image discrepancy judgment on structured visuals; VC-RewardBench exposes consistent weaknesses of current LVLMs, and Visual-ERM at 8B *improves over Qwen3-VL-235B-Instruct*, narrowing the gap to leading closed-source models. For test-time scaling, Visual-ERM provides interpretable feedback that drives reflection and revision, yielding further gains.

Our contributions are: **1)** We analyze existing reward paradigms for vision-to-code and identify four properties a reliable reward must satisfy, namely *visual*, *fine-grained*, *interpretable*, and *task-agnostic*; we further show that existing rewards each violate at least two of these properties and induce reward hacking under RL. **2)** Guided by this analysis, we propose Visual-ERM, a multimodal generative reward model that scores rendered outputs with fine-grained, interpretable, and task-agnostic signals, and we integrate it into both RL and test-time scaling. **3)** We release VisualCritic-RewardBench, a benchmark of 1,335 instances for fine-grained image-to-image discrepancy judgment. **4)** Visual-ERM improves Qwen3-VL-8B-Instruct by $+8.4$ on chart-to-code, $+2.7$ on table-to-markdown, and $+4.1$ on SVG-to-code, with further gains from reflection-and-revision at inference time.

2 Analysis of Reward Signals for Vision-to-Code

RL is the most promising route to scale vision-to-code beyond imitation, but RL is only as reliable as its reward. We first present the limitations of existing reward families in Fig. 2.

Setup. Let x denote the rendered reference image and $\hat{y}_{\text{img}}$ the rendered prediction. We define *visual equivalence* as the property that $\hat{y}_{\text{img}}$ matches x at every level a downstream user would inspect, including layout, axes, labels, and numeric content. We probe each reward in two complementary

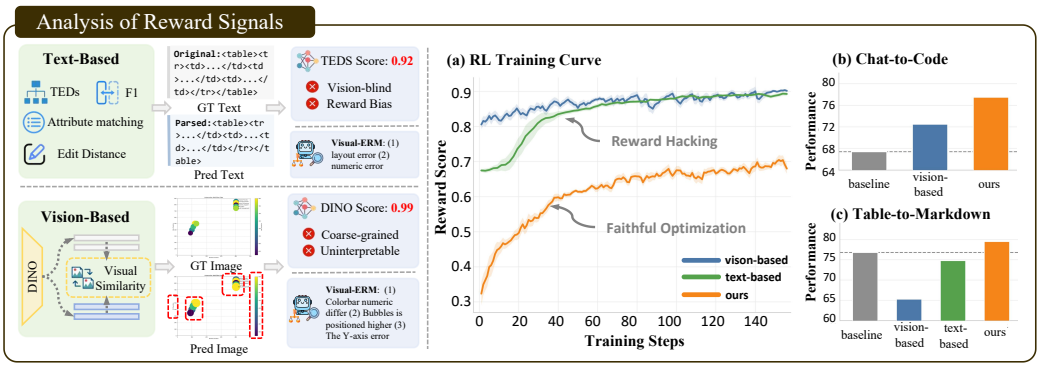


Figure 2: **Analysis of existing rewards and downstream impact.** *Left:* a text-based reward (TEDS) and a vision-based reward (DINO) assign near-perfect scores to predictions that Visual-ERM identifies as visually wrong. *Right:* (a) RL training curves where text- and vision-based rewards plateau via reward hacking while Visual-ERM keeps improving; (b, c) downstream gains on Chart-to-Code and Table-to-Markdown.

ways: qualitative case studies on held-out chart and table examples, and RL training under each reward followed by downstream evaluation.

Text-based rules are blind to rendering. TEDS, edit distance, and related metrics compare predicted code or markup tokens against a reference sequence and never inspect x or $\hat{y}_{\text{img}}$. A prediction that is textually close but renders to the wrong figure therefore receives a near-perfect score, and a prediction that renders correctly via different yet equivalent code is penalized. The top-left case in Fig. 2 shows a table-to-markdown output with **TEDS** $=$ 0.92 whose rendered version exhibits both header-hierarchy and numeric-cell errors, all of which Visual-ERM localizes.

Vision-encoder similarities are tuned for the wrong invariance and are uninterpretable. Vision foundation models such as DINO [6] and CLIP [29] are pre-trained to be invariant to translation, scale, and other transformations that preserve semantic content. This invariance is at odds with vision-to-code, where a small spatial shift or a permuted bar order changes the underlying data, and the resulting scalar also hides *which* element is wrong. The bottom-left case in Fig. 2 shows a chart with **DINO similarity above** 0.99 for which Visual-ERM enumerates colorbar, bubble-position, and y-axis errors.

Reward hacking under RL. We optimize a chart-to-code policy against each reward and report the proxy reward score over training in Fig. 2(a). The text-based and vision-based curves rise quickly and saturate, consistent with reward hacking: the policy maximizes the proxy without improving the rendered output. Optimizing against Visual-ERM keeps improving throughout training, suggesting that its supervision remains aligned with rendered-image fidelity. The next paragraph translates these proxy trends into rendered-task performance.

Downstream impact. Fig. 2(b, c) evaluates the resulting policies on Chart-to-Code and Table-to-Markdown. Policies trained with text-based or vision-based rewards do not consistently improve over the SFT baseline, and underperform it on Table-to-Markdown, indicating that the proxy gains in Fig. 2(a) are illusory. Optimizing against Visual-ERM yields the largest gains in our comparison on both tasks.

Summary. Together, the failure cases and the RL probes point to four properties a vision-to-code reward must satisfy: (i) be visual, operating in the rendered image space, since text-only rewards are blind to rendering; (ii) be fine-grained and locally sensitive, since coarse global similarity hides element-level errors; (iii) be interpretable, since a single scalar invites reward hacking and gives the policy no signal about *what* to fix; and (iv) be task-agnostic, since the same failures recur across charts, tables, and SVGs. Sec. 3 instantiates these four properties in Visual-ERM.

3 Methods

The analysis in Sec. 2 defines four properties for a vision-to-code reward: it must operate in the rendered image space, be fine-grained and locally sensitive, expose interpretable evidence, and remain task-agnostic. Visual-ERM instantiates these properties through three coupled components, each

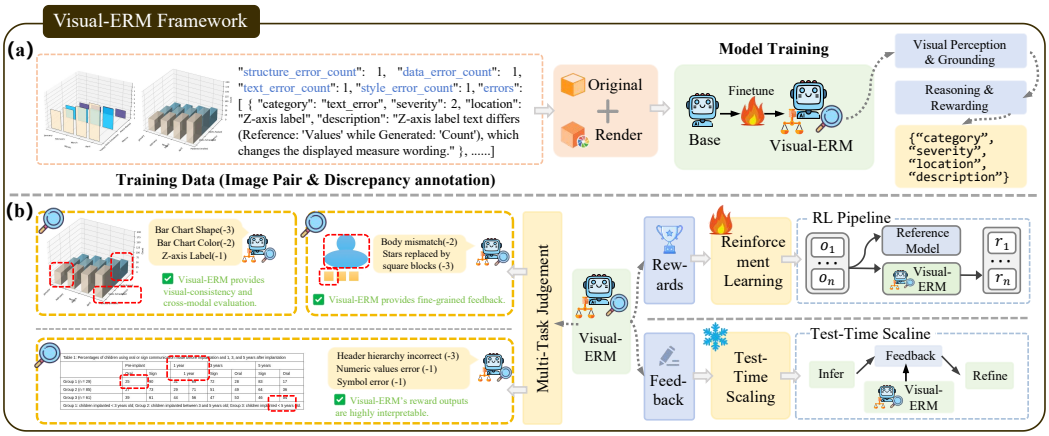


Figure 3: **(a) Training data and model training.** From paired ground-truth and rendered images, we obtain fine-grained discrepancy annotations (category, severity, location, description) and fine-tune a base LVLM into Visual-ERM. **(b) Deployment.** Visual-ERM produces unified judgments across chart, SVG, and table tasks, yielding scalar rewards for RL and natural-language feedback for test-time scaling.

visualized in Fig. 3: (i) a discrepancy-annotated dataset $\mathcal{D}_{\text{reward}}$ and a single LVLM trained on it; (ii) a deployment-time pipeline that converts Visual-ERM outputs into a scalar reward for RL; and (iii) the same outputs reused as natural-language feedback for test-time scaling. We also release VC-RewardBench, a diagnostic benchmark for fine-grained image-to-image discrepancy judgment.

Notation. We retain the symbols introduced in Sec. 2. Let $m \in \{\text{CHART}, \text{TABLE}, \text{SVG}\}$ index the task and $\mathcal{R}_m(\cdot)$ its renderer. A vision-to-code policy $\pi_\theta(y \mid x)$ takes the reference image x as input and emits structured text y, which is rendered to $\hat{y}_{\text{img}} = \mathcal{R}_m(y)$. We write $f_{\theta_{\text{ERM}}}$ for Visual-ERM; $y^\star$ denotes ground-truth code or markup.

3.1 Visual-ERM: Data, Annotation, and Training

The training pipeline shown in Fig. 3(a) consists of three stages: image-pair construction, fine-grained discrepancy annotation, and supervised fine-tuning.

Image-pair construction. We curate pairs $(x, \hat{y}_{\text{img}})$ that span the discrepancies a downstream policy is likely to produce, using two complementary sources. (1) *Targeted Edit*: high-capacity LVLMs perturb $y^\star$ to inject a pre-defined taxonomy of error types, providing systematic coverage of structural, data, textual, and stylistic failures. (2) *Natural Inference*: weaker LVLMs predict y directly, so the resulting errors match the distribution that an RL policy actually emits during rollout. Each candidate y is rendered with $\mathcal{R}_m$ to obtain $\hat{y}_{\text{img}}$ and paired with the reference image x.

Fine-grained discrepancy annotation. For every pair we collect a structured annotation $a = \{(\text{category}, \text{severity}, \text{location}, \text{description})_k\}_{k=1}^{K}$ that localizes each discrepancy and grades its impact. A natural concern is whether large open-source LVLMs already produce such annotations directly; we find they do not. Even Qwen3-VL-235B-Instruct [3] misses subtle structural and textual deviations, as quantified in Sec. 4.2. We therefore use a stronger model, GPT-5-mini [35], as a *bootstrap teacher* that proposes candidate discrepancies, which are then filtered by rendering-consistency checks (proposals referring to regions absent in $\hat{y}_{\text{img}}$ are discarded) and by agreement with the ground-truth code $y^\star$ at the structural level. The resulting reward dataset is $\mathcal{D}_{\text{reward}} = \{(m, x, \hat{y}_{\text{img}}, a)\}$. We stress that GPT-5-mini is used *only* to label the training set $\mathcal{D}_{\text{reward}}$ and plays no role at evaluation time (Sec. 3.4). The teacher therefore acts as a scalable labeler for training only, not as both annotator and judge; any residual annotator bias in $\mathcal{D}_{\text{reward}}$ is bounded by Visual-ERM's downstream RL gains (Sec. 4.1), which are measured on benchmarks independent of the training labels.

Training objective. Visual-ERM is a conditional generator $f_{\theta_{\text{ERM}}}(a \mid x, \hat{y}_{\text{img}})$ that produces the structured discrepancy list and per-error severities used in Sec. 3.2. We supervise it by negative log-likelihood:

$$\mathcal{L}(\theta_{\text{ERM}}) = \mathbb{E}_{(m,x,\hat{y}_{\text{img}},a)\sim\mathcal{D}_{\text{reward}}}\Big[-\log f_{\theta_{\text{ERM}}}\big(a \mid x, \hat{y}_{\text{img}}\big)\Big]. \tag{1}$$

For an annotation sequence $a = (a_1, \ldots, a_T)$, this expands to the token-level form

$$\mathcal{L}(\theta_{\text{ERM}}) = \mathbb{E}_{(m,x,\hat{y}_{\text{img}},a)}\Big[-\sum_{t=1}^{T} \log f_{\theta_{\text{ERM}}}\big(a_t \mid x,\, \hat{y}_{\text{img}},\, a_{<t}\big)\Big], \tag{2}$$

where $a_{<t} = (a_1, \ldots, a_{t-1})$. A single $f_{\theta_{\text{ERM}}}$ is trained on all m, supporting the task-agnostic property in Sec. 2.

3.2 Visual-ERM as an RL Reward

Pipeline. Following Fig. 3(b), each rollout draws $y \sim \pi_\theta(\cdot \mid x)$, renders $\hat{y}_{\text{img}} = \mathcal{R}_m(y)$, and queries Visual-ERM on $(x, \hat{y}_{\text{img}})$ to obtain a discrepancy set $\mathcal{E} = \{e_k\}_{k=1}^{K}$ with severities $s_k \geq 0$. We pair this with a render-success reward (RSR) so that unrenderable code is excluded before fidelity is scored.

Reward design. We aggregate severities into $\mathrm{S}_{\text{VERM}} = \sum_{k=1}^{K} s_k$, normalize within the task batch $\mathcal{T}$:

$$\widetilde{\mathrm{S}}_{\text{VERM}} = \frac{\mathrm{S}_{\text{VERM}}}{\max_{j \in \mathcal{T}} \mathrm{S}^{(j)}_{\text{VERM}} + \epsilon}, \tag{3}$$

and convert to a bounded reward

$$r_{\text{VERM}} = \mathrm{clip}\big(1 - \widetilde{\mathrm{S}}_{\text{VERM}},\, 0,\, 1\big). \tag{4}$$

The overall RL reward is

$$r = r_{\text{RSR}} + r_{\text{VERM}}, \tag{5}$$

where $r_{\text{RSR}} = 1$ if $\mathcal{R}_m(y)$ succeeds and 0 otherwise.

Policy optimization. We optimize π_θ with a Group Relative Policy Optimization (GRPO)-based [32] objective using the reward in Eq. 5 and a Kullback–Leibler (KL) anchor to a reference policy π_{ref}:

$$\begin{aligned} \max_{\theta} \; \mathbb{E}_{x \sim \mathcal{D}}\Big[& \mathbb{E}_{y \sim \pi_\theta(\cdot|x)}\big[r_{\text{RSR}}(\mathcal{R}_m(y)) \\ & + \mathrm{clip}\big(1 - \widetilde{S}_{\text{VERM}}(x, \mathcal{R}_m(y)),\, 0,\, 1\big)\big] \\ & - \beta\, \mathrm{KL}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x))\Big], \end{aligned} \tag{6}$$

where β controls the strength of KL regularization and $\mathcal{D}$ is the training distribution over reference images x. The training and downstream behavior of policies optimized under Eq. 6 are reported in Fig. 2 and Sec. 4.

3.3 Visual-ERM for Test-Time Scaling

A second use of Visual-ERM, shown in Fig. 3(b), is to provide feedback for iterative self-refinement at inference time. The policy first produces an initial prediction

$$y^{(0)} \sim \pi_\theta(\cdot \mid x), \tag{7}$$

which is rendered into $\hat{y}^{(0)}_{\text{img}} = \mathcal{R}_m(y^{(0)})$ and evaluated by Visual-ERM:

$$\big(r^{(0)},\, f^{(0)}\big) = f_{\theta_{\text{ERM}}}\Big(x,\, \hat{y}^{(0)}_{\text{img}}\Big), \tag{8}$$

where $r^{(0)}$ is the bounded reward from Eq. 5 and $f^{(0)}$ is the structured discrepancy description. If $r^{(0)}$ falls below a threshold, the policy revises its solution conditional on the previous draft and the feedback,

$$y^{(t+1)} \sim \pi_\theta\Big(\cdot \mid x,\, y^{(t)},\, f^{(t)}\Big), \tag{9}$$

and the loop repeats for up to T steps. The interpretability of $f^{(t)}$, rather than $r^{(t)}$ alone, is what enables this revision step; a scalar-only reward could not localize *what* to fix. We evaluate this capability in Sec. 4.3; gains are bounded by the policy's editing ability, since TTS reuses π_θ without weight updates.

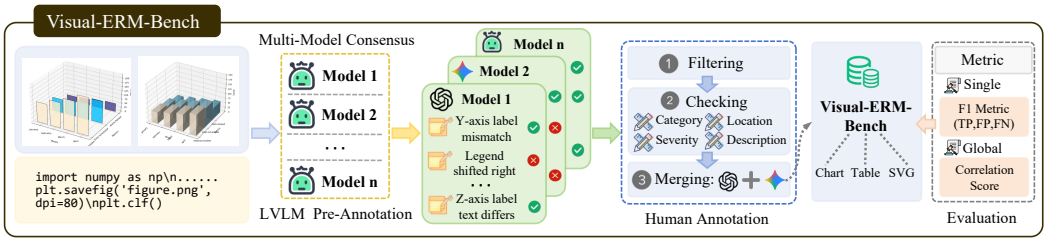


Figure 4: **VC-RewardBench.** We construct VC-RewardBench by first leveraging several advanced proprietary models for preliminary annotation, followed by manual consolidation and filtering, resulting in 1,335 high-quality instances.

3.4 VisualCritic-RewardBench

Existing reward benchmarks predominantly evaluate vision–language alignment [16], whereas the bottleneck for vision-to-code is image-to-image reconstruction fidelity. We therefore introduce VisualCritic-RewardBench (VC-RewardBench), a diagnostic suite that targets fine-grained image-to-image discrepancies in structured visuals.

VC-RewardBench contains 1,335 high-quality annotated instances, constructed following Fig. 4. Each instance pairs a reference image x with a corrupted counterpart $\hat{y}_{\text{img}}$ and a fine-grained discrepancy annotation a covering type, location, description, and severity. We adopt a multi-model consensus protocol: independent annotations from GPT-5-mini [35], Gemini-2.5-Pro, and Gemini-3-Pro [8] reduce architecture-specific bias, and PhD-level reviewers consolidate the labels into a single expert-vetted set. Example cases are provided in Sec. D.

Because VC-RewardBench mixes structured fields (e.g., counts) with free-form content (e.g., descriptions), exact-match accuracy is unsuitable. We adopt an LLM-as-Judge protocol: a judge LLM matches predicted discrepancies $\hat{A}$ to ground-truth annotations $A^\star$ to identify TP/FP/FN and compute Precision/Recall/F1. Since models also output per-discrepancy severities, we sum severities per instance and report the Pearson correlation with the ground-truth totals to measure overall scoring consistency, denoted as S_c. VC-RewardBench is a diagnostic for $f_{\theta_{\text{ERM}}}$ itself, separate from the downstream RL and TTS evaluations in Sec. 4.

4 Experiments

We evaluate Visual-ERM along three axes. (i) *RL utility*: whether using Visual-ERM as the reward in RL yields stronger policies across three vision-to-code tasks (Sec. 4.1). (ii) *Reward quality*: how well Visual-ERM judges fine-grained image-to-image discrepancies on VC-RewardBench (Sec. 4.2). (iii) *Test-time utility*: whether Visual-ERM's critiques improve on test-time scaling (Sec. 4.3).

4.1 Reinforcement Learning with Visual-ERM

Experimental Details and Benchmarks. We train Visual-ERM on top of Qwen3-VL-8B-Instruct [2]. To evaluate its utility as a reward model, we run GRPO [32] on three vision-to-code tasks: (1) *Chart-to-Code*, (2) *Table-to-Markdown*, and (3) *SVG-to-Code*, using Qwen3-VL-8B-Instruct as the policy model. For *Chart-to-Code*, we use ChartMimic [41] under both the *direct* (reproduce the input chart) and *customized* (generate a new chart under given style/data constraints) settings. For *Table-to-Markdown*, we report table-level metrics on OmniDocBench-v1/v1.5 [26] and olmOCRBench [28]. For *SVG-to-Code*, we evaluate on UniSVG [15]. We additionally compare against two stronger vision-to-code SFT policies, VinciCoder [47] and JanusCoderV-8B [36]. Please refer to Sec. A for more experimental details.

Results on Chart-to-Code. We start with the *Chart-to-Code* task. Using Qwen3-VL-8B-Instruct [3] and VinciCoder-8B-SFT [47] as policies, we run GRPO with Visual-ERM as the reward. Tab. 1 shows that Visual-ERM-guided RL improves Qwen3-VL-8B-Instruct by **+11.8** and **+4.9** average points on ChartMimic-v2 *direct* and *customized* [41]. Starting from the already strong VinciCoder-8B-SFT, the same recipe still delivers **+10.3** and **+9.8** under the two settings.

Table 1: **Evaluation Results on Chart-to-Code Tasks.** We evaluate on ChartMimic(Direct) and ChartMimic(Customized). ChartMimic(Direct) requires generating Python code that reproduces the input chart, while ChartMimic(Customized) requires generating code for a new chart that matches the given chart's style and data constraints. **Exec_rate** denotes the execution success rate; **Low** and **High** denote the low-level and high-level scores, respectively.

Model	ChartMimic [41] (Direct)				ChartMimic [41] (Customized)				Avg↑
	Exec_rate↑	Low↑	High↑	Overall↑	Exec_rate↑	Low↑	High↑	Overall↑	
Baseline									
InternVL3.5-8B [38]	62.2	41.9	48.9	45.4	79.7	53.0	63.3	58.2	51.8
Qwen3-VL-8B-Instruct [3]	82.2	62.7	72.7	67.7	86.8	66.3	76.8	71.6	69.6
VinciCoder-8B-SFT [47]	86.2	68.2	77.5	72.9	86.2	52.3	71.5	61.9	67.4
JanusCoderV-8B [36]	80.6	65.8	73.2	69.5	80.7	66.7	74.2	70.4	70.0
ChartMaster-7B [37]	93.3	74.5	82.1	78.3	88.8	59.5	74.2	66.8	72.6
Qwen3-VL-8B-Instruct [3]									
+ RL (DINO-based Reward)	90.8	71.1	81.9	76.5	91.7	71.2	80.3	75.8	76.1
+ RL (Visual-ERM)	92.5	74.4	84.6	79.5	91.5	71.8	81.1	76.5	78.0
$\Delta_{\text{vs. Qwen3-VL-8B-Instruct}}$	**+10.3**	**+11.9**	**+11.9**	**+11.8**	**+4.7**	**+5.5**	**+4.3**	**+4.9**	**+8.4**
VinciCoder-8B-SFT [47]									
+ RL (DINO-based Reward)	90.8	73.1	82.3	77.7	91.5	57.0	77.5	67.3	72.5
+ RL (Visual-ERM)	94.3	78.0	88.5	83.2	95.0	60.9	82.6	71.7	77.5
$\Delta_{\text{vs. VinciCoder-8B-SFT}}$	**+8.1**	**+9.8**	**+11.0**	**+10.3**	**+8.8**	**+8.6**	**+11.1**	**+9.8**	**+10.1**

Table 2: **Evaluation Results on Table-to-Markdown.** TEDS-S denotes TEDS-Structure-Only, and TA represents the table subtask of olmOCRBench. For Edit-Dist, where lower values indicate better performance, we use (100 − Edit-Dist) when computing the Average.

Model	OmniDocBench [26]			OmniDocBench-v1.5 [26]			olmOCRBench [28]	Avg↑
	TEDS↑	TEDS-S↑	Edit-Dist↓	TEDS↑	TEDS-S↑	Edit-Dist↓	TA↑	
Baseline								
Qwen2.5-VL-7B [4]	74.4	80.1	58.9	69.3	74.1	58.7	71.1	64.5
InternVL3.5-8B [38]	71.0	77.6	44.9	62.9	70.4	44.1	74.1	66.7
Qwen3-VL-8B-Instruct [3]	78.9	83.9	23.2	72.7	77.2	26.9	75.3	76.8
Qwen3-VL-8B-Instruct [3]								
+ RL (DINO-based Reward)	62.2	69.5	37.0	61.1	67.4	37.9	71.7	65.3
+ RL (TEDS-based Reward)	79.2	82.9	31.6	73.0	76.6	35.3	**78.6**	74.8
+ RL (Visual-ERM)	**81.4**	**86.3**	**20.7**	**75.4**	**80.4**	**24.2**	78.1	**79.5**
$\Delta_{\text{vs. Qwen3-VL-8B-Instruct}}$	**+2.5**	**+2.4**	**+2.5**	**+2.7**	**+3.2**	**+2.7**	**+2.8**	**+2.7**

We also compare Visual-ERM against the DINO-based RL reward in Tab. 1, and observe three advantages. *(1) Stronger downstream policies*: Visual-ERM-guided RL yields substantially larger gains than DINO-based RL on both Qwen3-VL-8B-Instruct and VinciCoder-8B-SFT backbones. DINO rewards reduce supervision to a fixed visual embedding space, where matching patch-level feature similarity prioritizes semantic alignment and global appearance while under-penalizing small but functionally critical deviations, so the proxy only loosely tracks human-perceived fidelity. *(2) Cross-modal coverage*: DINO rewards are unimodal and weak at penalizing errors carried by rendered text, allowing policies to improve the proxy while degrading textual faithfulness; Visual-ERM combines visual perception with cross-modal grounding, scoring reconstructions on both visual structure and rendered text. *(3) Fine-grained interpretable feedback*: trained as a generative judge, Visual-ERM emits decomposed discrepancy descriptions rather than a single scalar, which directly supports test-time scaling via reward-guided refinement (Sec. 4.3).

Results on Table-to-Markdown Parsing. We next turn to the *Table-to-Markdown* task. Starting from Qwen3-VL-8B-Instruct, we run GRPO with Visual-ERM as the reward and compare against two alternative signals: a rule-based Tree-Edit-Distance Similarity (TEDS) score, and a DINO-based feature similarity. Tab. 2 shows that Visual-ERM-based RL delivers consistent improvements, with an overall gain of **+2.7**; the gains span both textual-recognition and structural-reconstruction metrics, consistent with the cross-modal coverage of Visual-ERM's feedback.

Both alternatives fall short on this task. With TEDS, the training reward rises steadily, yet the policy improves only marginally on the target TEDS metric and slightly degrades on others, a pattern consistent with reward shortcutting in a purely textual/structural space that ignores visual cues. DINO-based RL fares worse: unlike charts, where DINO features still carry useful visual cues, tables are dominated by precise text and layout, and DINO-based RL not only fails to yield gains on *Table-to-Markdown* but also degrades performance across benchmarks.

Results on SVG-to-Code Parsing. We use VinciCoder-8B-SFT as policy and DINO-based RL as the feature-similarity baseline. Compared with charts and tables, SVG reconstruction relies more on visual geometry and styling cues and less on textual content. Tab. 3 shows that Visual-ERM-based

Table 3: **Results on UniSVG.** SSIM/LPIPS measure pixel-level and perceptual similarity, CLIP measures semantic alignment, and **Score** aggregates them.

Model	UniSVG [15] (ISVGEN)			
	SSIM↑	LPIPS↓	CLIP↑	**Score**↑
Baseline				
Qwen3-VL-8B	60.9	60.0	73.3	64.2
JanusCoderV-8B	58.1	61.7	72.6	62.8
VinciCoder-8B-SFT	81.1	19.2	92.5	87.9
+ RL (DINO)	76.9	23.3	92.7	86.3
+ RL (Visual-ERM)	85.2	12.6	95.2	91.6

Table 4: **Test-Time Scaling on *Chart-to-Code*.** Visual-ERM enables iterative self-reflection and revision; for the reflection turns study, see Tab. 6.

Model	ChartMimic [41]		**Avg**↑
	Direct↑	Cust.↑	
Qwen3-VL-8B-Instruct [3]	67.7	71.6	69.6
+ Reflection (self)	61.9	69.5	65.7
+ Reflection (Visual-ERM)	75.6	79.5	77.6
+ RL (Visual-ERM)	79.5	76.5	78.0
+ Reflection (self)	75.2	76.9	76.1
+ Reflection (Visual-ERM)	80.3	82.0	81.1

Table 5: **Evaluation Results on VC-RewardBench.** We evaluate a range of proprietary and open-source models on Visual-ERM-Bench. $F1_h$ denotes $F1_{hard}$, the strict-match F1 score; $F1_s$ denotes $F1_{soft}$, the soft-match F1 score; and S_c denotes the correlation score, measuring the overall agreement between the predicted scores and the ground-truth labels.

Model	**Chart**			**Table**			**SVG**			**AVG**		
	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c
Proprietary												
GPT-4o [1]	22.8	28.3	48.5	32.9	35.7	49.5	13.0	19.3	50.3	25.0	29.5	56.5
GPT-5.2 [35]	30.1	32.6	64.8	39.3	40.6	54.6	28.5	32.2	61.1	32.7	35.0	58.9
Gemini-2.5-Pro [8]	33.7	37.5	61.8	46.4	48.0	49.9	29.3	34.3	63.3	37.8	40.9	59.1
Gemini-3-Flash [10]	38.5	41.3	62.8	48.1	50.1	45.6	33.3	67.5	64.3	40.6	43.4	53.4
Open-source												
Qwen2.5-VL-7B [4]	3.9	5.4	11.2	2.3	3.1	12.6	1.9	7.5	37.9	2.8	5.1	15.2
InternVL3.5-8B [38]	2.5	5.7	11.0	9.9	10.9	31.7	6.1	13.1	48.9	6.7	9.6	32.5
Qwen3-VL-8B-Instruct [3]	3.3	3.5	3.8	7.0	7.8	21.4	6.1	9.4	27.1	5.3	6.5	17.5
Qwen3-VL-235B-Instruct [3]	28.0	31.8	47.2	35.7	37.4	56.2	19.4	22.8	51.5	29.5	32.4	56.2
Visual-ERM	39.9	42.8	61.2	56.4	57.6	74.8	28.3	32.6	59.6	42.1	44.7	58.4
$\Delta_{\text{vs. Qwen3-VL-8B-Instruct}}$	**+36.6**	**+39.3**	**+57.4**	**+49.4**	**+49.8**	**+53.4**	**+22.2**	**+23.2**	**+32.5**	**+36.8**	**+38.2**	**+40.9**

RL delivers consistent gains on the backbone. Notably, while DINO-based RL leads to performance degradation on the strong VinciCoder-8B-SFT baseline, Visual-ERM maintains its effectiveness. This suggests that Visual-ERM provides more robust and precise guidance than standard feature-similarity rewards, which may fail to provide meaningful gradients for highly optimized policies.

4.2 Reward-Model Evaluation on VC-RewardBench

In Sec. 4.1, the selected benchmarks measure vision-to-code parsing quality but do not directly probe whether a model can judge reconstruction fidelity or surface fine-grained discrepancies. We therefore evaluate on VC-RewardBench, which targets fine-grained discrepancy detection and interpretable feedback across vision-to-code tasks. Tab. 5 reports the results on VC-RewardBench. Built on Qwen3-VL-8B-Instruct, Visual-ERM improves $F1_h$/$F1_s$/S_c over its base by **+36.8**/**+38.2**/**+40.9**. Even Qwen3-VL-235B-Instruct struggles with fine-grained visual and textual discrepancies, whereas Visual-ERM at 8B matches or surpasses the strongest proprietary baselines on $F1_h$/$F1_s$ and remains competitive on S_c. We read this as evidence that fine-grained discrepancy detection and fidelity judgment come from reward-model specialization of a general-purpose LVLM, not from scale alone.

4.3 Ablation Studies

Test-Time Scaling with Visual-ERM. Visual-ERM's feedback is also useful at inference time. Rather than emitting only a scalar score, it returns localized discrepancy descriptions that a policy can act on. We insert Visual-ERM into the decoding loop: it critiques each candidate, and the policy revises its output before finalizing. With three reflection rounds on *Chart-to-Code*, this loop adds **+8.0** Avg over the base Qwen3-VL-8B-Instruct and a further **+3.1** on top of the Visual-ERM-RL-tuned policy (Tab. 4). Qualitative cases are provided in Fig. 5.

Ablation of the Reflection Rounds in Test-Time Scaling. In Tab. 4, we use a three-round reflection and revision pipeline by default. To study how the number of reflection rounds affects performance, we conduct an ablation where we vary the number of rounds while keeping the rest of the inference setup identical. Results are summarized in Tab. 6. Overall, increasing the number of reflection rounds yields consistent improvements, with diminishing returns beyond three rounds.

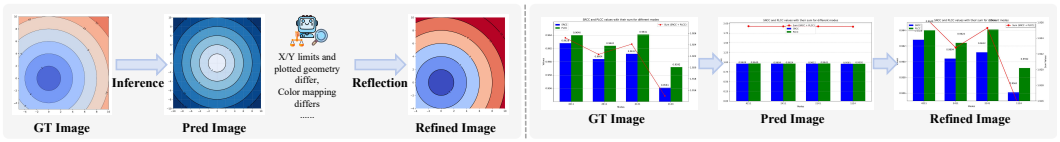


Figure 5: **Test-Time Scaling Cases.** Using Visual-ERM's fine-grained feedback, we enable an inference, reflection and refinement loop. Predictions are generated as text and rendered as images for visualization.

Table 6: **Ablation on the number of reflection rounds in test-time scaling.** We evaluate test-time scaling with different numbers of reflection and revision rounds d.

Model	ChartMimic [41] (Direct)				ChartMimic [41] (Customized)				Avg↑
	exec_rate↑	average↑	gpt_score↑	overall↑	exec_rate↑	average↑	gpt_score↑	overall↑	
Qwen3-VL-8B-Instruct [3] (+RL on Visual-ERM)	92.5	74.4	84.6	79.5	91.5	71.8	81.1	76.5	78.0
+ Reflection (w Visual-ERM) 2 rounds	92.8	75.2	84.5	79.8	92.0	74.4	82.8	78.6	79.2
+ Reflection (w Visual-ERM) 3 rounds	92.3	74.8	85.8	80.3	94.7	77.1	86.8	82.0	81.1
+ Reflection (w Visual-ERM) 4 rounds	92.0	74.5	85.4	80.0	94.2	76.9	86.1	81.5	80.7

Additional analysis. We provide extended results in the appendix, such as an evaluation on general VQA benchmarks to verify that Visual-ERM-guided RL preserves broad multimodal competence (Sec. B.1), an ablation on multi-task data mixing for both reward modeling (Sec. B.2.1) and downstream RL (Sec. B.2.2), a robustness study across LLM judges on VC-RewardBench (Sec. B.3), and an ablation on reward design with and without the render-success term (Sec. B.4).

Ablation on Targeted-Edit Data and the Natural-Inference Data. Targeted Edit accounts for 40% of the instances, while Natural Inference accounts for the remaining 60%. To quantify their individual contributions, we additionally compare reward models trained on Targeted-only, Natural-only, and Mixed data for the Table-to-Markdown task, while keeping the training data size and optimization budget identical. The results are shown in the Tab. 7. The Natural-Inference-only model performs slightly better than the Targeted-Edit-only model, while both are outperformed by the Mixed-data model, demonstrating the complementarity of the two data sources.

Human-Annotated Subset Check. We conducted a human-annotated subset check of 300 training instances, balanced across the three tasks, two data sources (Targeted Edit and Natural Inference), and the three severity levels.

The results are as follows. For teacher error rate, 3.2% of the errors reported by the teacher model were judged to be unsupported, while 7.3% of the human-identified errors were omitted. For inter-annotator agreement, the human annotators achieved a κ of 0.86 for error identification, indicating strong inter-annotator agreement. For severity-label consistency, teacher model agrees exactly with the human label in 88.7% of matched errors and differs by at most one severity level in 98.3% of cases. These results directly support the quality of the training annotations. Besides, Visual-ERM's performance on VC-RewardBench and its consistent gains across multiple visual parsing benchmarks further show that these annotations are useful in practice.

About LLM-as-a-Judge. The LLM neither assesses overall output quality nor assigns benchmark scores. It is used only to match predicted discrepancies with ground-truth items based on their descriptions, similar to semantic record matching. After alignment, scores are computed deterministically from the error types and severity labels. Its role is therefore limited to establishing item correspondence.

Agreement Study. We additionally asked human experts to perform the same item-matching procedure and recomputed the benchmark scores using the identical rule-based scoring function. The scores based on LLM exhibit a strong correlation with those based on human (Pearson's r=0.98; Spearman's ρ=0.97), confirming that the LLM reliably reproduces expert item alignment and has only a marginal effect on the final evaluation. This result, together with the judge-model ablation, supports the robustness of the VC-RewardBench evaluation protocol.

5 Related Work

Reward Models. To enable effective RL [32, 19], reward models (RMs) provide feedback that guide policy optimization. RMs can take several forms: (1) **Bradley–Terry (BT)** models that learn a scalar

Table 7: **Ablation on training data sources.** We compare mixed data with targeted-edit-only and natural-inference-only data.

Setting	OmniDocBench			OmniDocBench-v1.5			olmOCRBench	Avg↑
	TEDS↑	TEDS-S↑	Edit-Dist↓	TEDS↑	TEDS-S↑	Edit-Dist↓	TA↑	
Mixed	79.4	84.6	21.7	73.9	79.1	25.2	78.9	78.4
Targeted-Edit-Only	79.1	84.2	22.1	73.5	78.5	25.6	78.2	77.9
Natural-Infer-only	79.4	84.4	21.8	73.6	78.8	25.3	78.4	78.2

reward from pairwise comparisons and are often instantiated as discriminative rankers [5, 50, 44]; (2) **generative** RMs that produce natural language critiques or judgments which can be mapped to rewards [14, 43, 39, 20]; and (3) **thinking/agentic** RMs that perform multi-step evaluation, e.g., decomposing criteria, self-reflection, or invoking tools before returning a final score [9, 17, 27]. Most prior RMs are developed for text-centric generation (e.g., writing and dialogue) and do not support *visual-to-code* tasks, where quality is mainly determined by visual fidelity rather than text. Therefore, we propose Visual-ERM, a visual equivalence reward model for visual-to-code tasks.

Visual-to-Code Tasks. Visual-to-code spans a family of practical structured perception tasks that convert images into executable or structured representations. *Chart-to-Code* aims to parse charts into Python programs that can faithfully reproduce the original plots [48, 37, 46]. *Table-to-Markdown* converts tabular images into structured formats such as Markdown or HTML [18, 45, 24]. *SVG-to-Code* translates vector graphics into code representations [15, 42]. Such structured outputs facilitate downstream use and improve usability in real-world applications.

RL for Visual-to-Code Tasks. Despite its practical importance, visual-to-code remains challenging. Supervised fine-tuning (SFT) typically relies on large-scale, high-quality datasets [48, 49, 11], which are costly to curate. RL has been explored as an alternative, yet existing reward designs often fall into two extremes: (i) textual rule-based rewards [18], which score string-level or structural proxies in the text space without directly leveraging the visual evidence, and thus may introduce modality bias; and (ii) visual-encoder similarity-based rewards, such as DINO-based [34] similarity [47, 37, 31], which compare representations extracted by vision encoders but are often coarse-grained and offer limited interpretability. Motivated by these limitations, we propose Visual-ERM, a cross-modal reward model that provides fine-grained, interpretable, and task-agnostic feedback for Visual-to-Code.

6 Conclusion

We propose the Visual Equivalence Reward Model (Visual-ERM), a generative reward model that evaluates vision-to-code outputs in visual space, providing fine-grained, interpretable, and task-agnostic supervision. We further introduce VisualCritic-RewardBench to directly assess image-to-image discrepancy judgment across vision-to-code tasks. Experiments show that Visual-ERM is an effective supervisor for both reinforcement learning and test-time scaling, consistently improving vision-to-code performance across multiple tasks.

Acknowledgments

This work was supported by National Key R&D Program of China 2022ZD0161600, Shanghai Artificial lntelligence Laboratory, Hong Kong RGC TRS T41-603/20-R, the Centre for Perceptual and Interactive Intelligence (CPII) Ltd under the Innovation and Technology Commission (ITC)'s InnoHK. Dahua Lin is a PI of CPII under the InnoHK. This paper is also supported by National Science and Technology Major Project of China (Grant No. 2026ZD0128702)

References

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

[2] Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xiong-Hui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Rongyao Fang, 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, Li Ying Meng, Xuancheng Ren, Xin yi Ren, Sibo Song, Yu-Chen Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yihe Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Botao Zheng, Humen Zhong, Jingren Zhou, Fanxi Zhou, Jingren Zhou, Yuanzhi Zhu, and Keming Zhu. Qwen3-vl technical report. *ArXiv*, abs/2511.21631, 2025.

[3] 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. *arXiv preprint arXiv:2511.21631*, 2025.

[4] 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*, 2025.

[5] Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiao wen Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhen Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hong wei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kui-Jie Liu, Xiaoran Liu, Chen Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xing Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Rui Ze Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fen-Fang Zhou, Zaida Zhou, Jingming Zhuo, Yi-Ling Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. Internlm2 technical report. *ArXiv*, abs/2403.17297, 2024.

[6] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[7] 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. *arXiv preprint arXiv:2508.13587*, 2025.

[8] 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.

[9] Shengyuan Ding, Xinyu Fang, Ziyu Liu, Yuhang Zang, Yuhang Cao, Xiangyu Zhao, Haodong Duan, Xiaoyi Dong, Jianze Liang, Bin Wang, et al. Arm-thinker: Reinforcing multimodal generative reward models with agentic tool use and visual reasoning. *arXiv preprint arXiv:2512.05111*, 2025.

[10] Google. Gemini 3.1 Pro: A smarter model for your most complex tasks, February 2026.

[11] Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, et al. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pages 1834–1845, 2025.

[12] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv:2412.16720*, 2024.

[13] Lingjie Jiang, Shaohan Huang, Xun Wu, Yixia Li, Dongdong Zhang, and Furu Wei. Viscodex: Unified multimodal code generation via merging vision and coding models. *arXiv preprint arXiv:2508.09945*, 2025.

[14] Dahyun Kim, Chanjun Park, Sanghoon Kim, Wonsung Lee, Wonho Song, Yunsu Kim, Hyeonwoo Kim, Yungi Kim, Hyeonju Lee, Jihoo Kim, Changbae Ahn, Seonghoon Yang, Sukyung Lee, Hyunbyung Park, Gyoungjin Gim, Mikyoung Cha, Hwalsuk Lee, and Sunghun Kim. Solar 10.7b: Scaling large language models with simple yet effective depth up-scaling. *ArXiv*, abs/2312.15166, 2023.

[15] Jinke Li, Jiarui Yu, Chenxing Wei, Hande Dong, Qiang Lin, Liangjing Yang, Zhicai Wang, and Yanbin Hao. Unisvg: A unified dataset for vector graphic understanding and generation with multimodal large language models. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pages 13156–13163, 2025.

[16] Lei Li, Yuancheng Wei, Zhihui Xie, Xuqing Yang, Yifan Song, Peiyi Wang, Chenxin An, Tianyu Liu, Sujian Li, Bill Yuchen Lin, et al. Vl-rewardbench: A challenging benchmark for vision-language generative reward models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24657–24668, 2025.

[17] Renhao Li, Jianhong Tu, Yang Su, Hamid Alinejad-Rokny, Derek F Wong, Junyang Lin, and Min Yang. One model to critique them all: Rewarding agentic tool-use via efficient reasoning. *arXiv preprint arXiv:2510.26167*, 2025.

[18] Jun Ling, Yao Qi, Tao Huang, Shibo Zhou, Yanqin Huang, Jiang Yang, Ziqi Song, Ying Zhou, Yang Yang, Heng Tao Shen, et al. Table2latex-rl: High-fidelity latex code generation from table images via reinforced multimodal language models. *arXiv preprint arXiv:2509.17589*, 2025.

[19] Ziyu Liu, Zeyi Sun, Yuhang Zang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. Visual-rft: Visual reinforcement fine-tuning. *arXiv preprint arXiv:2503.01785*, 2025.

[20] Ziyu Liu, Yuhang Zang, Shengyuan Ding, Yuhang Cao, Xiaoyi Dong, Haodong Duan, Dahua Lin, and Jiaqi Wang. Spark: Synergistic policy and reward co-evolving framework. *arXiv preprint arXiv:2509.22624*, 2025.

[21] Ahmed Masry, Xuan Long Do, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. In *Findings of the association for computational linguistics: ACL 2022*, pages 2263–2279, 2022.

[22] Minesh Mathew, Viraj Bagal, Rubèn Tito, Dimosthenis Karatzas, Ernest Valveny, and CV Jawahar. Infographicvqa. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 1697–1706, 2022.

[23] Minesh Mathew, Dimosthenis Karatzas, and CV Jawahar. Docvqa: A dataset for vqa on document images. In *Proceedings of the IEEE/CVF winter conference on applications of computer vision*, pages 2200–2209, 2021.

[24] Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, et al. Mineru2. 5: A decoupled vision-language model for efficient high-resolution document parsing. *arXiv preprint arXiv:2509.22186*, 2025.

[25] OpenAI. Openai o3-mini system card, 2025.

[26] Linke Ouyang, Yuan Qu, Hongbin Zhou, Jiawei Zhu, Rui Zhang, Qunshu Lin, Bin Wang, Zhiyuan Zhao, Man Jiang, Xiaomeng Zhao, et al. Omnidocbench: Benchmarking diverse pdf document parsing with comprehensive annotations. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 24838–24848, 2025.

[27] Hao Peng, Yunjia Qi, Xiaozhi Wang, Zijun Yao, Bin Xu, Lei Hou, and Juanzi Li. Agentic reward modeling: Integrating human preferences with verifiable correctness signals for reliable reward systems. *arXiv preprint arXiv:2502.19328*, 2025.

[28] Jake Poznanski, Aman Rangapur, Jon Borchardt, Jason Dunkelberger, Regan Huff, Daniel Lin, Christopher Wilhelm, Kyle Lo, and Luca Soldaini. olmocr: Unlocking trillions of tokens in pdfs with vision language models. *arXiv preprint arXiv:2502.18443*, 2025.

[29] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PmLR, 2021.

[30] Juan A Rodriguez, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, David Vazquez, Christopher Pal, and Marco Pedersoli. Starvector: Generating scalable vector graphics code from images. *arXiv preprint arXiv:2312.11556*, 2023.

[31] Juan A Rodriguez, Haotian Zhang, Abhay Puri, Aarash Feizi, Rishav Pramanik, Pascal Wichmann, Arnab Mondal, Mohammad Reza Samsami, Rabiul Awal, Perouz Taslakian, et al. Rendering-aware reinforcement learning for vector graphics generation. *arXiv preprint arXiv:2505.20793*, 2025.

[32] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[33] Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2Code: Benchmarking multimodal code generation for automated front-end engineering. In *ACL*, 2025.

[34] Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025.

[35] Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, et al. Openai gpt-5 system card. *arXiv preprint arXiv:2601.03267*, 2025.

[36] Qiushi Sun, Jingyang Gong, Yang Liu, Qiaosheng Chen, Lei Li, Kai Chen, Qipeng Guo, Ben Kao, and Fei Yuan. Januscoder: Towards a foundational visual-programmatic interface for code intelligence. *arXiv preprint arXiv:2510.23538*, 2025.

[37] 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. *arXiv preprint arXiv:2508.17608*, 2025.

[38] Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, et al. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*, 2025.

[39] Yibin Wang, Yuhang Zang, Hao Li, Cheng Jin, and Jiaqi Wang. Unified reward model for multimodal understanding and generation. *arXiv preprint arXiv:2503.05236*, 2025.

[40] Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *Advances in Neural Information Processing Systems*, 37:113569–113697, 2024.

[41] Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, et al. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *arXiv preprint arXiv:2406.09961*, 2024.

[42] Yiying Yang, Wei Cheng, Sijin Chen, Xianfang Zeng, Fukun Yin, Jiaxu Zhang, Liao Wang, Gang Yu, Xingjun Ma, and Yu-Gang Jiang. Omnisvg: A unified scalable vector graphics generation model. *arXiv preprint arXiv:2504.06263*, 2025.

[43] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason E. Weston. Self-rewarding language models. *ArXiv*, abs/2401.10020, 2024.

[44] Yuhang Zang, Xiaoyi Dong, Pan Zhang, Yuhang Cao, Ziyu Liu, Shengyuan Ding, Shenxi Wu, Yubo Ma, Haodong Duan, Wenwei Zhang, et al. Internlm-xcomposer2. 5-reward: A simple yet effective multi-modal reward model. *arXiv preprint arXiv:2501.12368*, 2025.

[45] Jiarui Zhang, Yuliang Liu, Zijun Wu, Guosheng Pang, Zhili Ye, Yupei Zhong, Junteng Ma, Tao Wei, Haiyang Xu, Weikai Chen, et al. Monkeyocr v1. 5 technical report: Unlocking robust document parsing for complex patterns. *arXiv preprint arXiv:2511.10390*, 2025.

[46] Zhihan Zhang, Yixin Cao, and Lizi Liao. Enhancing chart-to-code generation in multimodal large language models via iterative dual preference learning. *arXiv preprint arXiv:2504.02906*, 2025.

[47] Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. Vincicoder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning. *arXiv preprint arXiv:2511.00391*, 2025.

[48] 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. *arXiv preprint arXiv:2501.06598*, 2025.

[49] Xu Zhong, Elaheh ShafieiBavani, and Antonio Jimeno Yepes. Image-based table recognition: data, model, and evaluation. *arXiv preprint arXiv:1911.10683*, 2019.

[50] Banghua Zhu, Evan Frick, Tianhao Wu, Hanlin Zhu, and Jiantao Jiao. Starling-7b: Improving llm helpfulness & harmlessness with rlaif, November 2023.

Appendix

In the appendix, we provide additional materials to support the main paper and facilitate a deeper understanding of Visual-ERM.

First, in Sec. A, we provide detailed descriptions of the models, datasets, benchmark construction, experimental setups and computational efficiency throughout the paper, aiming to ensure reproducibility.

Second, Sec. B presents additional experimental analyses and ablation studies, including evaluations on general VQA benchmarks, investigations of multi-task reward modeling, robustness analyses of different judge models, further studies on reward design, test-time scaling and so on.

Third, in Sec. C, we list the full prompt templates used in our experiments, covering reward model data generation, Visual-ERM inference across different vision-to-code tasks, and the evaluation protocol of VC-RewardBench.

Finally, Sec. D provides representative qualitative case studies from chart-to-code, table-to-markdown, and SVG-to-code tasks, illustrating typical visual discrepancies and the fine-grained feedback produced by Visual-ERM.

A More Details

A.1 Models

This work involves multiple models of different roles and scales. For clarity and reproducibility, we summarize the models used throughout the paper and their corresponding usage in each stage of the pipeline.

Visual-ERM (Reward Model). We build Visual-ERM on top of Qwen3-VL-8B-Instruct [3]. While Qwen3-VL-8B-Instruct already exhibits strong multimodal perception and reasoning abilities, we find that it is not a reliable *image-to-image discrepancy* judge out of the box—especially for structured visuals where the key differences are often text- and layout-centric. This limitation is empirically reflected in Sec. 4.2, where the base model shows poor discrimination on fine-grained visual mismatches.

Starting from this 8B backbone, we train Visual-ERM on a large-scale reward-modeling corpus spanning three vision-to-code domains: Chart-to-Code (104K), Table-to-Markdown (125K), and SVG-to-Code (111K). This training equips Visual-ERM with task-agnostic yet fine-grained judgment capabilities. Notably, despite its relatively small size, Visual-ERM substantially outperforms Qwen3-VL-235B-Instruct as a judge and achieves performance competitive with strong proprietary models, demonstrating that targeted reward model training can be more effective than simply scaling up a general-purpose LVLM.

Policy models for RL. For reinforcement learning, we consider two types of policy backbones. The first is Qwen3-VL-8B-Instruct, which serves as our default policy model across tasks. In this setting, we use Visual-ERM as the reward model and optimize Qwen3-VL-8B-Instruct with RL to improve its vision-to-code generation quality.

In addition, to test whether Visual-ERM can also improve stronger specialized parsers, we use VinciCoder-8B-SFT [47] as an alternative policy backbone on Chart-to-Code and SVG-to-Code. VinciCoder-8B-SFT provides a stronger supervised baseline for these tasks, and applying Visual-ERM-guided RL on top of it allows us to evaluate the reward model's transferability and its ability to refine already-competitive policies.

For Table-to-Markdown, we do not include VinciCoder-8B-SFT because it does not have a comparable task-specific SFT baseline for tables; therefore, we only use Qwen3-VL-8B-Instruct as the policy model for RL in this setting.

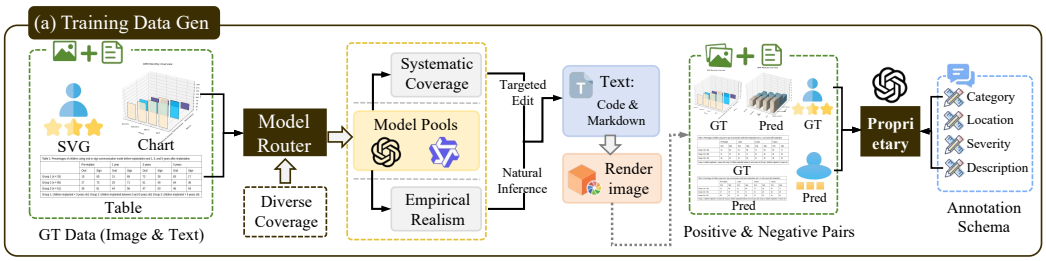


Figure 6: **Training Data Generation.** We construct our Visual-ERM's training data as shown in Figure.

A.2 Datasets

We use multiple datasets throughout the paper for reward-model training, RL policy optimization, and benchmark construction. For completeness, we summarize them here. The annotations and training data will be released publicly in the future, and qualitative examples are provided in Sec. D.

Reward-model training data. We first collect a pool of images and their corresponding ground-truth annotations from public vision-to-code datasets spanning charts, tables, and SVG-style graphics [48, 7, 13, 15, 30]. However, these raw pairs are not directly suitable for training a reward model, because Visual-ERM requires supervision on *image-to-image discrepancies* rather than only paired input-output examples. We therefore construct training instances by synthesizing realistic error patterns and converting them into *rendered* visual differences.

Concretely, we create imperfect predictions via two complementary routes: (i) *error injection* on ground-truth annotations using a strong yet cost-effective model (GPT-5-mini) as well as Qwen3-VL-235B-Instruct, and (ii) *model-generated predictions* by running Qwen3-VL-8B-Instruct on a subset of ground-truth images to obtain naturally occurring parsing errors. We then render each prediction back into an image, forming an (Original Image, Generated Image) pair that reflects the types of distortions encountered in practice. Given each image pair, we use GPT-5-mini to produce structured annotations, including fine-grained error descriptions (category, location, severity), as illustrated in Sec. D. This procedure yields three reward-modeling datasets: 104K instances for Chart-to-Code, 125K for Table-to-Markdown, and 111K for SVG-to-Code. The trainning data generation pipeline is shown in Fig. 6.

RL training data. The RL data are drawn from the same underlying sources, but require substantially less processing. Unlike reward-model training, RL does not require free-form error annotations: during training, Visual-ERM directly compares the original image with the re-rendered prediction and outputs an image-to-image reward. As a result, we only need the ground-truth images, and in many cases the ground-truth parsing text is not necessary. In our experiments, we use 11K images for Chart-to-Code RL, 40K for Table-to-Markdown RL, and 10K for SVG-to-Code RL.

VC-RewardBench construction data. We construct VC-RewardBench using a similar discrepancy-centric philosophy, but with substantially stricter quality control to ensure reliability as an evaluation benchmark. We begin with a candidate pool of 4.5K image pairs and obtain annotations from three state-of-the-art proprietary models: GPT-5-mini [35], Gemini2.5-Pro, and Gemini3-Pro [8]. We then manually consolidate, filter, and curate the collected annotations to resolve inconsistencies and remove ambiguous or low-quality cases. The final benchmark contains 1,335 carefully vetted examples, which we use to evaluate fine-grained image-to-image judging ability in structured visual domains.

A.3 VC-RewardBench Statistics

This section reports detailed statistics of VC-RewardBench. The benchmark contains 1,335 curated examples spanning three structured-visual domains: 595 chart instances, 442 SVG instances, and 298 table instances.

Beyond instance counts, VC-RewardBench provides fine-grained error annotations that characterize the discrepancy types between the original image and the re-rendered prediction. For the chart

subset, the annotations include 231 `text_error`, 619 `style_error`, 462 `data_error`, and 62 `structure_error`. For the table subset, we annotate 884 `text_error`, 284 `numeric_error`, and 185 `layout_error`. For the SVG subset, the error annotations contain 151 `style_error`, 218 `structure_error`, 360 `shape_error`, and 12 `text_symbol_error`.

Overall, these statistics indicate that VC-RewardBench covers a diverse range of realistic failure modes. In particular, chart examples are dominated by style- and data-related discrepancies, tables primarily exhibit text-centric recognition errors with a non-trivial portion of numeric and layout issues, and SVG examples emphasize geometric `shape_error` and structural composition mismatches. This diversity makes VC-RewardBench a challenging and diagnostic benchmark for evaluating fine-grained image-to-image judging capability.

A.4 Experimental Setups

We implement supervised training and RL fine-tuning using LLaMA-Factory and the VERL framework. For training Visual-ERM, we use a learning rate of 1×10^{-5} with a batch size of 32. We train for 3 epochs over the reward-modeling dataset. For reinforcement learning, we adopt GRPO with a learning rate of 1×10^{-6} and a batch size of 256. For each prompt, we sample 8 rollouts per GRPO update step to estimate the policy gradients.

A.5 Computational Efficiency and Training Latency

A primary concern in scaling reinforcement learning (RL) is the computational overhead introduced by reward models, particularly as their parameter count increases. We systematically benchmark the training latency of three reward supervision paradigms under identical experimental configurations: (1) **Text-based rules** (e.g., TEDS); (2) **Vision-encoder similarity** (DINOv2-large, 0.3B), following the framework in [47]; and (3) our proposed Visual-ERM (8B).

While text-based rewards are predictably the most efficient at 0.15 hours per step, they yield the least effective policy performance. Notably, Visual-ERM demonstrates a distinct computational advantage: despite its 8B parameter scale, it achieves a per-step latency of 0.17 hours, outperforming the 0.24 hours required by the 0.3B DINOv2-large baseline. This efficiency gap arises because DINO-based rewards [47], necessitate segmenting high-resolution images into multiple sub-patches to ensure scoring precision, which introduces significant computational overhead. In contrast, Visual-ERM's unified multimodal architecture processes structured visuals natively, providing high-fidelity reward signals without the need for additional pre-processing, thereby maintaining high training throughput.

B Additional Experimental Analyses

B.1 Evaluation on General VQA Benchmarks

While Visual-ERM substantially improves the policy model on a diverse set of vision-to-code tasks, it is important to examine whether these gains come with any trade-offs on broader multimodal capabilities. In particular, reinforcement learning with task-specific rewards may induce distributional shifts or over-specialization, potentially degrading performance on general-purpose visual question answering (VQA) benchmarks.

To assess the generality of the learned policy, we evaluate the RL-trained model on a suite of established VQA benchmarks that are closely related to the structured visual domains considered in this work. Concretely, we select multiple chart- and document-centric benchmarks that require fine-grained visual perception, text recognition, and cross-modal reasoning, thereby serving as a diagnostic for whether Visual-ERM-guided optimization preserves general visual understanding skills beyond code generation. These benchmarks include ChartQA [21], CharXiv_DQ [40], CharXiv_RQ [40], DocVQA [23] and InfoVQA [22].

Tab. 8 summarizes the results. Overall, Visual-ERM-guided RL does not degrade general VQA performance: the average score remains essentially stable and even shows slight improvements for two of the three RL policies. Concretely, compared to the base Qwen3-VL-8B-Instruct (AVG 78.3), RL on Chart-to-Code reaches 78.4 (+0.1) and RL on SVG-to-Code reaches 78.5 (+0.2), while RL on Table-to-Markdown is comparable at 78.1 (-0.2). At the benchmark level, the most consistent change is on CharXiv-RQ, where all RL variants improve over the base (46.0 $\rightarrow$ 47.2 / 46.7 / 46.4),

Table 8: **Results on general VQA benchmarks.** We evaluate the policy model trained with Visual-ERM-guided RL on multiple general-purpose benchmarks (with an emphasis on chart- and document-centric VQA) to examine whether its improved vision-to-code performance comes at the cost of general multimodal capability.

Models	ChartQA_TEST	CharXiv_DQ	CharXiv_RQ	DocVQA_VAL	InfoVQA_VAL	AVG
Qwen3-VL-8B-Instruct	82.9	83.8	46.0	95.6	83.1	78.3
+ RL on Chart-to-Code	82.6	83.5	47.2	95.6	83.1	78.4
+ RL on Table-to-Markdown	81.5	83.8	46.7	95.5	83.1	78.1
+ RL on SVG-to-Code	83.2	83.5	46.4	95.8	83.6	78.5

Table 9: **Effect of Multi-Task Data Mixing.** We compare reward models trained on mixed multi-task data vs. single-task data.

Model	Chart			Table			SVG			AVG		
	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c
Visual-ERM												
+ Chart-Data-Only	40.1	42.6	58.1	17.2	20.4	10.5	11.3	15.3	50.1	26.3	29.3	23.7
+ Table-Data-Only	17.1	20.1	49.3	53.6	54.9	68.8	14.1	19.0	55.2	31.7	34.2	56.6
+ SVG-Data-Only	16.4	21.0	56.0	11.4	17.6	37.0	26.4	31.0	57.3	17.5	22.6	44.2
+ Mix-Data	39.9	42.8	61.2	56.4	57.6	74.8	28.3	32.6	59.6	42.1	44.7	58.4

suggesting better fine-grained reasoning over chart-centric queries. Meanwhile, document-centric performance is preserved: DocVQA remains nearly unchanged (95.6 $\rightarrow$ 95.6 / 95.5 / 95.8), and InfoVQA is stable with a small gain for the SVG RL policy (83.1 $\rightarrow$ 83.6). These results indicate that the gains on vision-to-code do not arise from sacrificing general multimodal competence; instead, Visual-ERM provides a largely non-destructive learning signal that improves structured visual parsing while maintaining broad chart/document understanding.

B.2 Evaluation of the Effect of Multi-Task Data Mixing

B.2.1 Effect of Multi-Task Data Mixing on VC-RewardBench

Visual-ERM is trained to support multiple tasks that share partial failure modes but also exhibit task-specific characteristics. To examine whether joint training on mixed data is beneficial or induces negative transfer, we ablate the reward model's training data composition by comparing a unified model trained on all three data sources with task-specific variants trained on a single data type. Results on VC-RewardBench are reported in Tab. 9.

Mixing chart, table, and SVG data yields the best overall performance. We attribute this to cross-task transfer of error patterns (e.g., recognition and layout behaviors learned from tables can generalize to charts due to overlapping failure modes) leading to a mutually beneficial effect under joint training. In Sec. B.2.2, we further validate this finding by comparing the downstream RL performance of reward models trained on different data sources.

As shown in Tab. 9, reward models trained on single-task data exhibit a strong task bias. For example, Chart-Data-Only performs well on the Chart subtask ($F1_h = 40.1, F1_s = 42.6$), but performs much worse on Table and SVG tasks (e.g., Table $F1_h = 17.2$, SVG $F1_h = 11.3$). Similarly, although Table-Data-Only achieves strong results on the Table subtask ($F1_h = 53.6$, $S_c = 68.8$), it generalizes poorly to Chart ($F1_h = 17.1$). In contrast, the mixed-data RM (Mix-Data) achieves more balanced and consistently strong performance across all three task categories. It maintains nearly the same performance on Chart ($F1_h = 39.9$ vs. 40.1), while significantly improving on Table and SVG (Table: $F1_h = 56.4$, SVG: $F1_h = 28.3$). As a result, it attains the best average performance (AVG: $F1_h = 42.1, F1_s = 44.7, S_c = 58.4$). These results indicate that multi-task training does not introduce obvious negative transfer; instead, it provides positive transfer by sharing structured understanding and code generation abilities, leading to a more generalizable reward model.

B.2.2 Effect of Multi-Task Data Mixing on RL

To further validate this phenomenon in a practical setting, we apply RMs trained with different datasets to actual RL training and evaluate the effects in real usage. We select the Table-to-Markdown task for this experiment, and the results are shown in Tab. 10. As shown in Tab. 10, both the Table-only RM and the mixed-data RM lead to stable improvements in the Table-to-Markdown RL scenario. Compared with the baseline, RM (table-data-only) improves TEDS and TEDS-S on both benchmarks

Table 10: **Ablation Study of Multi-Task Data Mixing**. We perform RL on the table parsing task using different reward models: one reward model is trained only on table data, while the other reward model is jointly trained on table, chart, and SVG data. TEDS-S denotes TEDS-Structure-Only, and TA represents the table subtask of olmOCRBench. For Edit-Dist, where lower values indicate better performance, we use (100 − Edit-Dist) when computing the Average.

Model	OmniDocBench			OmniDocBench-v1.5			olmOCRBench	Avg↑
	TEDS↑	TEDS-S↑	Edit-Dist↓	TEDS↑	TEDS-S↑	Edit-Dist↓	TA↑	
Baseline								
Qwen3-VL-8B-Instruct	78.9	83.9	23.2	72.7	77.2	26.9	75.3	76.8
RL with Different Reward Model								
+ RM (table-data-only)	79.5	84.4	21.6	74.4	79.4	25.0	**79.4**	78.6
Δ	**+0.6**	**+0.5**	**+1.6**	**+1.7**	**+2.2**	**+1.9**	**4.1**	**+1.8**
+ RM (mix-data, Visual-ERM)	**81.4**	**86.3**	**20.7**	**75.4**	**80.4**	**24.2**	78.1	**79.5**
Δ	**+2.5**	**+2.4**	**+2.5**	**+2.7**	**+3.2**	**+2.7**	**+2.8**	**+2.7**

and reduces Edit-Dist, resulting in an overall average gain of +1.8. More importantly, the mixed-data RM achieves further improvements over the single-task RM on most metrics. On OmniDocBench, it reaches $TEDS = 81.4$ and $TEDS\text{-}S = 86.3$, while further reducing Edit-Dist to 20.7. On olmOCRBench-v1.5, it also achieves higher TEDS/TEDS-S and lower Edit-Dist. Overall, it yields an average improvement of +2.7. Although the mixed RM shows a slight drop on the TA metric compared to the Table-only RM (78.1 vs. 79.4), the overall gains are larger, suggesting that a richer multi-task reward signal provides a more robust optimization direction for RL, improving both final performance and generalization on real Table-to-Markdown tasks.

B.3 Ablation on Different Judge Models

VC-RewardBench evaluates a reward model's ability to identify fine-grained image-to-image discrepancies and vision-to-code reconstruction errors. In addition to objective metrics (e.g., scalar scores, types and severities), VC-RewardBench also includes *free-form* discrepancy descriptions. Since these descriptions are open-ended and may use different wording to refer to the same underlying issue, an exact string match is insufficient for reliable evaluation.

LLM-assisted matching protocol. To evaluate such free-form outputs in a scalable and consistent manner, we employ an LLM-as-Judge to perform *error-level alignment* between the reward model's predicted error list and the ground-truth error list. Specifically, given the predicted errors and the annotated errors, the judge determines for each predicted item whether it (i) correctly matches a ground-truth error, (ii) is a hallucinated/unsupported claim, or (iii) corresponds to an error that exists but is described incorrectly. Symmetrically, the judge also identifies ground-truth errors that are missed by the model. Based on this alignment, we compute two variants of F1: (i) a strict matching criterion ($F1_h$, *hard*) that requires close semantic and attribute-level agreement, and (ii) a relaxed criterion ($F1_s$, *soft*) that tolerates minor paraphrases or partial matches, reflecting a more lenient notion of equivalence.

Why judge ablations matter. Using an LLM introduces a potential dependency on the judge's calibration and reasoning style. However, in VC-RewardBench the judge is only asked to perform relatively constrained *matching* (rather than open-ended preference ranking), which we hypothesize to be less sensitive to the specific judge model. To validate this, we rerun the same Visual-ERM predictions on VC-RewardBench while varying the judge among several widely-used API models, and report the resulting scores in Tab. 11.

Results and analysis. Tab. 11 shows that the evaluation is highly stable across judges. First, the S_c is identical for all judges (e.g., Chart: 61.2, Table: 74.8, SVG: 59.6, AVG: 58.4), as expected since it is computed deterministically and does not rely on the LLM matcher. Second, the judge-dependent metrics ($F1_h$ and $F1_s$) vary only modestly: on average, $F1_h$ ranges from 40.8 to 43.0 (a span of 2.2 points) and $F1_s$ ranges from 42.4 to 45.1 (a span of 2.7 points) across all judges. At the task level, the variability remains limited—for instance, Table $F1_s$ ranges from 55.7 to 57.8—indicating that different judges largely agree on which predicted errors are correct versus hallucinated or missing.

Table 11: **Ablation on LLM Judges.** VC-RewardBench uses LLM-assisted judging to verify answer correctness. To assess the robustness of this evaluation protocol, we compare multiple LLM-as-Judge models and report the resulting consistency across judges.

Judger	Chart			Table			SVG			AVG		
	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c	$F1_h$	$F1_s$	S_c
GPT-5-mini	39.9	42.8	61.2	56.4	57.6	74.8	28.3	32.6	59.6	42.1	44.7	58.4
GPT-5.2	40.9	45.4	61.2	55.0	57.1	74.8	24.5	31.6	59.6	40.8	45.1	58.4
Gemini2.5-Pro	40.8	42.1	61.2	54.8	55.7	74.8	27.1	28.7	59.6	41.2	42.4	58.4
Gemini3-Flash	41.5	43.7	61.2	56.5	57.8	74.8	28.5	32.1	59.6	43.0	44.9	58.4

Table 12: **Ablation Study of Reward Design**. *RSR* denotes the Render-Success Reward. TEDS-S denotes TEDS-Structure-Only, and TA represents the table subtask of olmOCRBench. For Edit-Dist, where lower values indicate better performance, we use (100 − Edit-Dist) when computing the Average.

Model	OmniDocBench			OmniDocBench-v1.5			olmOCRBench	Avg↑
	TEDS↑	TEDS-S↑	Edit-Dist↓	TEDS↑	TEDS-S↑	Edit-Dist↓	TA↑	
Qwen3-VL-8B-Instruct	78.9	83.9	23.2	72.7	77.2	26.9	75.3	76.8
GRPO (w/o RSR)	80.9	85.6	21.8	75.2	79.9	25.9	79.0	79.0
GRPO (w RSR)	81.4	86.3	20.7	75.4	80.4	24.2	78.1	79.5

Overall, these results suggest that VC-RewardBench is not overly sensitive to the choice of LLM judge. Since the judge is used for structured error-to-error matching under well-specified criteria, replacing the judge model yields consistent conclusions, supporting the robustness of our evaluation protocol.

B.4 Ablation Study on Reward Design

We investigate two reward designs. The first directly uses the scalar score produced by the reward model as the reward signal. To improve training stability, the second design augments the reward with an additional render-success reward that indicates whether the generated output can be successfully rendered, which is similar to a format reward. To combine the render-success reward with the reward-model score in a principled manner, we further apply score normalization before aggregation.

We conduct an ablation study comparing these two reward formulations and report the results in Tab. 12. While both designs yield comparable performance gains, the formulation incorporating a render-success reward achieves a slightly larger improvement, which we attribute to its enhanced training stability. Specifically, we observe that during the initial stages of training, the model occasionally produces code that cannot be successfully rendered, leading to intermittent zero rewards. However, the policy rapidly learns to navigate these formatting constraints, allowing the optimization focus to transition quickly from basic renderability to the primary objective of preserving fine-grained visual characteristics. This phased learning trajectory ensures that the model effectively prioritizes structural and semantic fidelity once the fundamental format requirements are mastered.

B.5 More Test-Time Scaling Results

In the main paper, we evaluate reflection-based test-time scaling (TTS) on Chart-to-Code. To further validate the effectiveness and generality of this inference-time refinement strategy, we additionally conduct TTS experiments on SVG-to-Code, with results reported in Tab. 13. We report SSIM and CLIPScore (higher is better) and LPIPS (lower is better).

The table contains two independent comparisons. The first compares Qwen3-VL-8B-Instruct with and without Visual-ERM-guided reflection. Reflection yields consistent improvements across all metrics: CLIPScore increases from 73.3 to 74.6, SSIM remains comparable (60.9 vs. 60.4), and LPIPS decreases from 60.0 to 58.1 (improved). Overall, the aggregated score improves from 64.2 to 65.2 (+1.0), demonstrating that Visual-ERM can provide actionable feedback for inference-time refinement even without RL.

Prompt for Distillation Reward Modeling Datasets (Chart) — Part1

```
You are an **Experienced Specialist for Data Visualization**. You will be provided with **two images**:
1. **Original Image**: a chart rendered using ground-truth Matplotlib code.
2. **Generated Image**: a chart rendered using AI-generated Matplotlib code for the **Original Image**.
Your task is to **compare the Generated Image against the Original Image** and identify **all visual discrepancies** that affect:
* correctness of the data,
* layout / structure, or
* aesthetic consistency (when it impacts readability or overall fidelity).
You are evaluating **only what is visible in the two images**, not the underlying code.
---
### Step 1: Compare the two images
Carefully compare the Original and Generated images:
* Look at chart types, subplots, axes, data shapes, colors, legends, labels, ticks, and annotations.
* Focus on differences that a developer would need to fix in their Matplotlib code.
If there is **no meaningful difference** between the two images (ignoring tiny pixel-level differences such as anti-aliasing), then you should report **zero errors**.
---
### Step 2: Classify each discrepancy into one category
For **every discrepancy you find**, assign it to **exactly one** of the following categories:
##### A. `structure_error` (Layout & Chart Type)
**Definition:** Problems with the overall structure or composition of the figure.
**Typical cases:**
1. Wrong chart type (e.g., pie vs. donut, bar vs. line, radar vs. line).
2. Wrong number of subplots (e.g., 1 subplot instead of 2).
3. Subplots arranged in wrong rows/columns or in the wrong positions.
4. Subplots missing or extra.
5. Axes that should be shared/aligned in the Original are not shared/aligned in the Generated (or vice versa).
##### B. `data_error` (Content & Geometry)
**Definition:** The visualized data itself is incorrect or distorted.
**Typical cases:**
1. **Geometry mismatch:** The shape of the curve, polygon, or bar pattern is clearly different (e.g., an increasing line becomes flat or decreasing).
2. **Value distortion:** Bar heights, pie slice angles, or point positions do not match the original trend or relative ratios.
3. **Scale / limits mismatch:** Different xlim/ylim or axis scales that significantly change how the data appears.
4. **Missing / extra data:** Missing series, hallucinated extra lines, wrong number of groups or categories.
##### C. `text_error` (Labels & Annotations)
**Definition:** Problems with any visible text in the chart.
Text includes: titles, subtitles, axis labels, tick labels, legend text, annotations, and text boxes.
**Typical cases:**
1. Missing or incorrect titles, axis labels, tick labels, or legend entries.
2. Misplaced labels (e.g., attached to the wrong axis, wrong subplot, or wrong data series).
3. Text overlapping with other elements in a way that harms readability.
4. Obvious typos or formatting differences that clearly reduce readability (e.g., extremely small font where the original is normal).
##### D. `style_error` (Aesthetics & Appearance)
**Definition:** Visual style differences that may not directly change the data values, but affect the overall look and clarity.
**Typical cases:**
1. **Color mismatch:** Different color palette or clearly wrong color mapping for data series / categories.
2. **Markers & lines:** Wrong line style (dashed vs. solid), marker shape, or line width.
3. **Grid / background:** Missing or extra gridlines, wrong background color, frame visibility changes.
4. Other stylistic differences that noticeably change the visual appearance, even if the data is still correct.
---
### Step 3: Assign severity for each error
For each error, assign a **severity**:
**1 (Minor)**: Purely aesthetic or very small differences. The chart remains correct and easy to understand
  (e.g., slightly different shade of color, small font size difference that doesn't hurt readability).
**2 (Moderate)**: Affects readability or requires correction for confident use
  (e.g., missing axis label, legend partially blocking data, noticeably confusing color choices).
**3 (Critical)**: The chart is **factually wrong or structurally broken**, or seriously misleading
  (e.g., wrong chart type, missing subplot, reversed trend, wrong scale that changes interpretation, missing key data series).
---
```

Figure 7: **Prompt for distillation.** The prompt used to distill GPT-5-mini to construct reward-modeling training data for Visual-ERM. We show the Chart-specific prompt here; due to its length, this table includes only the first part.

Prompt for Distillation Reward Modeling Datasets (Chart) — Part2

```
### Step 4: Output format (very important)
You must output **ONLY** a single valid JSON object, with **no additional text**, **no explanations**, and **no Markdown code fences**.

Use exactly this JSON schema:
{
  "structure_error_count": int,
  "data_error_count": int,
  "text_error_count": int,
  "style_error_count": int,
  "errors": [
    {
      "category": "structure_error | data_error | text_error | style_error",
      "severity": 1 | 2 | 3,
      "location": "Specific location (e.g., 'Left subplot title', 'Red line data', 'Legend')",
      "description": "Concise description of the error."
    }
  ]
}

**Additional constraints:**
1. If a category has **no errors**, its `*_error_count` **must be 0**.
2. The `errors` list can be **empty** if the Generated Image is visually consistent with the Original Image (ignoring tiny pixel-level differences).
3. The numeric counts (`*_error_count`) must exactly match the number of errors you report for each corresponding category in the `errors` list.
```

Figure 8: **Prompt for distillation.** The prompt used to distill GPT-5-mini to construct reward-modeling training data for Visual-ERM. We show the Chart-specific prompt here; due to its length, this table includes only the second part.

Model	UniSVG(ISVGEN)			
	SSIM↑	LPIPS↓	CLIP↑	**Score↑**
Qwen3-VL-8B-Instruct	60.9	60.0	73.3	64.2
+ Reflection (Visual-ERM)	60.4	58.1	74.6	65.2
Qwen3-VL-8B-Instruct (+RL on Visual-ERM)	64.3	49.7	77.5	69.4
+ Reflection (Visual-ERM)	63.9	49.2	78.1	69.8

Table 13: **Test-Time Scaling on SVG-to-Code**. We evaluate TTS on the *SVG-to-Code* task. Leveraging Visual-ERM as an evaluator, the model performs multiple rounds of self-reflection and revision, leading to improved parsing performance.

The second comparison evaluates TTS on the Visual-ERM-guided RL policy (Qwen3-VL-8B-Instruct + RL on Visual-ERM). In this setting, applying reflection further boosts performance: LPIPS stays essentially stable (49.7 vs. 49.2, improved), and CLIPScore increases from 77.5 to 78.1. The overall score rises from 69.4 to 69.8 (+0.4). These results indicate that reflection provides additional gains on top of RL, and the RL-trained policy benefits from a stronger starting point, achieving the best absolute performance in both CLIPScore and overall score.

B.6 Scalability of Visual-ERM

Discrepancy Patterns. The primary categories defined in our training are macro-level abstractions that encompass nearly the entire spectrum of errors in visual-to-code tasks. We observe that almost all fine-grained discrepancies, including those not explicitly seen during training, are effectively combinations or specific instances of these core failure modes. By grounding the reward model in these fundamental dimensions, Visual-ERM develops a generalized capability to identify and penalize diverse error patterns regardless of their novelty.

Our data generation strategy further reinforces this adaptability through two synergistic modes: **1) Systematic Coverage (Targeted Edit):** We utilize high-capacity models to deliberately perturb structured representations, ensuring that the reward model maps the complete theoretical space of structural and semantic failures. **2) Empirical Realism (Natural Inference):** We sample authentic errors directly from weaker models to align the training distribution with the messy, unpredictable failure modes encountered in practice.

Together, these modes ensure that Visual-ERM does not merely memorize a fixed list of labels but instead masters the underlying logic of visual-to-code equivalence. This allows the model to remain robust and precise when evaluating complex, hybrid, or previously unencountered discrepancies.

Broadening Task Scope. While our current evaluation focuses on structured visuals such as charts, tables, and SVGs, Visual-ERM is designed to capture **atomic discrepancy patterns** that are fundamentally shared across nearly all vision-to-code domains. We argue that the core challenges in tasks like UI reconstruction or mathematical diagramming are not unique, but are instead composed of the same structural and semantic building blocks mastered by our model. For instance, the spatial layout and color fidelity critical for UI reconstruction are directly mirrored in our evaluation of complex chart and table structures.

In essence, Visual-ERM does not just learn task-specific rules; it learns the universal rules of visual-to-code equivalence. Because these "unseen" tasks rely on the same underlying principles of preserving shape, color, and structure, Visual-ERM can be naturally extended to a wider range of vision-to-code scenarios without requiring fundamental architectural changes.

C Prompts

Visual-ERM relies on several designed prompting templates at multiple stages of the pipeline. Specifically, we use prompts for: (i) distilling GPT-5-mini to construct reward-modeling training data for Visual-ERM, (ii) Visual-ERM inference across different vision-to-code tasks (Chart-to-Code, Table-to-Markdown, and SVG-to-Code), and (iii) the LLM-assisted judging protocol used in VC-RewardBench for evaluating subjective, free-form error descriptions. We provide the full prompts in this section for reproducibility.

C.1 Distillation Prompt

To train Visual-ERM, we first build a reward-modeling dataset by distilling GPT-5-mini. Each training instance consists of an *image pair*: an **Original Image** and a **Generated Image** rendered from a candidate output (e.g., model-generated code/markdown). The distillation target is a structured comparison: the model must identify and describe *all visually meaningful discrepancies* between the two images, focusing strictly on what is observable in the rendered results rather than the underlying code.

We construct these image pairs via an automated process that combines (i) *error injection* using a stronger yet cost-effective model (GPT-5-mini) to synthesize diverse, realistic failure patterns, and (ii) *lightweight policy inference* to mimic errors produced by smaller models in practice. The resulting pairs, together with a detailed instruction prompt, are provided to GPT-5-mini to generate high-quality supervision signals for reward modeling. Figures 7 and 8 show the Chart-specific distillation prompt (split into two parts due to length).

Prompt overview (Chart). The prompt casts the judge as an "experienced specialist for data visualization" and explicitly defines the evaluation objective: compare the Generated Image against the Original Image and report discrepancies that affect (i) correctness of the data, (ii) layout/structure fidelity, and (iii) visual consistency/readability. Importantly, it instructs the judge to output `zero errors` if no meaningful difference exists.

Error taxonomy and severity. To ensure consistent and fine-grained supervision, the prompt requires every discovered discrepancy to be assigned to exactly one of four categories: *structure_error* (layout and chart type), *data_error* (content/geometry and value distortion), *text_error* (titles, labels, legends, ticks, and annotations), and *style_error* (aesthetic and appearance). For each error, the judge also assigns a severity level on a three-point scale (minor / moderate / critical), which captures how strongly the discrepancy impacts fidelity or interpretation. This taxonomy is designed to reflect common failure modes in chart rendering and to provide training targets that are both interpretable and actionable.

Structured JSON output. Beyond natural-language descriptions, the prompt enforces a strict output format. As shown in Fig. 8, the judge must output **only** a single valid JSON object following a

Prompt for Visual-ERM — Chart

```
PROMPT_CHART2CODE_JUDGEMENT = """\
You are an **Experienced Specialist for Data Visualization**. You will be provided with **two images**:

1. **Original Image**: a chart rendered using ground-truth Matplotlib code.
2. **Generated Image**: a chart rendered using AI-generated Matplotlib code for the **Original Image**.

Your task is to **compare the Generated Image against the Original Image** and identify **all visual discrepancies**, then
summarize them in a strict JSON format.

Additionally, You should assign a **severity score** for each error:
- 1 (minor): small errors that barely affect readability or understanding.
- 2 (medium): errors that affect partial understanding and require manual correction for reliable use.
- 3 (severe): structural or key-content errors that break reliable alignment or can significantly mislead.

You MUST output a single JSON object **ONLY**.
- Do NOT include any extra text.
- The JSON schema MUST match exactly:
{
  "structure_error_count": int,
  "data_error_count": int,
  "text_error_count": int,
  "style_error_count": int,
  "errors": [
    {
      "category": "structure_error | data_error | text_error | style_error",
      "severity": 1 | 2 | 3,
      "location": "Specific location (e.g., 'Left subplot title', 'Red line data', 'Legend')",
      "description": "Concise description of the error."
    }
  ]
}
"""
```

Figure 9: **Visual-ERM inference prompt.** The prompting template used by Visual-ERM at inference time. This prompt is specialized for the **Chart-to-Code** setting.

fixed schema, including: (i) per-category error counts, and (ii) an explicit list of error items, each annotated with `category`, `severity`, a localized `location` description (e.g., "legend", "left subplot title"), and a concise `description`. The prompt additionally specifies consistency constraints (e.g., counts must match the number of listed errors; the error list must be empty when the two images are visually consistent). These constraints substantially reduce annotation ambiguity and make the distilled outputs directly usable for reward-model training and evaluation.

Why this design. This prompt design provides two practical benefits. First, it produces *fine-grained* supervision that decomposes visual discrepancies into interpretable components (structure/data/text/style) rather than collapsing them into a single opaque score. Second, the enforced JSON format enables scalable filtering, aggregation, and auditing of the distilled labels, which is critical for building large reward-modeling datasets with reliable quality control.

C.2 Visual-ERM Inference Prompts

After constructing the distillation dataset, we train our reward model, Visual-ERM, to produce fine-grained, structured feedback for image-to-image visual equivalence. Since this behavior is learned through supervised fine-tuning, Visual-ERM does not require an overly restrictive prompt at inference time. Instead, we adopt lightweight and efficient prompting templates that primarily (i) specify the comparison setup (Original vs. Generated), (ii) request exhaustive discrepancy discovery conditioned on the task, and (iii) enforce a strict JSON output schema.

We design a task-specific prompt for each vision-to-code setting—Chart-to-Code, Table-to-Markdown, and SVG-to-Code. These prompts share the same overall structure and interface, but differ in the error taxonomy and schema fields to reflect domain-specific failure modes (e.g., chart type/axes/legend for charts, layout and numeric/text errors for tables, and shape/style/text-symbol discrepancies for SVGs). The complete prompts are provided in Fig. 9, Fig. 10, and Fig. 11.

Prompt for Visual-ERM — Table

```
PROMPT_TABLE2MARKDOWN_JUDGEMENT = """\
You are a table parsing quality auditor. You will be given two table images:

- Image 1: the original table screenshot (original)
- Image 2: the parsed / rendered table image (parsed)

Your job is to compare the original vs. parsed images and identify all discrepancies in the parsed table relative to the original,
then summarize them in a strict JSON format.

For every discrepancy, assign exactly ONE of the following categories:
1) layout_error (structure/layout errors)
2) text_error (text recognition errors)
3) numeric_error (numeric/symbol/unit errors)

Assign a severity score for each error:
- 1 (minor): small errors that barely affect readability or understanding
- 2 (medium): errors that affect partial understanding and require manual correction for reliable use
- 3 (severe): structural or key-content errors that break reliable alignment or can significantly mislead

You MUST output a single JSON object ONLY.
- Do NOT include any extra text.
- Do NOT wrap the JSON in markdown code fences like ```json.
- The JSON schema MUST match exactly:

{
  "layout_error_count": int,
  "text_error_count": int,
  "numeric_error_count": int,
  "errors": [
    {
      "type": "layout_error | text_error | numeric_error",
      "description": "A short description of what is wrong and where it is wrong",
      "severity": 1|2|3
    }
  ]
}
"""
```

Figure 10: **Visual-ERM inference prompt.** The prompting template used by Visual-ERM at inference time. This prompt is specialized for the **Table-to-Markdown** setting.

Chart-to-Code prompt (Fig. 9). The chart prompt frames Visual-ERM as a data visualization specialist and focuses the comparison on discrepancies that matter for faithful chart reproduction. It explicitly decomposes errors into `structure_error`, `data_error`, `text_error`, and `style_error`, covering (i) global composition (chart type, subplot arrangement, axes sharing), (ii) geometric/value fidelity (bar heights, line trends, point positions, scale mismatches), (iii) textual correctness (titles, tick labels, legends, annotations), and (iv) appearance-level mismatches that may affect readability. This taxonomy is particularly suitable for charts because visually minor differences (e.g., small font changes) can be less important than semantic distortions (e.g., wrong scale or swapped series), and the explicit severity field further encourages Visual-ERM to surface high-impact errors prominently. In practice, the chart prompt also encourages localized descriptions (e.g., "legend", "x-axis ticks", "left subplot line") which makes the feedback more actionable for downstream debugging and revision (such as test-time scaling).

Table-to-Markdown prompt (Fig. 10). The table prompt casts Visual-ERM as a table parsing quality auditor and emphasizes structural alignment and textual/numeric accuracy. Compared to charts, tables have less ambiguity in global structure but are highly sensitive to *layout* (row/column spans, merged cells, alignment) and *cell-level content*. Accordingly, the prompt uses a concise three-way categorization: `layout_error`, `text_error`, and `numeric_error`. This design reflects typical failure patterns in table parsing: slight layout shifts can cascade into many downstream cell mismatches, while OCR-style errors often manifest as character substitutions or missing tokens. Separating `numeric_error` from `text_error` is also important in practice, since numerical discrepancies (units, decimal points, sign errors) are often more harmful even when visually subtle. The

Prompt for Visual-ERM — SVG

```
PROMPT_IMG2SVG_JUDGEMENT = """\
You are an expert QA Specialist for Vector Graphics & Icon Generation. You will be provided with two images to compare
visually:

- Image 1 (Ground Truth): The original icon/graphic rendered from correct SVG code.
- Image 2 (Prediction): An icon/graphic rendered from AI-generated SVG code attempting to reproduce Image 1.

Your job is to compare the original vs. parsed images and identify all discrepancies in the parsed image relative to the original,
then summarize them in a strict JSON format.

Assign a severity score for each error:
- 1 (minor): small errors that barely affect readability or understanding
- 2 (medium): errors that affect partial understanding and require manual correction for reliable use
- 3 (severe): structural or key-content errors that break reliable alignment or can significantly mislead

You MUST output a single JSON object ONLY.
- Do NOT include any extra text.
- Do NOT wrap the JSON in markdown code fences like ```json.
- The JSON schema MUST match exactly:

{
  "structure_error_count": int,
  "shape_error_count": int,
  "style_error_count": int,
  "text_symbol_error_count": int,
  "other_error_count": int,
  "errors": [
    {
      "category": "structure_error | shape_error | style_error | text_symbol_error | other_error",
      "severity": 1 | 2 | 3,
      "location": "Visual location (e.g., 'Train Wheel', 'Eye Contour', 'Canvas Background')",
      "description": "Concise description of the visual mismatch."
    }
  ]
}
"""
```

Figure 11: **Visual-ERM inference prompt.** The prompting template used by Visual-ERM at inference time. This prompt is specialized for the **SVG-to-Code** setting.

enforced JSON schema provides a predictable interface for aggregating errors and computing metrics, while keeping the prompt lightweight enough for efficient inference.

SVG-to-Code prompt (Fig. 11). The SVG prompt targets vector-graphic reconstruction, where fidelity hinges on geometric primitives and styling attributes. To capture these characteristics, the prompt introduces categories beyond those used for raster-like structured visuals: `shape_error` (incorrect geometry or missing/extraneous components), `style_error` (stroke width, fill, color, opacity), `text_symbol_error` (glyphs, labels, icons) and `structure_error` (global composition and grouping). This taxonomy reflects the fact that SVG failures often involve precise contours and layering, which may not be well described by "data/text/style" alone. Requiring a location field further encourages spatial grounding, making the output interpretable and enabling targeted correction during RL or test-time scaling.

Common design choices across tasks. Across all three prompts, we adopt several shared choices that improve usability and robustness: (i) the judge is instructed to compare *only what is visible* in the rendered images, preventing reliance on spurious code cues; (ii) the output must be a *single valid JSON* object with a fixed schema, which reduces post-processing ambiguity and supports scalable training/evaluation; and (iii) the severity score provides a unified notion of error impact, enabling downstream pipelines (e.g., reflection-and-revision) to prioritize the most critical issues first. Together, these prompts serve as a lightweight interface that activates Visual-ERM's learned capabilities while remaining consistent across heterogeneous vision-to-code domains.

VC-RewardBench Evaluation Prompt

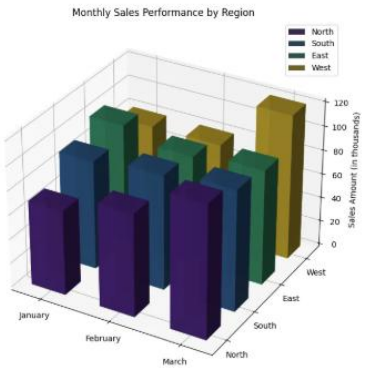
```
prompt = f"""
You are a strict "error alignment evaluator." Your task is: compare the error list from PRED with the error list from GT,
determine which ones refer to the same specific error point, and output structured JSON.

[Data Category]
{category}

[Allowed Error Types (you may only match within these types)]
{allowed_types}

[Matching Rules (VERY IMPORTANT)]
1) A match is only allowed when pred.type == gt.type OR pred.category == gt.category. Otherwise, it must be judged as not
matching.
2) A "match" requires that the descriptions point to the same specific error point (same location/object/cell/chart element),
and that the error phenomenon is the same or highly consistent.
3) If the pred description is more generalized than gt but clearly refers to the same error point, it can be considered
match_level="partial"; otherwise it is "no".
4) Each pred can match at most one gt, and each gt can match at most one pred (1-to-1 matching).
5) Do not match randomly: it is better to leave unmatched than to make an incorrect match.

[Your output MUST be strictly valid JSON, with no extra text]
The JSON output format is:
{
  "matches": [
    {
      "pred_id": 0,
      "gt_id": 3,
      "match_level": "yes" | "partial"
    }
  ],
  "unmatched_pred": [1, 2],
  "unmatched_gt": [0, 4],
  "notes": "Optional, at most one sentence"
}

[PRED Error List]
{json.dumps(pred_list, ensure_ascii=False, indent=2)}

[GT Error List]
{json.dumps(gt_list, ensure_ascii=False, indent=2)}
""".strip()
```

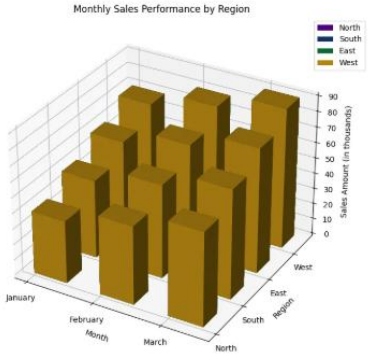
Figure 12: **VC-RewardBench evaluation prompt.** For the subjective, free-form error descriptions in VC-RewardBench, we use an LLM-assisted judge to perform error matching and verification; the prompt used in this evaluation protocol is shown here.

C.3 VC-RewardBench Evaluation Prompt

A key component of VC-RewardBench is the evaluation of *free-form error descriptions*, which summarize the visual mismatches between the original image and the image re-rendered from the parsed output. Since these descriptions are subjective and highly open-ended, rule-based string matching is brittle and fails to account for paraphrases, partial matches, or differences in granularity. We therefore use an LLM-as-Judge to perform verification: it determines which predicted errors are correctly matched to ground-truth errors, which are hallucinated, and which are only partially aligned.

Figure 12 shows the prompt used for this evaluation. The prompt is intentionally designed to be *constrained* and *deterministic*: it restricts matching to a predefined set of error types, enforces strict type/category consistency, and requires 1-to-1 alignment between predicted and ground-truth error items. It also explicitly encourages conservative decisions (preferring "unmatched" over uncertain matches) and outputs a single JSON object with match indices and match levels (`yes` vs. `partial`). These constraints reduce judge arbitrariness and make the resulting metrics more robust across different judge models (as further validated in Tab. 11).

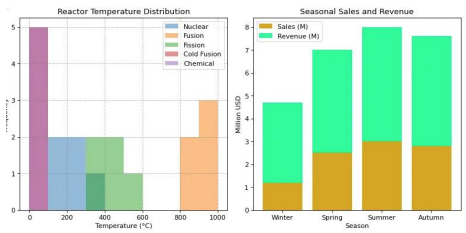


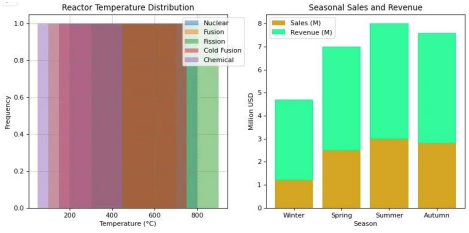


GT Image **Reference Image**

Annotation:

[{"type":"data_error","severity":3,"description":"Bar heights and z-axis range differ from the reference (e.g., reference shows values up to ~120 for West March, generated peaks at ~90) — data values and scale are inconsistent."},
{"type":"style_error","severity":2,"description":"Reference uses distinct colors per region (North purple, South blue, East green, West gold); generated image renders all bars in a single gold color, losing the region color mapping."},
{"type":"text_error","severity":1,"description":"Generated image includes axis labels 'Month' and 'Region' that are not present in the reference, altering textual layout/labels of the chart."},
{"type":"data_error","severity":3,"description":"The relative magnitudes of the data are completely wrong."}]





GT Image **Reference Image**

Annotation:

[{"type":"data_error","severity":3,"description":"The histogram data is incorrectly visualized. The original chart shows frequency counts up to 5 with distinct distribution peaks for different categories. The generated chart displays overlapping bars with a uniform height of 1.0, and the y-axis scale is incorrect (0-1.0 instead of 0-5)."},
{"type":"data_error","severity":3,"description":"In the reactor temperature distribution, the temperatures corresponding to the data and their frequencies are completely incorrect."},
{"type":"data_error","severity":1,"description":"In the reactor temperature distribution, the x-axis range should extend to 1000 rather than 800."}]

Figure 13: **Case study.** We show representative examples from the Chart-to-Code reward-model training data and the benchmark.

EGS Project	Type	Country	Capacity (MW_e)	Plant Type	Depth (km)
Soultz (EU)	R&D	France	1.5	Binary (ORC)	4.2
Desert Peak	R&D	USA	11-50	Binary (ORC)	(Unknown)
Landau	Commercial	Germany (EU)	3	Binary (ORC)	3.3
Paralana (phase I)	Commercial	Australia	7-30	Binary (ORC)	4.1
Cooper Basin	Commercial	Australia	250-500	Binary (Kalina)	4.3
The Geysers	Demonstration	USA	(Unknown)	Flash	3.5-3.8
Ogachi	R&D	Japan	(Unknown)	Flash	1.0-1.1
United Downs, Redruth	Commercial	United Kingdom	10	Binary (ORC)	4.5
Eden Project	Commercial	United Kingdom	3	Binary (ORC)	3-4

GT Image

EGS Projec	Type	Country	Capcity (MW)	Plant Type	Depth
Soultz (E U)	R&D	France	1,5	Binary (ORC)	4.2
Desert Peak	R&D	USA	11-5O	Binary (ORC)	(Unknown)
Landau	Germany (EU)	Commercial	3	Binary (ORC)	3.3
Paralana (phase I)	Commercial	Australia	7-30	Binary (ORC)	4.1
Cooper Basin	Commercial	Australia	250-500	Binary (Kalina)	B.3
The Geysers	Dmonstration	USA	(Unknown)	Flash	3.5-3.8
Ogachi	R&D	Japn	Flash	(Unknown)	1.0-1.1
United Downs, Redruth	Commercial	United Kingdom	1O	Binary (ORC)	4.5
Eden Project	Commercial	United Kingdm	3	Bnary (ORC)	3-4

Reference Image

Annotation:

[{"type":"text_error","severity":2,"description":"Header cell 'EGS Projec' is missing final 't' (should be 'EGS Project')"},
{"type":"text_error","severity":2,"description":"Header 'Capacity (MW)' is missing the trailing 'e' unit from original 'Capacity (MWe)'"},
{"type":"text_error","severity":2,"description":"Header 'Depth' is missing the unit '(km)' present in the original"},
{"type":"text_error","severity":1,"description":"Project name 'Soultz (E U)' shows an extra space in '(E U)' vs original '(EU)'"},
{"type":"numeric_error","severity":1,"description":"Soultz capacity rendered as '1,5' (comma) instead of '1.5' (dot) in Capacity column"},
{"type":"numeric_error","severity":1,"description":"Desert Peak capacity rendered as '11-5O' (letter O) instead of '11-50' (zero)"},
{"type":"layout_error","severity":3,"description":"Row 'Landau' has Type and Country columns swapped (shows 'Germany (EU)' under Type and 'Commercial' under Country)"},
{"type":"numeric_error","severity":1,"description":"Cooper Basin depth rendered as 'B.3' (letter B) instead of '4.3'"}]

Responses		Year h =0	Year h =1	Year h =2	Year h =3	Year h =4
Δ_h *Short-term interest rate*	OLS	1.00	0.69^{***} (0.06)	0.45^{***} (0.04)	0.38^{***} (0.05)	0.35^{***} (0.07)
	IV	1.00	1.31^{***} (0.16)	1.02^{***} (0.19)	0.80^{***} (0.19)	0.39^{***} (0.14)
Δ_h *Long-term interest rate*	OLS	0.34^{***} (0.03)	0.33^{***} (0.04)	0.32^{***} (0.04)	0.26^{***} (0.03)	0.26^{***} (0.03)
		0.42^{***} (0.05)	0.55^{***} (0.09)	0.67^{***} (0.13)	0.60^{***} (0.15)	0.39^{***} (0.13)
Δ_h *Mortgage loans /GDP*	OLS	-0.11^{***} (0.04)	-0.15^{**} (0.06)	-0.25^{***} (0.08)	-0.29^{**} (0.11)	-0.45^{***} (0.15)
	IV	-0.45^{***} (0.15)	-1.19^{***} (0.38)	-1.87^{***} (0.61)	-2.35^{***} (0.76)	-2.82^{***} (0.86)
Δ_h log *House prices /income*	OLS	0.35 (0.33)	0.15 (0.40)	-0.33 (0.48)	-0.67 (0.51)	-0.90 (0.56)
	IV	-0.18 (0.79)	-1.76 (1.67)	-3.72^{*} (2.05)	-5.02^{**} (2.27)	-4.37^{**} (1.88)

GT Image

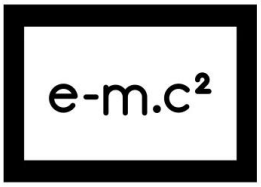
Respones		Year h =O	Yearh=1	Year h =2	Year h =3	Year h-4
Δh Short-ter interest rate	OLS	1.00	0.69*** (0.06)	1.02*** (0.04)	0.38*** (0.05)	0.35*** (0.14)
	IV	1.00	1.31*** (0.16)	0.45*** (0.19)	O.8O*** (0.19)	0.39*** (0.07)
Δh Long term int rate	OLS	0.34 ** (0.03)	0.33*** (0.04)	0.32*** (0.04)	0.26*** (0.03)	0.26*** (0.03)
		0.42*** (0.05)	0.55*** (0.09)	0.67*** (0.13)	0.60*** (O.I5)	0.39*** (0.13)
Δh Mortage loans /GDP	OLS	-0.11*** (0.04)	-0.15** (0.06)	-0.25*** (0.08)	-0.29** * (0.11)	-0.45*** (0.15)
	IV	-0.45*** (0.15)	-1.19*** (0.38)	-1.87*** (0.61)	-2.35*** (0.76)	-2.82*** (0.86)
Δh log House prices /incom	OLS	-0.18 (0.79)	-1.76 (1.67)	-3.72* (2.05)	-5.02** (2.27)	-4.37** (1.88)
	IV	O.35 (0.33)	0.15 (0.40)	-0.33 (0.48)	-0.67 (0.51)	-0.9O (0.56)

Reference Image

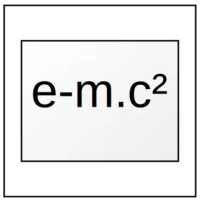
Annotation:

[{"type":"text_error","severity":2,"description":"Column header 'Responses' is misspelled as 'Respones' in the parsed table"},
{"type":"text_error","severity":2,"description":"Column header 'Year h=0' rendered as 'Year h =0' (letter O used instead of digit 0)"},
{"type":"text_error","severity":1,"description":"Column header 'Year h=1' rendered as 'Yearh=1' (missing space)"},
{"type":"text_error","severity":2,"description":"Column header 'Year h=4' rendered as 'Year h-4' (equals sign replaced by dash / formatting changed)"},
{"type":"layout_error","severity":3,"description":"Δh Short-term interest rate: OLS and IV numeric cells are misaligned/swapped (Year h=2 shows 1.02 in OLS and 0.45 in IV instead of vice versa) and corresponding standard errors are similarly misaligned"},
{"type":"text_error","severity":1,"description":"Δh Long-term interest rate (OLS, Year h=0) printed as '0.34 **' (two asterisks) in parsed but original shows three asterisks '0.34***'"}]

Figure 14: **Case study.** We show representative examples from the Table-to-Markdown reward-model training data and the benchmark.



GT Image



Reference Image

Annotation:

[{"type":"structure_error","severity":2,"description":"Prediction introduces an extra thin inner border (double-border) that is not present in the ground truth."},
{"type":"structure_error","severity":2,"description":"Ground truth has a thick black outer bezel/frame; prediction replaces it with a thin outer border altering the frame structure."},
{"type":"shape_error","severity":2,"description":"Glyph geometry differs — letters are bulkier and proportions are changed (strokes/curves not matching ground-truth shapes)."},
{"type":"shape_error","severity":2,"description":"The dot symbol is circular in the ground truth but square in the prediction. Similarly, the hyphen has rounded caps in the ground truth but is rectangular in the prediction."}]



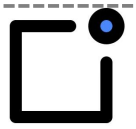
GT Image



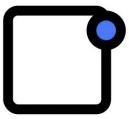
Reference Image

Annotation:

[{"type":"structure_error","severity":3,"description":"The ground-truth continuous dark horizontal bar across the middle of the strings is missing/broken in the prediction (no uninterrupted band spanning all strings)."},
{"type":"structure_error","severity":2,"description":"An extra pointed spike/extrusion appears on the rightmost yellow border that is not present in the ground truth (hallucinated shape)."},
{"type":"shape_error","severity":2,"description":"Widths and spacing of the inner bars are inconsistent and uneven compared to the evenly spaced, uniform-width strings in the ground truth."},
{"type":"structure_error","severity":3,"description":"The frame is severely malformed with a jagged, protruding shape instead of a smooth continuous curve."}]



GT Image



Reference Image

Annotation:

[{"type":"structure_error","severity":3,"description":"Ground truth has a distinct large black outer circle placed outside the square; prediction shows a much smaller/partially overlapping black ring that is clipped against the square instead of being a separate, detached circle."},
{"type":"structure_error","severity":3,"description":"Ground truth square has an open/terminated top-right segment to accommodate the external dot; prediction renders the square corner closed (no deliberate opening), causing the dot to overlap the corner."},
{"type":"style_error","severity":2,"description":"Blue inner circle is mis-centered and partly clipped/overlapping the square in the prediction; its size/placement relative to the outer ring differs from the ground truth."}]

Figure 15: **Case study.** We show representative examples from the SVG-to-Code reward-model training data and the benchmark.

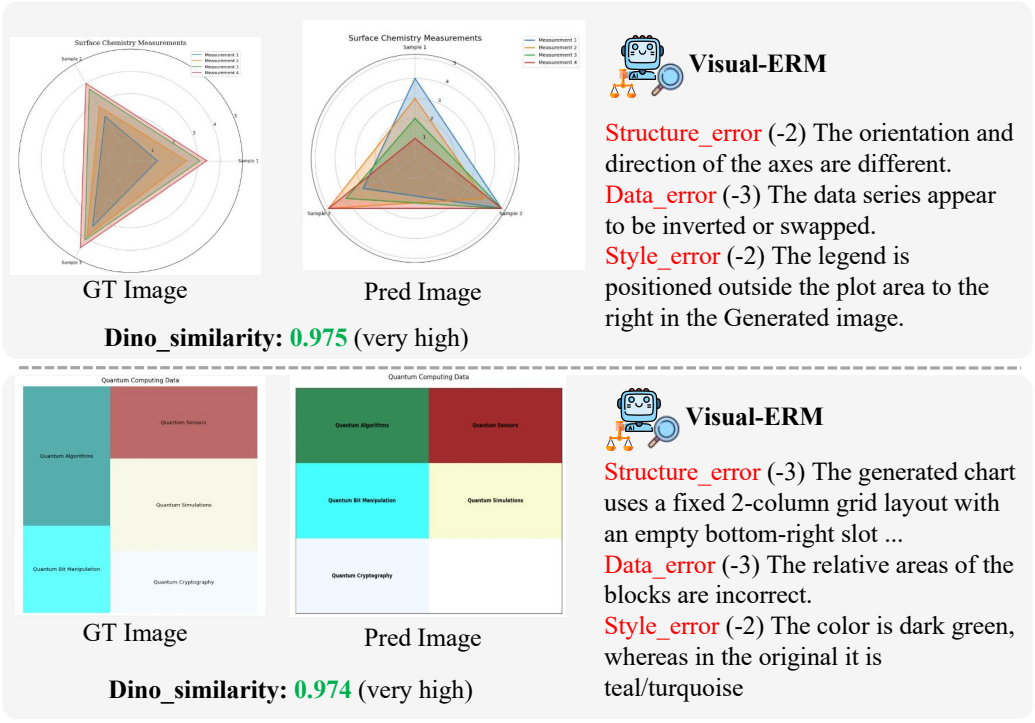


Figure 16: **Further Case Analysis.** We present additional cases to highlight the limitations of current reward models.

D Case Study

D.1 Data Cases

In this section, we provide representative qualitative examples to illustrate the data format and error annotations used in our pipeline. The reward-model training data and VC-RewardBench examples follow a similar structure—both are built from (Original Image, Re-rendered Image) pairs and accompanied by fine-grained error descriptions. Compared to the training data, VC-RewardBench is labeled by stronger proprietary models and further curated through human filtering, making it more reliable for evaluation. We showcase examples from VC-RewardBench across three domains (Chart, Table, and SVG), as shown in Fig. 13, Fig. 14, and Fig. 15.

Chart cases (Fig. 13). The chart examples highlight typical failure modes where likely textual and structural text are insufficient, and the discrepancies must be judged directly in the rendered space. In the first example, the annotations capture multiple error types: a high-severity `data_error` indicating mismatched bar magnitudes and/or incorrect z-axis scaling, a `style_error` where the color mapping collapses into a single palette (losing per-category distinction), and a `text_error` that flags unexpected or misplaced labels (e.g., additional axis/title tokens that alter readability). This case demonstrates that visually faithful reconstruction requires jointly preserving data geometry, semantic encodings (color-to-category), and text placement. In the second example, the dominant issues are again `data_error`: the predicted histogram overlays bars incorrectly and uses an inconsistent axis range, which changes the implied distribution. The annotations emphasize that even when the global chart type appears correct, local numeric scaling and bin/value placement errors can lead to critical semantic distortion.

Table cases (Fig. 14). The table examples illustrate that errors often concentrate on OCR- and layout-related details, where small differences translate into large structural mistakes in the parsed table. In the first example, the annotations include multiple `text_error` instances on header strings (missing characters, missing units) as well as `numeric_error` cases where punctuation or symbols change the value semantics (e.g., comma vs. dot in decimals, letter `O` vs. digit `0`). Additionally, a

high-severity `layout_error` captures column swapping, which is particularly harmful because it can propagate to many cell assignments and invalidate the table schema. In the second example, the errors focus on fine-grained text rendering and alignment: header tokens are misspelled, spacing/formatting changes introduce mismatched column names, and significance markers are rendered inconsistently. The `layout_error` further reflects misalignment of numeric entries across columns, a common structured-visual failure that is hard to detect in the text space but clearly visible after rendering.

SVG cases (Fig. 15). The SVG examples emphasize geometry- and structure-centric discrepancies that arise in vector-graphic reconstruction. In the first case, the annotations include `structure_error` due to an extra inner border and an altered frame composition, along with `shape_error` describing changed glyph geometry (stroke thickness and curve profiles) and subtle symbol differences (e.g., rounded vs. rectangular caps). In the second case, the prediction introduces hallucinated protrusions and fails to reproduce continuous elements (e.g., a missing uninterrupted horizontal band), while also exhibiting inconsistent spacing of repeated bars—a combination of `structure_error` and `shape_error`. In the third case, the mismatches include incorrect global composition (outer ring size/placement), an incorrect top-right termination detail, and a mis-centered inner circle, reflecting intertwined `structure_error` and `style_error`. These cases show that SVG fidelity depends critically on precise spatial relationships, path geometry, and layering, which motivates an image-to-image judge that can reason over fine-grained rendered differences.

Overall, these examples demonstrate that Visual-ERM and VC-RewardBench capture diverse, realistic discrepancy patterns across domains, and that the error annotations provide interpretable and actionable supervision signals for both training and evaluation.

D.2 Further Case Analysis

We provide additional cases to demonstrate the limitations of existing reward models, as illustrated in Fig. 16.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract and introduction accurately reflect the paper's contributions and scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [N/A]

   Justification: No limitations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: Yes, the paper provide the full set of assumptions and a complete (and correct) proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Yes.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The paper provide open access to the data and code.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper specify all the training and test details.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: No.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.

- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: No.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [No]

    Justification: This work does not present immediate societal impacts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: Yes.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Yes.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.