ViCo: Visual-oriented Coding with Self-Reflection for Chart Replication

Jiaxin Duan, Dian Jiao, Shuai Zhao
China Electronics Cloud Technology Co., Ltd.
{duanjiaxin,jiaodian,zhaoshuai}@cestc.cn
jiaxdan@gmail.com, 554848018@qq.com, t.zhaosh@gmail.com

Abstract

This paper addresses the challenge of generating high-quality academic charts that match the visual standards of human-authored papers. While existing AI agents can produce well-structured text and code, their generated visualizations often lack the stylistic and semantic fidelity of human designs. Advanced coding agents that employ self-reflection mechanisms exhibit poor visual reasoning and limited reflection following, resulting in sparse reward signals that severely undermine their reinforcement learning (RL). We propose ViCo, a training framework for visual-oriented coding that employs iterative reflections to align generated chart images progressively with the reference. We first introduce a self-supervised warm-up stage, which augments *Monte Carlo Tree Search* with consistency-based pruning to synthesize high-quality reflection trajectories, ensuring that each coding step strictly follows the outcomes of prior reflections. A multi-step RL algorithm is then developed, using *counterfactual baselines* to estimate advantage for reflection and action steps within each refinement cycle, thereby addressing the reward sparsity. To enable efficient reward in massive training, we propose an automatic, multifaceted evaluation framework that assesses charts' style, layout, and semantic consistency via a hierarchical heterogeneous layout graph structure. Experiments on three public benchmarks demonstrate that ViCo, trained on an 8B model, achieves performance close to proprietary LLMs with adequate reflection capabilities.

1 Introduction

Scientific plotting is fundamental for academic publications, providing a concise visual representation of complex data that enables readers to grasp key findings, trends, and comparisons quickly. Despite significant advances in AI-powered academic assistants that can generate well-structured text and syntactically correct code (Liu et al., 2026; Ghafarollahi and Buehler, 2024), the visual quality of automatically produced scientific plots still lags considerably behind those created by human researchers (Zhao et al., 2025b; Zhang et al., 2024). A promising direction to bridge this gap is to leverage retrieval-based methods to obtain high-quality reference images from the existing literature, and then guide an AI agent to replicate the visual style of these references while making only minimal adjustments to accommodate the actual data (Zhu et al., 2026a,b). This paradigm, referred to as chart-to-code generation (Zhao et al., 2025b; Jiang et al., 2025a), has recently attracted growing attention due to its potential in producing lossless representations that preserve all critical details of a chart, including colors, coordinates, and data values.

Training an AI agent to faithfully reproduce reference plots from real data is challenging. On the one hand, the agent must possess **strong coding capabilities** to invoke appropriate toolkits (e.g., Matplotlib, Seaborn) and generate executable scripts that produce the desired visual output (Zhang et al., 2026). More critically, the robust visual reasoning capacity is required to compare its own rendered output against the reference image, identify discrepancies, and iteratively refine the code accordingly. To this end, several recent works have explored self-reflection mechanisms (Renze and Guven, 2024; Ji et al., 2023), where an agent alternates between code generation and self-critique to progressively improve its output. VisRefiner (Deng et al., 2026) employs a reinforcement learning (RL) stage for self-refinement by observing visual differences between the rendered prediction and the target design. ChartSketcher (Huang et al., 2025) introduces a multimodal feedback-driven step-by-step reasoning method that annotates intermediate reasoning steps directly onto the chart.

Despite these advances, existing reflection-based coding agents still suffer from critical limitations. First, **the effectiveness of reflection is often ques-**

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 41043–41071
October 24-29, 2026 ©2026 Association for Computational Linguistics

tionable. Empirical observations reveal that in many cases the agent fails to identify the root cause of the visual discrepancy, producing superficial or even misleading critiques that lead to negligible improvements or, worse, degrade the code quality over multiple iterations (Wang et al., 2025). Second, the agent also exhibits **limited reflection-following capability.** Even when the reflection correctly identifies a flaw, the subsequent code generation step may ignore or misinterpret the suggested fix, resulting in a "say one thing, do another" behavior. Third, optimizing such multi-turn ReAct (Yao et al., 2023) processes via RL faces the severe credit assignment challenge caused by reward sparsity. In a typical reflection loop, the global reward (e.g., replication fidelity) is only available at the end of a long trajectory. RL algorithms, like PPO (Schulman et al., 2017) or GRPO (Shao et al., 2024), struggle to attribute this sparse global reward to specific coding or reflection steps within the trajectory (Deng et al., 2026). The lack of fine-grained credit assignment hinders the model's ability to learn constructive reflection and robust coding that follow its own introspective comments, leading to suboptimal convergence or even performance degradation during training. Additionally, existing approaches often rely on heavy proprietary models (e.g., GPT-5 (Singh et al., 2026)) as judges for quality assessment, which is computationally prohibitive for large-scale training (Zhao et al., 2025b).

To address these challenges, we propose **ViCo**, a **Vi**sual-oriented **Co**ding framework that enhances chart replication through reflective self-training. To prevent unproductive reflections and execution failures, ViCo introduces a **self-supervised warm-up** phase using trajectories sampled via Monte Carlo Tree Search (MCTS) with **consistency-based pruning**. A *checklist mechanism* is employed during MCTS simulation to evaluate whether the generated code strictly aligns with the preceding reflection. Trajectory paths that violate this consistency or fail to yield monotonic reward improvements are pruned immediately. This process produces high-quality trajectories in which each coding step demonstrably executes its corresponding reflection to achieve higher rewards, enabling effective behavior cloning of the self-reflection policy. Subsequently, a counterfactual-based advantage estimating mechanism is introduced in multi-step RL to address credit assignment. By curating distinct counterfactual baselines for the reflection and coding steps within each reflection episode,

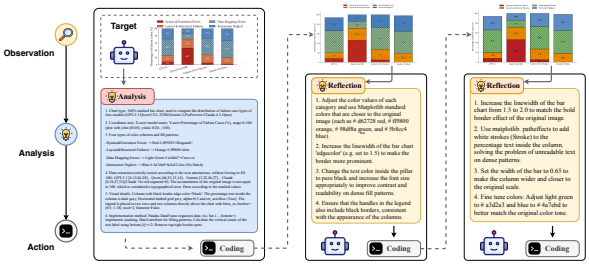


Figure 1: Overview of visual coding with self-reflection.

ViCo generates dense, step-level learning signals, which effectively stabilizes RL training without requiring expensive process reward models. Finally, to scale ViCo on large datasets, we propose a lightweight, graph-grounded evaluation framework based on a Hierarchical Heterogeneous Layout Graph (HHLG). By modeling charts as hierarchical tree-graph structures, HHLG facilitates precise element alignment and assesses replication fidelity across key complementary dimensions, including semantic, layout, and style consistency, against ground-truth references. Empirical results (§ 4.3) reveal that our HHLG-based evaluation correlates strongly with human judgments, while remaining explainable and computationally efficient for joint warm-up and RL training.

Our contributions are three-folds: 1) We propose **ViCo**, a training framework that integrates self-supervised warm-up with counterfactual-based reinforcement learning, enabling robust and effective self-reflection in visual-oriented code generation for academic chart replication. 2) We design a lightweight yet structurally comprehensive reward framework based on a hierarchical heterogeneous layout graph, which provides stable, human-aligned evaluation signals and supports scalable reinforcement learning. 3) Extensive evaluations across three challenging benchmarks demonstrate that ViCo systematically improves code pass rates (exceeding 95%) while yielding LLM-judged scores highly competitive with state-of-the-art proprietary closed-source models.

2 Problem Formulation

ViCo replicates an academic chart $\mathcal{X}$ by alternating reflection and coding over K consecutive steps, as shown in Figure 1. Starting from an initial code a_0 generated directly from $\mathcal{X}$, at each macro episode $t \in \{1, \ldots, K\}$, the agent first performs a textual **reflection** action r_t to analyze the visual discrepancies between the previous render and $\mathcal{X}$, and

then executes a **coding** action a_t to refine the code script. This iterative generation process yields a macro trajectory:

$$\tau = \big(a_0, (r_1, a_1), (r_2, a_2), \ldots, (r_K, a_K)\big).$$

Mathematically, this sequential decision-making process is a finite-horizon Partially Observable Markov Decision Process (POMDP), defined by the tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O} \rangle$. The underlying data values and source code of the target chart are hidden, constituting the unobservable state $s_t \in \mathcal{S}$. Instead, the agent makes decisions based on partial visual observations $o_t = (\mathcal{X}, \mathcal{I}_{t-1}) \in \Omega$, where $\mathcal{I}_{t-1}$ is the chart image compiled from the previously generated code. The agent's action at each step is a composed macro action $u_t = (r_t, a_t) \in \mathcal{A}$. The training objective is to learn a policy $\pi_\theta(u_t \mid h_t)$ (conditioned on the history h_t) that maximizes the expected terminal reward: $\theta^* = \arg\max_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R_{\text{final}}]$, where the global reward R_{final} measures the visual consistency between the final rendered chart $\mathcal{I}_K$ and the reference $\mathcal{X}$.

3 Training Framework

This section presents the training framework of ViCo, which consists of a self-supervision-based warming-up stage followed by reinforcement learning for multi-step agentic coding. The overall procedure is demonstrated in Figure 2 and detailed below.

3.1 Self-Supervised Warm-Up with MCTS

To learn a policy that can perform effective reflection during coding, a major challenge is the lack of high-quality reflection–coding trajectories. A common approach is to use a strong LLM as a teacher and generate pseudo-trajectories. However, the reasoning patterns of a strong teacher may differ substantially from those of a weaker student, making direct imitation less effective and potentially disrupting the student's existing reasoning behavior, especially with limited training data. Therefore, we propose a self-supervised warm-up approach that leverages MCTS to synthesize high-quality trajectories through the student's own exploration. Given a search tree that encodes candidate reflection-coding sequences, a **consistency-constrained and monotonicity-guaranteed** sampling mechanism is used to prune such branches:

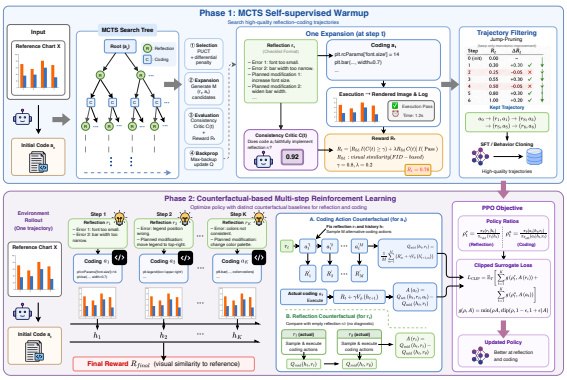


Figure 2: Overview of the ViCo two-stage training.

1) the coding step does not faithfully follow the preceding reflection, and 2) iterative refinement fails to improve the rendered quality.

Selection with Differential Penalty: Starting from the root node, the algorithm recursively selects child nodes using a variant of PUCT (Predictor Upper Confidence Bound applied to Trees). To avoid wasting computation on branches that have stagnated, we introduce a **differential penalty term** that penalizes nodes where recent reward improvements are negative. For a given state s_{t-1} and candidate action $u_t = (r_t, a_t)$, the selection score is:

$$\text{UCB}^*(s_{t-1}, u_t) = Q(s_{t-1}, u_t) + c \cdot P_{\text{prior}} \cdot \frac{\sqrt{N(s_{t-1})}}{1+N(s_{t-1}, u_t)} - \mathcal{P}(\Delta R_{\text{history}}),$$

where $Q(s_{t-1}, u_t)$ is the maximum reward achieved along this branch, $N(\cdot)$ counts visits, P_{prior} is the prior probability from the policy model, and c is an exploration constant. The penalty term $\mathcal{P}(\Delta R_{\text{history}})$ is defined as:

$$\mathcal{P}(\Delta R_{\text{history}}) = \beta \cdot \mathbb{I}\{\text{last } \rho \text{ steps show } \Delta R \le 0\},$$

where β is a scaling coefficient. This mechanism guides search away from dead-end branches.

Expansion with Structured Consistency Constraints: At a leaf node, the policy model generates M candidate child actions $\{u_t^{(i)}\}_{i=1}^{M}$. To allow the quantification of reflection effectiveness, we impose a **structured output format** through prompt engineering: the reflection r_t must be expressed as a **bulleted checklist** of error causes and planned modifications, followed by the code a_t. After generation, each code script is executed in a sandbox environment, and the resulting rendered image, execution logs, and reflection text together form the new state $s_t^{(i)}$.

Evaluation with Reflection-Consistency Critic and Reward Shaping: To address the "say one thing, do another" problem, a lightweight **Critic** LLM is used to evaluate the consistency between the reflection checklist r_t and the following codes a_t, outputs a continuous **consistency score** $C(t) \in [0, 1]$. The reward for the current macro episode is shaped as:

$$R_t = R_{\text{fid}}(t)(1 + \lambda C(t))\mathbb{I}(C(t) \geq \gamma)\mathbb{I}(\text{Pass})$$

where $\mathbb{I}(\text{Pass})$ is the environment test pass rate (1 if execution succeeds, 0 otherwise), γ is a consistency threshold (0.8), and λ is a gain coefficient (0.2). $R_{fid}(t)$ systematically evaluates the fidelity of the rendered code (detailed in Section 3.3.2). If consistency is severely lacking ($C(t) < \gamma$), even a successful execution yields zero reward. This discourages superficial successes that ignore the reflection's guidance. We also implement an **early stopping mechanism** based on the reward differential $\Delta R_t = R(t) - R(t-1)$. A patience counter ρ (initialized to $\rho_{\max}$) decrements each time $\Delta R_t \leq 0$. When ρ reaches zero, the branch is labeled a "Dead End" and ceases exploration.

Backpropagation with Max-Backup: Given the sparse and binary nature of the final correctness (either the plot matches or it does not), we adopt **Max-Backup** to propagate the highest reward encountered along a path. For each ancestor node i on the path from the current leaf to the root:

$$Q(s_i, u_{i+1}) \leftarrow \max\left(Q(s_i, u_{i+1}), V\right),$$
$$N(s_i, u_{i+1}) \leftarrow N(s_i, u_{i+1}) + 1,$$

where $V = R_t$ is the leaf's reward. This ensures that high-value branches are quickly prioritized.

Trajectory Extraction: Once MCTS completes, we extract the path that yielded the highest final reward $R_{\text{final}} = 1.0$ (i.e., a perfect visual match). The raw trajectory inevitably contains intermediate steps with non-positive reward differentials. To distill an efficient correction pattern, we apply **Jump-Pruning**: remove all macro episodes where $\Delta R_t \leq 0$, and concatenate the remaining steps, bridging the environmental feedback from the pruned step to the next retained step. The resulting trajectories for SFT have a strictly increasing reward sequence:

$$\tau_{\text{SFT}} = \left(a_0, (r_{t_1}, a_{t_1}), (r_{t_2}, a_{t_2}), \ldots, (r_{t^*}, a_{t^*})\right),$$

where t^* is the terminal step.

3.2 Counterfactual-based Multi-step RL

After warmup on self-supervised trajectories, we further employ policy gradient optimization to optimize the expected reward under the POMDP (§ 2).

3.2.1 Step-wise Advantage Estimation

To address the sparse reward problem inherent in multi-step visual replication, we extend the Proximal Policy Optimization (PPO) (Schulman et al., 2017) framework by incorporating a counterfactual-based advantage estimation method inspired by COMA (Foerster et al., 2018). Note that each macro action episode u_t consists of a sequential, non-exchangeable reflection step r_t and coding step a_t, we compute advantage for the individual step using differential counterfactual rollouts based on the same history h_t.

Advantage of Coding Step $A(a_t)$: Given the history up to h_t and the chosen reflection r_t, the action-value under the current trajectory is:

$$Q_{\text{act}}(h_t, r_t, a_t) = R_t + \gamma V_\phi(h_{t+1}),$$

where V_ϕ is the critic network parameterized by ϕ. To evaluate the quality of a_t under the given reflection, we fix r_t and sample M alternative coding actions $\{a'_1, \ldots, a'_M\}$ from the policy $\pi_\theta(\cdot \mid h_t, r_t)$. Each alternative is executed in the sandbox, producing rewards R'_m and next histories $h'_{t+1,m}$. The expected action-value under the same reflection is approximated through Monte Carlo simulation:

$$Q_{\text{mid}}(h_t, r_t) = \frac{1}{M} \sum_{m=1}^{M} \left(R'_m + \gamma V_\phi(h'_{t+1,m})\right).$$

The advantage for the actual coding action is then computed as:

$$A(a_t) = Q_{\text{act}}(h_t, r_t, a_t) - Q_{\text{mid}}(h_t, r_t).$$

Advantage of Reflection Step $A(r_t)$: To isolate the contribution of the reflection itself, we construct a counterfactual baseline where the reflection is replaced by an *Empty Reflection* $r_\emptyset$ (a prompt instructing the model to directly write code without diagnostic analysis). Under this condition, we sample M coding actions $\{a^\emptyset_1, \ldots, a^\emptyset_M\}$ from $\pi_\theta(\cdot \mid h_t, r_\emptyset)$ and compute the baseline action-value:

$$Q_{\text{mid}}(h_t, r_\emptyset) = \frac{1}{M} \sum_{m=1}^{M} \left(R^\emptyset_m + \gamma V_\phi(h^\emptyset_{t+1,m})\right).$$

The reflection advantage is then isolated as:

$$A(r_t) = Q_{\text{mid}}(h_t, r_t) - Q_{\text{mid}}(h_t, r_\emptyset).$$

3.2.2 Training Objective

We optimize the policy π_θ using an actor-critic PPO formulation, modified to optimize the sequential action components. The policy parameters θ are updated by maximizing the clipped surrogate objective using the computed step-wise advantages:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_\tau \big[\sum_{t=1}^{K} g(\rho_t^r, A(r_t)) + g(\rho_t^a, A(a_t))\big]$$

where the standard clipped surrogate loss function:

$$g(\rho, A) = \min\big(\rho A,\ \text{clip}(\rho, 1-\epsilon, 1+\epsilon)A\big),$$

and the action ratios are defined as:

$$\rho_t^r = \frac{\pi_\theta(r_t \mid h_t)}{\pi_{\theta_{\text{old}}}(r_t \mid h_t)}, \quad \rho_t^a = \frac{\pi_\theta(a_t \mid h_t, r_t)}{\pi_{\theta_{\text{old}}}(a_t \mid h_t, r_t)},$$

and ϵ represents the clip range.

The critic network V_ϕ is updated concurrently by regressing towards the target return using the mean squared error (MSE) loss:

$$\mathcal{L}^{\text{VF}}(\phi) = \frac{1}{2}\mathbb{E}_\tau \left[\sum_{t=1}^{K} \left(V_\phi(h_t) - \text{Return}_t\right)^2\right],$$

where $\text{Return}_t = \sum_{l=t}^{K} \gamma^{l-t} R_l$ is the discounted cumulative reward from step t onwards. $R_l = 0$ for $l < K$ and $R_K = R_{\text{final}}$ due to the sparse visual alignment reward.

3.3 Automatic Evaluation based on HHLG

Evaluating the fidelity of academic chart replication is fundamentally a structural, semantic, and visual alignment task. To bypass the latency, cost, and lack of interpretability of LLM-as-a-judge approaches, we propose a lightweight automatic framework grounded on a **Hierarchical Heterogeneous Layout Graph** (HHLG) structure, which significantly facilitates the localization of elements and allows the assessment of their alignment.

3.3.1 Hierarchical Data Structure: HHLG

We use heterogeneous data structures to model the hierarchical layout of complex charts. The global multi-panel layout is modeled as a **macro-tree**, whereas individual subplots are represented as **micro-graphs**. Crucially, within each micro-graph, we explicitly separate text elements and visual elements as distinct node categories.

Macro-Tree (for Global Topology): We first partition a chart into N distinct subplot regions $\mathcal{S} = \{s_1, s_2, \ldots, s_n\}$ using projection profile analysis. We define a Macro-Tree T, where the root represents the canvas and its children represent subplots, capturing their global, nested spatial arrangements (e.g., side-by-side, grid, or inset).

Micro-Graph (for Local Topology): For each subplot s_k, we construct a directed Micro-Graph $G_k = (V_k, E_k)$. Nodes within this graph are categorized into two distinct, non-overlapping sets: **1) Text Nodes (V_{text}):** Extracted via a high-precision OCR engine - MinerU. These represent title strings, axis labels, tick legends, and in-chart annotations. Each node $u \in V_{\text{text}}$ stores its text content t_u and normalized bounding box B_u. **2) Visual Nodes (V_{visual}):** Extracted using class-agnostic segmentation anything model (Kirillov et al., 2023). These represent visual data-carrying elements like bars, lines, scatter points, and pie slices. Each node $v \in V_{\text{visual}}$ stores its segmented mask M_v and bounding box B_v. For dense datasets (e.g., scatter plots), individual points are grouped into a single cluster node to maintain graph tractability. **3) Typed Edges (E_k):** Edges are directed and typed to capture both geometric and functional relationships: $E_k \subseteq (V_k \times V_k \times \mathcal{R})$, where the relation type $r \in \mathcal{R}$ includes spatial relations (e.g., `left-of`, `above`) and semantic association relations (e.g., a text node in V_{text} `labels` a visual node in V_{visual}).

3.3.2 Elements Alignment and Assessment

Once the HHLG is constructed for both the reference chart x and the generated chart y, we compute their consistency alongside four dimensions.

Semantic Consistency (S_{sem}) measures the textual alignment between the reference and generated charts. Instead of analyzing the entire image, we isolate the computation to the Text Nodes V_{text} across matched subplots. For the text nodes of reference $V_{\text{text},x}$ and generator $V_{\text{text},y}$, we solve a bipartite matching problem. The pairwise alignment cost $c(u, v)$ between a reference text node u and a generated text node v is defined as:

$$c(u,v) = \alpha \cdot d_{\text{edit}}(t_u, t_v) + (1-\alpha)\cdot(1 - \text{IoU}(B_u, B_v))$$

where d_{edit} is the normalized Levenshtein distance of the strings, and IoU measures the spatial bounding box intersection-over-union. The final semantic

score S_{sem} is computed via the Hungarian algorithm to minimize global alignment cost, normalized by the node count.

Layout Consistency (S_{lay}) evaluates the spatial and structural preservation of the chart layout at both the macro and micro levels:

$$d_{\text{layout}}(x, y) = d_{\text{Tree}}(T_x, T_y) + \sum_k \text{GED}(G_{x,k}, G_{y,\pi(k)})$$

where d_{Tree} is the tree edit distance matching the macro subplot layouts, and $\pi(k)$ represents the optimal subplot mapping. The micro-layout discrepancy GED computes the Graph Edit Distance between matched subplot micro-graphs. Because nodes are explicitly categorized into V_{text} and V_{visual}, the node substitution cost in GED is constrained: a text node cannot match a visual node. This categorization significantly prunes the search space, allowing us to use a fast, linear-time bipartite approximation of GED without sacrificing evaluation accuracy. The final score is normalized as:

$$S_{\text{lay}} = \exp(-\beta \cdot d_{\text{layout}})$$

Style Consistency (S_{style}). To evaluate styles like font faces, gridline textures, and rendering strokes without interference from global spatial layout shifts, we extract visual style features locally. For each matched subplot node in the Macro-Tree, we extract the image patch bounding the subplot. We pass these patches through a convolutional backbone (i.e., VGG-19 (Simonyan and Zisserman, 2015)) to extract activation feature maps F^l at intermediate layers. We then calculate style via the Gram Matrix G^l, which discards spatial configurations and focuses strictly on texture and stylistic patterns:

$$G^l_{i,j} = \frac{1}{H_l W_l} \sum_{m=1}^{H_l W_l} F^l_{i,m} F^l_{j,m}$$

The style score is determined by the distance between the Gram matrices of matched subplot patches:

$$S_{\text{style}} = \exp\left(-\gamma \sum_{l \in L} w_l \|G^l_x - G^l_y\|^2_F\right)$$

where $\|\cdot\|_F$ denotes the Frobenius norm.

Color Consistency (S_{color}). To prevent incorrect color-to-entity mappings (e.g., swapping colors between a bar and a line), color consistency is calculated directly on the Visual Nodes V_{visual}. For each visual node $v \in V_{\text{visual}}$ in the micro-graphs, we extract the dominant color vector $\mathbf{c}_v$ in the perceptually uniform LAB color space from its segmented mask M_v (excluding backgrounds). During micro-graph alignment, we compute the color distance between aligned visual node pairs. The color consistency score is defined as:

$$S_{\text{color}} = \frac{1}{|M|} \sum_{(u,v) \in M} \left(1 - \frac{\|\mathbf{c}_u - \mathbf{c}_v\|_2}{\Delta E_{\text{max}}}\right)$$

where M is the set of successfully matched visual node pairs (u, v) from the layout alignment step, and ΔE_{max} represents the maximum possible perceptual color difference. This guarantees that color evaluation is explicitly bound to specific visual elements.

Composite Metric. The final multi-dimensional evaluation score R is formulated as a weighted linear combination of the four graph-grounded metrics: $R_{fid} = w_1 S_{\text{sem}} + w_2 S_{\text{lay}} + w_3 S_{\text{style}} + w_4 S_{\text{color}}$, where $\sum_{i=1}^{4} w_i = 1$. By grounding all four dimensions in the hierarchical graph, this metric provides stable, highly interpretable, and computationally efficient signals for our two-stage training.

4 Experiments

4.1 Benchmarks

Three challenging benchmarks are used in our experiments: *RealChart2Code* (Zhang et al., 2026), *ChartMimic* (Yang et al., 2025), and *Plot2Code* (Wu et al., 2025). Appendix B presents their details and specialized metrics.

4.2 Main Results

Table 1 compares ViCo against prevalent closed- and open-source models (Scores of the compared models are reported in (Zhang et al., 2026)). Results demonstrate that ViCo achieves outstanding code compilation reliability, consistently outperforming both open- and closed-source models in **Pass Rate (Pass %)**. On RealChart2Code, ViCo (8B) secures a **98.8%** pass rate, surpassing Claude-4.5-Opus (87.7%) and Intern-VL-3.5-241B (54.3%). It also reaches **100.0%** on ChartMimic (Customized) and **99.1%** on Plot2Code. This represents a substantial improvement over the Qwen3-VL base models, showing absolute gains (↑) of up to 18.6% in pass rates and significant boosts across visual/textual scores (e.g., a +2.5 rating gain

Table 1: Evaluation results on three benchmarks. *: Only the chart replication task is evaluated. The best score within each category is **bolded** and the second-best is underlined. †: Customized replication is achieved through a post-hoc data refinement. (↑): the absolute performance gain of ViCo against the Qwen3-VL base models (4B/8B).

Model	Size	RealChart2Code*		ChartMimic				Plot2Code		
		Pass (%)	Score	Direct P.(%)	Direct S.	Cust. P†.(%)	Cust. S†.	Pass (%)	Text	Rating
Closed-source LLMs										
Claude-4.5-Sonnet (Anthropic, 2025)	-	85.8	7.1	**100.0**	90.1	99.5	91.5	93.9	73.3	8.8
Claude-4.5-Opus (Anthropic, 2025)	-	**87.7**	7.8	98.5	92.3	**100.0**	95.1	**99.6**	75.4	9.0
Gemini-2.5-Flash (Google DeepMind, 2026)	-	63.2	5.2	88.5	69.8	89.1	74.4	87.9	72.8	8.6
Gemini-3-Pro-Preview (Google DeepMind, 2026)	-	82.1	**9.0**	97.3	**96.0**	**100.0**	**96.8**	90.9	79.5	**9.6**
GPT-5.1 (Singh et al., 2026)	-	71.2	5.7	97.8	89.8	98.5	91.2	98.5	**82.6**	8.8
Open-source LLMs										
DeepSeek-VL (Lu et al., 2024)	7B	9.7	0.4	41.3	19.7	59.3	37.6	64.4	32.6	2.3
Intern-VL-3.5 (Zhu et al., 2025)	241B	**54.3**	**3.5**	90.5	76.9	90.7	78.4	**90.2**	69.0	7.6
Intern-VL-3.5 (Zhu et al., 2025)	30B	20.3	0.8	85.6	68.1	87.2	68.5	88.6	66.3	6.7
Qwen3-VL (Bai et al., 2025)	235B	49.2	3.3	**91.5**	**80.4**	**92.7**	**81.8**	89.8	**69.8**	**8.9**
Qwen3-VL (Bai et al., 2025)	30B	17.3	0.8	89.5	68.6	89.1	69.9	88.9	65.0	7.9
GLM-4.5V (Hong et al., 2025)	106B	38.4	2.8	87.4	72.8	86.7	71.2	84.1	61.7	5.8
GLM-4.1V (Hong et al., 2025)	9B	12.8	0.7	86.6	70.2	87.1	70.1	74.1	59.1	4.9
MiMo-VL-RL (Team, 2025)	7B	11.0	0.4	83.2	58.5	87.7	64.7	73.5	55.5	4.3
ChartCoder (Zhao et al., 2025b)	-	48.0	**3.5**	90.9	75.3	92.1	76.7	87.9	54.5	4.7
ViCo (Qwen3-VL)	4B	98.4(↑18.5)	3.2(↑1.4)	98.1(↑14.2)	78.8(↑12.5)	100.0(↑16.8)	79.2(↑15.0)	97.2(↑18.6)	69.5(↑12.3)	8.1(↑2.5)
ViCo (Qwen3-VL)	8B	98.8(↑12.2)	3.4(↑0.9)	99.2(↑9.5)	81.5(↑8.2)	100.0(↑7.5)	80.9(↑6.8)	99.1(↑10.3)	78.4(↑8.9)	8.4(↑1.6)

for ViCo-4B on Plot2Code). Furthermore, despite its compact scale, ViCo's alignment scores closely approach those of state-of-the-art closed-source models. On ChartMimic, ViCo (8B) obtains a Direct Score of **81.5** (surpassing Gemini-2.5-Flash's 69.8). On Plot2Code, it achieves a Text score of **78.4** and an 8.4 Rating, which are highly competitive with Gemini-2.5-Flash (72.8 and 8.6, respectively) and Claude-4.5-Sonnet (73.3 and 8.8). While evaluation scores on RealChart2Code are generally low across all evaluated models due to the challenges of complex multi-subplot structures, ViCo (8B) remains highly competitive with a **3.4** point (against ChartCoder's 3.5). Our ablation study (§ 4.3) attributes the degraded performance on multi-subplots primarily to the bottleneck of the visual rewarding mechanism rather than generation failure.

4.3 Ablation Studies

We conduct ablation studies using ViCo (8B) variants to evaluate the individual contributions of our core components. Table 2 summarizes the performance across different configurations.

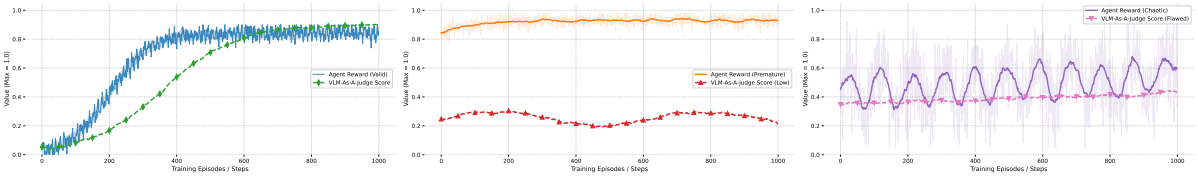
HHLG vs. Open LLM-As-A-Judge. We select 200 samples from RealChart2Code and ChartMimic, categorizing them into simple layouts (no subplots or 1×2 subplots) and complex structures (2×2 subplots or above) for a correlation analysis against human judgment. As shown in Table 3, our HHLG-based metric exhibits a stronger correlation with human judgment across both simple ($\rho = 0.724$) and complex ($\rho = 0.458$) settings.

Table 2: Ablation studies of ViCo (8B) configurations evaluated on RealChart2Code and ChartMimic (Direct Mimic). We report the Pass Rate (%) and visual Score.

Configuration	RealChart2Code		ChartMimic	
	Pass (%)	Score	Pass (%)	Score
ViCo (8B) (**Full Framework**)	**98.8**	**3.4**	**99.2**	**81.5**
w/o RL (Warmup-only)	96.5	3.2	94.2	77.1
w/o SFT (Cold-start RL)	50.5	2.7	69.9	69.7
Rewarding Strategy (Sec. 4.3, Group 1)				
HHLG → LLM-As-A-Judge	94.1	2.8	95.0	72.3
MCTS Filtering & Constraints (Sec. 4.3, Group 2)				
w/o Consistency Checking	96.3	3.1	98.5	76.4
w/o Format Constraints	96.1	3.2	99.4	78.1
w/o Jump-Pruning	96.5	3.2	89.1	77.8
RL Credit Assignment (Sec. 4.3, Group 3)				
w/o Counterfactual (Multi-turn PPO)	47.1	3.0	87.2	75.1
w/o Reflection Credit	49.0	3.1	88.9	77.0

This higher alignment with human explains the ablation results in Table 2 (Group 1): replacing HHLG with Qwen3-VL-8B as the reward model degrades the visual score from 81.5 to 72.3 on ChartMimic and from 3.4 to 2.8 on RealChart2Code. Additionally, the correlation scores of all three reward mechanisms with human preferences decrease sharply on complex layouts (with all Spearman $\rho < 0.5$). This uniform decline reveals that providing precise, automatic visual reward signals for dense, multi-subplot configurations remains a persistent challenge for current models. This reward-utility bottleneck on complex layouts is the primary reason why ViCo's visual score improvement on RealChart2Code.

MCTS Consistency Constraints. We isolate the components of our MCTS-based trajectory sampling framework (Table 2, Group 2). Excluding



(a) Our original PPO (b) Our PPO w/o HHLG-based reward (c) Our PPO w/o refined credit assignment

Figure 3: Training dynamics of multi-step PPO with different rewarding and credit assignment mechanisms.

Table 3: Correlation between different evaluation metrics and human judgment, categorized by chart complexity (*Sim.*: Simple, *Com.*: Complex).

Evaluation Metric	Spearman (ρ)		Pearson (r)		Kendall (τ)	
	Sim.	*Com.*	*Sim.*	*Com.*	*Sim.*	*Com.*
Baselines (LLM-as-a-Judge)						
Qwen3-VL (8B)	0.425	0.184	0.451	0.203	0.318	0.125
GPT-4o	0.712	0.321	**0.738**	0.355	**0.546**	0.241
HHLG-Based (Ours)	**0.724**	**0.458**	0.701	**0.412**	0.503	**0.317**

Table 4: Performance breakdown across layout complexity tiers on RealChart2Code.

Model	Simple (1×1)		Medium (2×2–2×3)		Complex (> 6 subplots)	
	Pass (%)	Score	Pass (%)	Score	Pass (%)	Score
Qwen3-VL-8B	90.8	5.8	78.8	2.8	20.3	0.5
ChartCoder	89.5	5.6	72.3	3.5	28.0	0.7
ViCo (8B)	**100.0**	**8.7**	99.0	6.7	99.0	1.2
+ *Divide-and-Conquer*	–	–	**100.0**	**8.2**	**100.0**	**7.6**

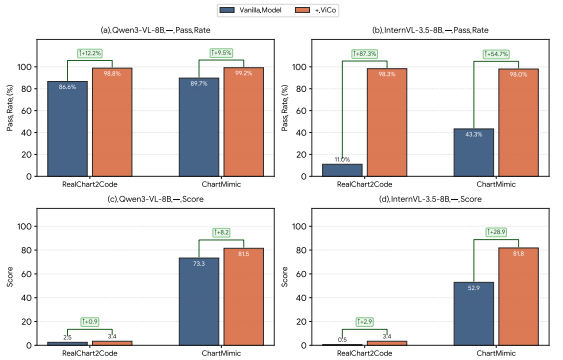


Figure 4: Cross-backbone generalization of ViCo evaluated on RealChart2Code and ChartMimic.

the *consistency critic check* results in a drop in visual score (from 81.5 to 76.4 on ChartMimic and 3.4 to 3.1 on RealChart2Code), indicating the model occasionally ignores its own planned corrections. Removing *format constraints* degrades performance across both benchmarks (e.g., dropping to 78.1 on ChartMimic). Lastly, removing jump-pruning degrades trajectory quality and execution success, causing the code pass rate on ChartMimic to drop to 89.1%. This highlights the necessity of keeping search trajectories focused on strictly positive refinement actions.

Counterfactual-based Credit Assignment. As demonstrated in Table 2 (Group 3), vanilla multi-turn PPO (without counterfactual baselines) suffers from a severe performance decline: the RealChart2Code pass rate plunging from 98.8% to 47.1%, while omitting reflection credit drops the pass rate to 49.0%. These steep declines are visually explained by the training curves in Figure 3. Under our full framework (Figure 3a), the agent reward and the actual visual evaluation score scale synchronously, steadily converging to a high level. This indicates a robust mapping between visual critiques (r_t) and coding edits (a_t). In contrast, when using standard LLM-as-a-judge rewards without HHLG (Figure 3b), the policy quickly exploits weaknesses in the reward model (reward hacking), leading to mismatched training dynamics where agent rewards spike prematurely to near-maximum while the true evaluation scores remain flat and low. Furthermore, removing our credit assignment mechanism (Figure 3c) causes severe credit blurring under sparse terminal rewards. This leads to highly chaotic, periodic fluctuations in agent rewards throughout training, with no substantive gains in visual evaluations.

5 Discussions

5.1 Performance on Complex Layouts

To investigate the effect of structural density on replication fidelity, we evaluated 200 sampled RealChart2Code charts categorized across three complexity tiers: *Simple* (1×1), *Medium* (2×2–2×3), and *Complex* (> 6 subplots, e.g., 3×3 grids). As shown in Table 4, end-to-end ViCo-8B sustains near-perfect code execution ($\geq 99.0\%$) and delivers substantial visual score gains on Simple (8.7 vs. 5.8) and Medium (6.7 vs. 2.8) tiers over the base model. However, direct end-to-end generation on dense Complex grids degrades to a score of 1.2 across all models due to visual grounding and resolution bottlenecks. **Remedy:** To overcome this

limitation, we developed a modular **Divide-and-Conquer pipeline** that decomposes multi-panel figures into individual subplots via bounding-box projection analysis, replicates each panel independently with ViCo, and programmatically stitches them back into the detected canvas grid. This strategy drastically elevates the Complex tier visual score from 1.2 to **7.6** (and Medium to **8.2**) with a 100.0% pass rate, demonstrating a practical and robust pathway for deploying ViCo on high-density academic charts.

5.2 Cross-Backbone Generalization

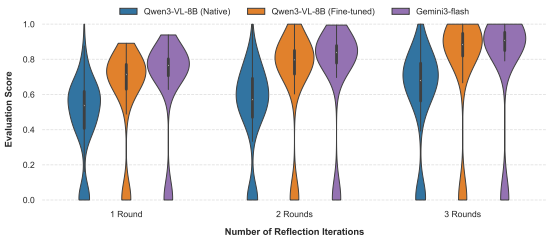
We instantiate and evaluate ViCo on another representative open-source multimodal backbone, **Intern-VL-3.5-8B**, alongside our default backbone **Qwen3-VL-8B**. As shown in Figure 4, the native InternVL-3.5-8B exhibits severe execution and visual alignment deficiencies on code generation. Integrating ViCo yields drastic, across-the-board performance leaps: code pass rates surge to **98.3**% (+87.3%) on RealChart2Code and **98.0**% (+54.7%) on ChartMimic, while visual fidelity scores jump to **3.4** and **81.8** (+28.9%), respectively, matching the competitive performance of ViCo on Qwen3-VL-8B. These consistent, decisive gains across distinct vision-language architectures confirm that ViCo's structured warm-up and counterfactual credit allocation mechanisms are universally transferable rather than reliant on specific model priors.

5.3 Effectiveness of Self-Reflection

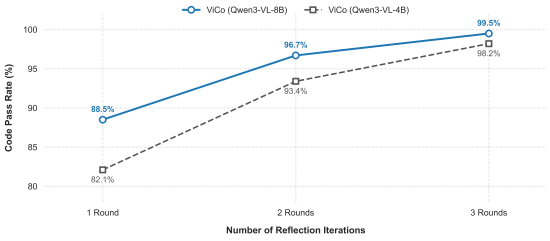
We analyze the performance of ViCo up to three refinement rounds on 200 ChartMimic samples. In Figure 5a, the score distribution of Qwen3-VL-8B remains stagnant and widely dispersed across rounds, ViCo steadily shifts its evaluation scores toward the top range (0.8–1.0) by Round 3, closely matching the trajectory of the proprietary Gemini3-flash. Figure 5b show a monotonic scaling of code pass rates over iterations. This proves that iterative self-reflection, guided by our consistency constraints, systematically resolves execution errors and progressively refines visual alignment.

5.4 Budget of Counterfactual Sampling

We analyze the variance of the counterfactual advantage estimators $A(r_t)$ and $A(a_t)$ across varying sample budgets $M \in \{2, 4, 8, 10\}$. As shown in Figure 6, scaling M from 2 to 8 drastically shrinks gradient variance and prevents destructive policy



(a) Comparison of GPT-4o evaluation scores.



(b) Comparison of code pass rate.

Figure 5: Evaluation against reflection iterations.

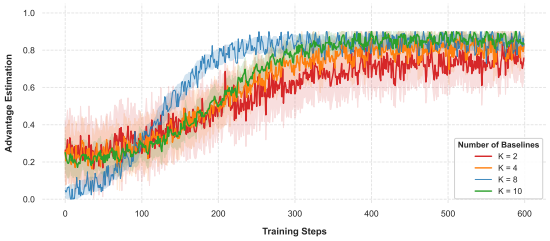


Figure 6: Comparison of advantage estimation in PPO with varying number of counterfactual baselines.

updates caused by noisy Monte Carlo approximations. Beyond $M = 8$, the marginal variance reduction becomes negligible while the wall-clock sandbox execution time scales linearly. This confirms that $M = 8$ strikes an optimal balance, providing sufficiently dense and low-variance credit assignment signals without incurring unnecessary computational overhead during RL training.

6 Conclusion

We propose ViCo, a visual-oriented coding framework that replicates high-quality academic charts through iterative self-reflection. By cascading self-supervised warm-up and counterfactual-based reinforcement learning, ViCo addresses ineffective self-critique and sparse reward encountered in multi-turn refinement. We also propose HHLG, which enables multifaceted reward encompassing chart quality - style, layout, and semantic alignment. With 8B parameters, our trained ViCo models achieves performance close to proprietary LLMs with reflection capabilities across widely used benchmarks.

Limitations

Despite the performance improvements demonstrated by ViCo, several limitations remain to be addressed in future work.

- First, while ViCo achieves strong results on standard replication benchmarks, **complex multi-subplot structures remain a significant challenge**. As observed in our experiments on RealChart2Code, multi-panel layouts (such as dense 2×2 and 3×3 multiple subplots) involve intricate geometric alignments and highly concentrated visual features. These complex arrangements pose severe difficulties for both precise visual-oriented code generation and accurate automatic evaluation. Resolving these challenges likely requires more sophisticated visual-spatial modeling and layout-aware reasoning paradigms.

- Second, **the computational overhead during RL training is relatively high**. Because our counterfactual-based credit assignment mechanism requires curating separate baselines for both the reflection and coding actions at each refinement step, the policy optimization process demands multiple parallel rollouts and counterfactual evaluations. While this design is essential for providing dense, step-level training signals and stabilizing RL without an expensive process reward model, it introduces additional computational burden and training time compared to standard multi-step RL algorithms.

- Third, **the scope of our study is limited to replicating academic chart**. Although our core methodology - encompassing reflective self-training and hierarchical visual layout alignment - is generalizable to a broader spectrum of visual-to-code generation tasks, we have not yet evaluated its efficacy on other complex domains. These include web page replication, user interface (UI) reconstruction, or general visual design translation. Validating and adapting ViCo to these wider, general-purpose coding benchmarks remains an important direction for our future research.

References

Amro Abbas, Kushal Tirumala, Dániel Simig, Surya Ganguli, and Ari S. Morcos. 2023. Semdedup: Data-efficient learning at web-scale through semantic deduplication. *Preprint*, arXiv:2303.09540.

Anthropic. 2025. Claude sonnet 4.5 system card. Technical report, Anthropic. Accessed: May 21, 2026.

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, and 45 others. 2025. Qwen3-vl technical report. *CoRR*, abs/2511.21631.

Jie Deng, Kaichun Yao, and Libo Zhang. 2026. Visrefiner: Learning from visual differences for screenshot-to-code generation. *CoRR*, abs/2602.05998.

Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2974–2982. AAAI Press.

Alireza Ghafarollahi and Markus J. Buehler. 2024. Sciagents: Automating scientific discovery through multi-agent intelligent graph reasoning. *Preprint*, arXiv:2409.05556.

Google DeepMind. 2026. Gemini 3 pro model card. Technical report, Google.

Wenyi Hong, Wenmeng Yu, Xiaotao Gu, Guo Wang, Guobing Gan, Haomiao Tang, Jiale Cheng, Ji Qi, Junhui Ji, Lihang Pan, Shuaiqi Duan, Weihan Wang, Yan Wang, Yean Cheng, Zehai He, Zhe Su, Zhen Yang, Ziyang Pan, Aohan Zeng, and 66 others. 2025. Glm-4.5v and glm-4.1v-thinking: Towards versatile multimodal reasoning with scalable reinforcement learning. *CoRR*, abs/2507.01006.

Muye Huang, Lingling Zhang, Jie Ma, Han Lai, Fangzhi Xu, Yifei Li, Wenjun Wu, Yaqiang Wu, and Jun Liu. 2025. Chartsketcher: Reasoning with multimodal feedback and reflection for chart understanding. *CoRR*, abs/2505.19076.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating LLM hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843, Singapore. Association for Computational Linguistics.

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

Lingjie Jiang, Shaohan Huang, Xun Wu, Yixia Li, Dongdong Zhang, and Furu Wei. 2025b. Viscodex: Unified multimodal code generation via merging vision and coding models. *CoRR*, abs/2508.09945.

Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. 2023. Segment anything. *Preprint*, arXiv:2304.02643.

Jiaqi Liu, Shi Qiu, Mairui Li, Bingzhou Li, Haonian Ji, Siwei Han, Xinyu Ye, Peng Xia, Zihan Dong, Congyu Zhang, Letian Zhang, Guiming Chen, Haoqin Tu, Xinyu Yang, Lu Feng, Xujiang Zhao, Haifeng Chen, Jiawei Zhou, Xiao Wang, and 16 others. 2026. Autoresearchclaw: Self-reinforcing autonomous research with human-ai collaboration. *Preprint*, arXiv:2605.20025.

Xiaoyuan Liu, Tian Liang, Zhiwei He, Jiahao Xu, Wenxuan Wang, Pinjia He, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Trust, but verify: A self-verification approach to reinforcement learning with verifiable rewards. *CoRR*, abs/2505.13445.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Haoyu Lu, Wen Liu, Bo Zhang, Bingxuan Wang, Kai Dong, Bo Liu, Jingxiang Sun, Tongzheng Ren, Zhuoshu Li, Hao Yang, Yaofeng Sun, Chengqi Deng, Hanwei Xu, Zhenda Xie, and Chong Ruan. 2024. Deepseek-vl: Towards real-world vision-language understanding. *CoRR*, abs/2403.05525.

Leland McInnes and John Healy. 2018. UMAP: uniform manifold approximation and projection for dimension reduction. *CoRR*, abs/1802.03426.

Junbo Niu, Zheng Liu, Zhuangcheng Gu, Bin Wang, Linke Ouyang, Zhiyuan Zhao, Tao Chu, Tianyao He, Fan Wu, Qintong Zhang, Zhenjiang Jin, Guang Liang, Rui Zhang, Wenzheng Zhang, Yuan Qu, Zhifei Ren, Yuefeng Sun, Yuanhong Zheng, Dongsheng Ma, and 42 others. 2025. Mineru2.5: A decoupled vision-language model for efficient high-resolution document parsing. *Preprint*, arXiv:2509.22186.

NVIDIA. 2024. Nemo curator: Gpu-accelerated data curation for training ai models.

OpenAI. 2024. GPT-4o system card. *CoRR*, abs/2410.21276.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, Proceedings of Machine Learning Research, pages 8748–8763. PMLR.

Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. 2020. Zero: memory optimizations toward training trillion parameter models. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2020, Virtual Event / Atlanta, Georgia, USA, November 9-19, 2020*, page 20. IEEE/ACM.

Matthew Renze and Erhan Guven. 2024. The benefits of a concise chain of thought on problem-solving in large language models. In *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, page 476–483. IEEE.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Yikang Li, Yupeng Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *CoRR*, abs/2402.03300.

Karen Simonyan and Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Aaditya Singh, Adam Fry, Adam Perelman, Adam Tart, Adi Ganesh, Ahmed El-Kishky, Aidan McLaughlin, Aiden Low, AJ Ostrow, Akhila Ananthram, Akshay Nathan, Alan Luo, Alec Helyar, Aleksander Madry, Aleksandr Efremov, Aleksandra Spyra, Alex Baker-Whitcomb, Alex Beutel, Alex Karpenko, and 467 others. 2026. Openai gpt-5 system card. *Preprint*, arXiv:2601.03267.

Minimax Team. 2025. Mimo-vl technical report. *CoRR*, abs/2506.03569.

Xingqi Wang, Yiming Cui, Xin Yao, Shijin Wang, Guoping Hu, and Xiaoyu Qin. 2025. Charthal: A fine-grained framework evaluating hallucination of large vision language models in chart understanding. *CoRR*, abs/2509.17481.

Xumeng Wen, Zihan Liu, Shun Zheng, Zhijian Xu, Shengyu Ye, Zhirong Wu, Xiao Liang, Yang Wang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. 2025. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. *CoRR*, abs/2506.14245.

Chengyue Wu, Zhixuan Liang, Yixiao Ge, Qiushan Guo, Zeyu Lu, Jiahao Wang, Ying Shan, and Ping Luo. 2025. Plot2code: A comprehensive benchmark

for evaluating multi-modal large language models in code generation from scientific plots. In *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, Findings of ACL, pages 3006–3028. Association for Computational Linguistics.

Yupeng Xie, Zhiyang Zhang, Yifan Wu, Sirong Lu, Jiayi Zhang, Zhaoyang Yu, Jinlin Wang, Sirui Hong, Bang Liu, Chenglin Wu, and Yuyu Luo. 2026. Visjudge-bench: Aesthetics and quality assessment of visualizations. *Preprint*, arXiv:2510.22373.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, Gongye Liu, Xiaomei Nie, Deng Cai, and Yujiu Yang. 2025. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R. Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Jiajun Zhang, Yuying Li, Zhixun Li, Xingyu Guo, Jingzhuo Wu, Leqi Zheng, Yiran Yang, Jianke Zhang, Qingbin Li, Shannan Yan, Zhetong Li, Changguo Jia, Junfei Wu, Zilei Wang, Qiang Liu, and Liang Wang. 2026. Realchart2code: Advancing chart-to-code generation with real data and multi-task evaluation. *CoRR*, abs/2603.25804.

Liang Zhang, Anwen Hu, Haiyang Xu, Ming Yan, Yichen Xu, Qin Jin, Ji Zhang, and Fei Huang. 2024. Tinychart: Efficient chart understanding with visual token merging and program-of-thoughts learning. *CoRR*, abs/2404.16635.

Xuanle Zhao, Deyang Jiang, Zhixiong Zeng, Lei Chen, Haibo Qiu, Jing Huang, Yufeng Zhong, Liming Zheng, Yilin Cao, and Lin Ma. 2025a. Vinci-coder: Unifying multimodal code generation via coarse-to-fine visual reinforcement learning. *CoRR*, abs/2511.00391.

Xuanle Zhao, Xianzhen Luo, Qi Shi, Chi Chen, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2025b. Chartcoder: Advancing multimodal large language model for chart-to-code generation. *CoRR*, abs/2501.06598.

Dandan Zhu, Zhen Lin, Yixuan Weng, Panzhong Lu, Qiujie Xie, Yifan Wei, Sifan Liu, Qiyao Sun, and Yue Zhang. 2026a. Paperbanana: Automating academic illustration for ai scientists. *arXiv preprint arXiv:2601.23265*.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, Zhangwei Gao, Erfei Cui, Xuehui Wang, Yue Cao, Yangzhou Liu, Xingguang Wei, Hongjie Zhang, Haomin Wang, Weiye Xu, and 32 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *CoRR*, abs/2504.10479.

Minjun Zhu, Zhen Lin, Yixuan Weng, Panzhong Lu, Qiujie Xie, Yifan Wei, Sifan Liu, Qiyao Sun, and Yue Zhang. 2026b. Autofigure: Generating and refining publication-ready scientific illustrations. *arXiv preprint arXiv:2601.23261*.

A Related Work

A.1 Code Generation from Visual Inputs

The task of translating visual representations (e.g., charts, UI screenshots) into executable code has garnered significant attention. ChartCoder (Zhao et al., 2025b) pioneered the chart-to-code paradigm by leveraging a Code LLM backbone and the Snippet-of-Thought (SoT) method. Its training relies on static supervised fine-tuning without visual outcome feedback. VinciCoder (Zhao et al., 2025a) proposed a unified multimodal code generation model with a coarse-to-fine visual reinforcement learning (ViRL) strategy. VisCodex (Jiang et al., 2025b) uses task vector merging to combine vision and coding models. More closely related to us are methods that incorporate visual feedback for code refinement. VisRefiner (Deng et al., 2026) introduces a training framework where models learn from visual differences between rendered predictions and target designs, followed by a reinforcement learning stage for self-refinement. However, VisRefiner operates on discrete refinement steps without an explicit structured reflection mechanism that enforces consistency between critique and subsequent code changes. In contrast, our proposed ViCo framework employs a dedicated reflection step with consistency constraints, ensuring that every coding action strictly follows the preceding reflection, thereby addressing the "say one thing, do another" problem.

A.2 Reinforcement Learning with Visual Feedback for Code Generation

Reinforcement Learning with Verifiable Rewards (RLVR) (Wen et al., 2025; Liu et al., 2025) has been increasingly applied to improve code generation quality through visual rewards. VinciCoder (Zhao et al., 2025a) uses DINOv2-based visual similarity as a reward for group relative policy optimization (GRPO) (Shao et al., 2024). VisRefiner (Deng et al., 2026) designs a hierarchical reward including format, refinement, and quality components, also optimized with GRPO. While these methods demon-

strate the effectiveness of visual reward signals, they suffer from crucial limitations. First, the reward is only provided at the end of each generation episode, leading to sparse feedback. Additionally, they lack a mechanism to assign credit to individual reflection and coding actions within a multi-step trajectory. We address these limitations through a novel counterfactual-based reinforcement learning framework. By constructing two baselines - a "no-reflection" baseline to isolate the value of reflection and a "given-reflection" baseline to evaluate action quality - the method assigns dense, step-level advantage signals to both reflection and execution actions. This approach overcomes the reward sparsity problem that plagues standard policy gradient methods like PPO and GRPO when applied to multi-turn reflection trajectories.

A.3 Self-Reflection and Iterative Refinement in LLMs

The concept of self-reflection has been explored in various LLM-based reasoning tasks. ChartSketcher (Huang et al., 2025) introduces a Sketch-CoT mechanism where models annotate intermediate reasoning steps onto charts via a programmatic sketching library, iteratively feeding visual annotations back into reasoning. However, its reflection is grounded in visual sketches rather than code execution outputs, and it does not enforce consistency between reflection and subsequent actions. TinyChart (Zhang et al., 2024) employs Program-of-Thoughts (PoT) learning for numerical reasoning but lacks iterative refinement. Empirical studies on reflection effectiveness reveal two major challenges: models often generate superficial or incorrect reflections (ineffective reflection), and even when reflections are accurate, models frequently fail to implement the suggested changes in the subsequent code (inconsistent execution). Our proposed MCTS sampling with consistency pruning and multi-step reinforcement learning ensures that every output trajectory exhibits strictly increasing rewards and strict adherence to preceding reflections.

B Benchmarks

We evaluate our framework across three challenging benchmarks representing diverse layout granularities:

- **RealChart2Code** is a complex, large-scale benchmark designed to evaluate code generation from real-world datasets across 50 plot types and multi-panel layouts (Zhang et al., 2026).
- **ChartMimic** focuses on reproducing intricate academic-style charts from raw images, specifically testing the model's visual fidelity and layout structure through the *Direct Mimic* and *Customized Mimic* tasks (Yang et al., 2025).
- **Plot2Code** is composed of diverse visualization tasks mapping scientific paper figures directly to execution-ready code (Wu et al., 2025).

B.1 Evaluation Metrics

Functional correctness is measured via the **Pass Rate (%)** (the ratio of generated code that compiles and runs without sandbox runtime errors) (Zhang et al., 2026; Wu et al., 2025). To assess layout, typographic, and stylistic fidelity, we report the **Score** (or **Rating**) evaluated by standard multimodal models-as-a-judge (such as GPT-4o (OpenAI, 2024)) (Yang et al., 2025). For code-level comparisons, we report token-level **Text** similarity (Wu et al., 2025).

C Dataset

C.1 Data Collection and Filtering Pipeline

To cultivate a robust understanding of academic-grade plotting structures and diverse scientific styles, we curate a high-quality visualization training set extracted directly from contemporary computer science and AI research publications.

Source Document Collection. We gather a corpus of 50K PDF papers published in the past three years. To ensure high graphic standards and rigorous formatting, we target publications from top-tier venues (AAAI, ACL, NeurIPS, ICLR, EMNLP, ICML, NAACL) alongside raw preprints from arXiv.

Figure Extraction. For all collected PDF manuscripts, we extract embedded figures and spatial coordinates using MinerU (Niu et al., 2025), an open-source document parsing toolkit capable of isolating structured visual elements from multi-column scientific layouts.

Figure Filtering. Scientific papers contain diverse visual structures (e.g., system diagrams, photos) unsuitable for chart replication code learning. We use VisJudge (Xie et al., 2026), an automated aesthetic and quality assessment model, to filter out

System Prompt

You are an expert in reproducing scientific figures and charts. Given a chart image, you generate a Python script to reproduce it accurately. After thorough reasoning, strictly output code in this format:

````
<code>
```python
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns

(Your generated code goes here)

plt.tight_layout()
plt.savefig("output.png", dpi=150, bbox_inches="tight")
```
</code>
````

Environmental Feedback (Debug)

Code execution error, error message is:

`[Execution Status, Error Traces]`

Please analyze the cause of the error and try again.

non-chart images, heavily distorted plots, structure-naive figures, and low-resolution graphics.

Dataset Derivation. Following filtering, we obtain **over 60K high-quality chart images**, spanning both standalone plots and complex multi-panel subplots. We allocate 50K instances for the Stage-1 warm-up SFT, and retain the remaining 10K instances as training references for Stage-2 reinforcement learning.

C.2 Train-Test De-duplication Protocol

To ensure our evaluation measures true generalization rather than rote memorization, we clarify the benchmark sources and enforce a strict de-duplication pipeline prior to training.

We computed visual similarity between our candidate 60K training pool and all benchmark test sets using three progressive criteria:

1. **Exact Matches:** Cryptographic pixel hashing (MD5) to eliminate identical image files.

2. **Perceptual Hashing (pHash):** Identification of near-duplicates differing only in resolution, compression, or margin padding. We enforce a strict Hamming distance threshold of ≤ 4, a conservative standard in image deduplication engines that isolates identical visual targets with near-zero false matches.

3. **CLIP-based Semantic Filtering:** Identification of charts with highly similar layouts and visual representations. We employ a Cosine Similarity threshold of ≥ 0.88 using CLIP `ViT-L/14` (Radford et al., 2021) embeddings, following standard web-scale semantic pruning practices (Abbas et al., 2023; NVIDIA, 2024). For ChartMimic charts with available metadata, we also cross-match arXiv paper identifiers and exclude figures originating from identical source manuscripts.

Any training instance flagged by these criteria was permanently removed prior to training. Table 5 presents the detailed filtering statistics.

C.3 Latent Space Distribution Analysis

To verify the diversity and coverage of our curated training set ($N \approx 60,000$), we analyze its semantic representation against the evaluation benchmarks. We use the vision encoder of CLIP (`ViT-L/14`) to extract 768-dimensional visual feature vectors, subsequently projected to a 2D manifold using UMAP (McInnes and Healy, 2018) with a Cosine Distance metric.

Environmental Feedback (Reflection)

Please compare the differences between the generated image and the original reference image, summarize improvement methods, and output in the following format:

<improvement> [suggestions for improvement] </improvement>

<code> [improved code] </code>

Based on the above analysis, if you believe that there is no need to make further modifications to the current code, simply output <improvement> DONE </improvement> and end the answer.

Table 5: Statistics of the multi-stage sample-level de-duplication pipeline across evaluation benchmarks.

Benchmark	Exact Match	pHash	CLIP Threshold	Filtered from Train (%)
RealChart2Code (Zhang et al., 2026)	0 (0.00%)	Hamming ≤ 4	Cosine sim. ≥ 0.88	0 (0.00%)
ChartMimic (Yang et al., 2025)	0 (0.00%)	Hamming ≤ 4	Cosine sim. ≥ 0.88	154 (0.26%)
Plot2Code (Wu et al., 2025)	0 (0.00%)	Hamming ≤ 4	Cosine sim. ≥ 0.88	50 (0.08%)

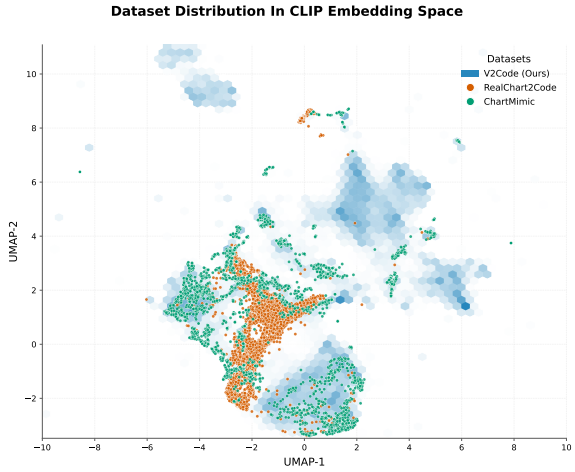


Figure 7: 2D manifold projection of visual representations across datasets.

As illustrated in Figure 7, our training dataset establishes a broad, continuous latent foundation. The distributions of ChartMimic (Yang et al., 2025) and RealChart2Code (Zhang et al., 2026) exhibit substantial distributional overlap with our corpus without forming disjoint clusters. This confirms strong domain alignment, while the multi-stage filtering in §C.2 guarantees sample-level non-contamination.

D Implementation Details

D.1 Sandboxed Sandbox Environment

To prevent code execution safety hazards and standardize environmental variables, all code snippets generated during evaluation and MCTS rollout are executed inside isolated Docker containers based on Python 3.13. Pre-installed libraries include `pandas (2.2)`, `numpy (2.1)`, `matplotlib (3.9.2)`, `seaborn (0.13.2)`, and `scipy (1.14)`. Each container is restricted to 1 CPU core and 4 GB RAM, with a strict execution timeout of 120 seconds.

D.2 Training Configuration

We instantiate ViCo using two open-source vision-language backbones: **Qwen3-VL-4B** and **Qwen3-VL-8B** (Bai et al., 2025). The 4B model serves as a compute-efficient agent, while the 8B variant leverages expanded multimodal capacity to model intricate spatial layouts.

During training, both the visual projection connector and the language model backbone are unfrozen for end-to-end multimodal alignment, while the visual encoder is kept frozen to stabilize visual feature extraction and minimize peak memory usage. All training sessions are accelerated with DeepSpeed ZeRO-3 (Rajbhandari et al., 2020) on an 8×NVIDIA A800 GPU node. Table 6 reports the exact parameter setup distinguished by training phases.

D.3 Hyperparameter Settings

To permanently resolve notation overloading and provide a unified reference, we consolidate all hyperparameters across the MCTS warm-up stage, multi-step PPO reinforcement learning, and the HHLG evaluation framework into Table 7.

Table 6: Hyperparameter settings for Phase 1 (Self-Supervised Fine-tuning, SSFT) and Phase 2 (Multi-step Reinforcement Learning, MSRL) training of ViCo.

Hyperparameter	Phase 1 (SSFT)	Phase 2 (MSRL)
Backbone Model	Qwen3-VL (Bai et al., 2025)	
Optimizer	AdamW (Loshchilov and Hutter, 2019)	
Learning Rate	1×10^{-5}	5×10^{-6}
Learning Rate Schedule	Cosine decay	Cosine decay
Warmup Ratio	0.03	0.05
Batch Size	128	64
Discount Factor (γ)	–	0.95
Counterfactual Samples (K)	–	5
Max Sequence Length (Tokens)	32,768	32,768

E Theoretical and Empirical Analysis of Reflection Advantage

E.1 Logical Derivation: Quality vs. Presence

A critical theoretical question is whether the reflection advantage $A(r_t) = Q_{\text{mid}}(h_t, r_t) - Q_{\text{mid}}(h_t, r_\emptyset)$ merely rewards the superficial *presence* of critique tokens, or genuinely measures the *content quality and utility* of the specific critique r_t.

Because downstream coding actions a_t are sampled directly from the conditioned policy $\pi_\theta(a_t \mid h_t, r_t)$, the expected intermediate value $Q_{\text{mid}}(h_t, r_t)$ causally inherits the validity of r_t:

- **Case 1: Misleading Critique (Poor Content):** If r_t contains hallucinated or erroneous diagnoses (e.g., misclassifying plot types or prescribing incorrect coordinate transformations), it actively biases the subsequent coding policy. The downstream execution is likely to fail or introduce severe graphical deviations. Consequently, the expected reward under this flawed reflection is lower than executing directly without critique ($r_\emptyset$):

$$Q_{\text{mid}}(h_t, r_t) < Q_{\text{mid}}(h_t, r_\emptyset) \implies A(r_t) < 0.$$

  The policy is thus explicitly penalized for generating misleading reflections despite their presence.

- **Case 2: Constructive Critique (High Content Quality):** If r_t correctly diagnoses visual discrepancies and provides actionable instructions, it guides the subsequent coding policy to resolve rendering errors:

$$Q_{\text{mid}}(h_t, r_t) > Q_{\text{mid}}(h_t, r_\emptyset) \implies A(r_t) > 0.$$

  The policy receives positive advantage because the critique yielded positive marginal utility.

Therefore, $r_\emptyset$ functions as an unbiased, neutral control baseline, allowing $A(r_t)$ to measure the true marginal contribution of the reflection's content.

E.2 Computational Tractability of Alternative Baselines

The theoretical ideal for isolating reflection content is a fully marginalized content-contrastive baseline:

$$A_{\text{contrast}}(r_t) = Q_{\text{mid}}(h_t, r_t) - \frac{1}{J} \sum_{j=1}^{J} Q_{\text{mid}}(h_t, r'_j),$$

where $\{r'_1, \ldots, r'_J\}$ are alternative candidate reflections sampled from $\pi_\theta(\cdot \mid h_t)$.

However, computing A_{contrast} is computationally prohibitive. Because rendered code must be compiled and executed inside isolated Docker containers to obtain visual rewards, evaluating Q_{mid} across J reflections requires sampling M coding rollouts each, demanding $\mathcal{O}(J \times M)$ sandbox compilation cycles per optimization step. With $J = 8$ and $M = 8$, this scales training time from tens of GPU hours to hundreds of hours. By contrast, our neutral baseline $r_\emptyset$ requires only $\mathcal{O}(2M)$ rollouts, ensuring stable advantage estimation within accessible academic compute limits.

E.3 Empirical Verification of Advantage Correlation

To empirically verify that $A(r_t)$ aligns with actual step-level visual improvements, we conduct a post-hoc correlation analysis over 200 held-out validation trajectories (comprising 600 discrete reflection-coding refinement steps). For each step t, we measure:

1. The estimated reflection advantage $A(r_t)$.

2. The ground-truth visual consistency gain: $\Delta R_{\text{true}} = R(t) - R(t-1)$, where $R(t)$ denotes the HHLG visual fidelity score of the rendered chart at step t.

Statistical analysis reveals a strong, statistically significant positive correlation between the estimated advantage and the true visual gain:

- **Spearman Rank Correlation (ρ): $\mathbf{0.742}$** ($p < 0.001$)

- **Pearson Correlation (r): $\mathbf{0.718}$** ($p < 0.001$)

This tight alignment confirms that $A(r_t)$ reliably captures meaningful improvements in visual fidelity rather than rewarding token presence.

Table 7: Comprehensive hyperparameter settings across the MCTS warm-up stage, multi-step PPO reinforcement learning, and the HHLG-based reward evaluation framework. Overloaded notations from prior versions have been systematically disambiguated.

Hyperparameter	Symbol	Value	Explanation
Stage 1: MCTS-Based Trajectory Synthesis & Warm-Up			
Max reflection steps	K	3	Maximum iteration depth of the multi-turn coding loop
Branching factor	C	8	Number of candidate code expansions generated per node
Exploration constant	c	1.25	Standard PUCT exploration weight
Maximum patience	$\rho_{\max}$	2	Sequence tolerance for consecutive non-improving steps
Penalty coefficient	β	0.8	Scaling factor for differential history penalty
Consistency threshold	τ	0.8	Strictness limit for matching reflection text to code block
Consistency gain	λ	0.2	Weight coefficient of consistency reward component
Stage 2: Multi-Step PPO Reinforcement Learning			
Counterfactual samples	M	8	Parallel rollouts used to compute step-wise advantage baselines
Discount factor	γ_{rl}	1.0	Temporal decay rate for downstream credit assignment
Clip range	ϵ	0.2	PPO surrogate objective clipping boundary
Reward Engine: HHLG-Based Multi-Dimensional Perceptual Reward			
Semantic weight	w_1	0.25	Importance weight of text OCR alignment component (S_{sem})
Layout weight	w_2	0.25	Importance weight of macro/micro graph layout component (S_{lay})
Style weight	w_3	0.25	Importance weight of VGG Gram texture component (S_{style})
Color weight	w_4	0.25	Importance weight of CIELAB visual mask color component (S_{color})
Bipartite cost weight	α	0.5	Balances Levenshtein distance vs. IoU in semantic bipartite matching
Layout scale factor	β_{lay}	0.1	Scaling factor mapping GED graph distances to layout scores
Style scale factor	γ_{st}	0.05	Scaling factor mapping Gram matrix distances to style scores
Matched visual pairs	$\mathcal{P}$	–	Index set of matched visual node pairs from layout alignment

Table 8: Stage-level performance decomposition of ViCo (8B) evaluated on RealChart2Code and ChartMimic (Direct Mimic).

Training Configuration	RealChart2Code		ChartMimic	
	Pass (%)	Score	Pass (%)	Score
Vanilla SFT (Teacher Pseudo-labels)	85.2	2.4	90.5	68.4
Single-Stage Control Baselines				
MCTS Warm-up (w/o RL)	89.4	2.6	92.1	72.8
Cold-start RL (w/o Warm-up)	62.5	1.8	71.3	45.2
Full ViCo (Warm-up + RL)	**98.8**	**3.4**	**99.2**	**81.5**

F More Ablations Results

F.1 Stage-Level Performance Decomposition

Because our training corpus contains raw rasterized chart images without ground-truth plotting scripts or human reflection logs, constructing a 50K human-written baseline is practically infeasible. To decouple the gains from in-domain supervision, MCTS trajectory curation, and multi-step RL, we construct a strong pseudo-labeled baseline using Qwen3-VL-235B (Teacher). Given an initial program, its rendered image, sandbox execution logs, and the target image, the teacher generates a single fixed reflection-code pair without branching, consistency checks, or MCTS pruning.

As shown in Table 8, MCTS warm-up outperforms static teacher SFT by enforcing reflection-execution consistency. Furthermore, while cold-start RL struggles due to the high initial search entropy of multi-turn code generation, pairing MCTS warm-up with counterfactual RL yields substantial cumulative gains across both compilation pass rates and visual scores.

F.2 Leave-One-Component-Out Reward Ablation

To isolate the marginal contribution of each perceptual dimension within our HHLG reward framework, we conduct a Leave-One-Component-Out (LOO) ablation on the ViCo-4B policy. We retrain both the SFT warm-up and RL stages under four ablated configurations by setting each component weight to 0.0 in turn (with the remaining three dimensions weighted equally at 0.33). Table 9 summarizes the execution pass rates, overall benchmark visual scores, and all four individual HHLG sub-scores on ChartMimic.

The empirical breakdown confirms three key properties:

1. **Decoupled Sensitivity:** Disabling any single

Table 9: Leave-One-Component-Out (LOO) ablation of the HHLG reward formulation across SFT warm-up and RL stages using ViCo-4B evaluated on ChartMimic.

Training Configuration	Pass (%)	Score	S_{sem}	S_{lay}	S_{style}	S_{color}
Phase 1: SFT Warm-Up Stage						
ViCo-4B (Warmup-only)	**89.1**	**69.4**	**75.4**	**69.5**	**63.8**	**65.2**
w/o Semantic	88.2	44.8	32.2	70.8	64.3	65.6
w/o Layout	88.3	46.5	74.8	38.6	58.1	60.2
w/o Style	88.5	58.2	75.6	69.2	36.4	64.9
w/o Color	88.7	57.3	75.1	69.4	63.4	34.8
Phase 2: Multi-Step RL Stage						
ViCo-4B (Full Framework)	**98.1**	**78.8**	**84.5**	**78.8**	**72.6**	**74.3**
w/o Semantic	97.2	51.5	38.2	80.5	73.1	74.9
w/o Layout	**98.1**	53.4	83.9	42.4	66.8	68.3
w/o Style	97.6	68.2	84.7	78.4	44.5	73.8
w/o Color	97.9	66.4	84.2	78.6	72.2	40.5

sub-reward triggers a sharp collapse in its corresponding sub-score (e.g., S_{sem} drops from 84.5 to 38.2 when $w_{\text{sem}} = 0$), while unaffected dimensions remain stable or experience minor relaxation gains due to reduced optimization constraints.

2. **Necessity of Full Composition:** The composite visual score experiences severe degradation across all ablated settings in both stages, proving that visual chart replication requires balanced multi-objective constraints to avoid degenerate solutions (e.g., generating correct layout geometry with mismatched semantic labels).

F.3 Extended RL Training Horizon and Reward Hacking Analysis

To examine whether HHLG is susceptible to delayed Goodhart's law effects or reward hacking under extended policy updates, we prolong policy optimization to 2× the standard training horizon (up to 2,000 steps). We periodically benchmark checkpoints on RealChart2Code by evaluating internal HHLG reward alongside independent GPT-4o visual judging scores.

As demonstrated in Figure 8, the internal HHLG reward and independent GPT-4o evaluations scale synchronously without divergent divergence up to 2,000 steps. Beyond step 1,500, visual fidelity exhibits mild plateauing (3.6 → 3.5) without catastrophic degradation. This highlights the structural robustness of discrete graph-grounded rewards over unconstrained neural judges, confirming that early stopping near 1,000–1,500 steps is optimal for computational efficiency.

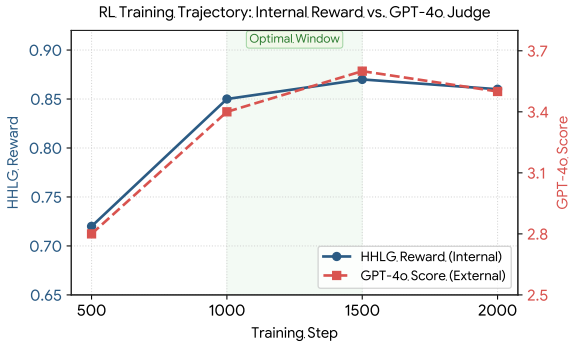


Figure 8: Trajectory of internal HHLG rewards versus independent GPT-4o evaluation scores under extended training horizons on RealChart2Code. Both metrics align synchronously, demonstrating no reward hacking up to 2,000 steps.

G More Analyses

G.1 Parsing Sensitivity and Error Accumulation

Because HHLG computes semantic, layout, style, and color metrics in parallel from a shared graph, an error in one metric does not structurally propagate into others. However, upstream parsing noise (e.g., OCR or segmentation failures) could affect common downstream node extractors. We evaluate this robustness by injecting synthetic noise into 200 human-annotated examples and measuring metric perturbations alongside shifts in Spearman correlation with human judgments ($\Delta\rho_{\text{human}}$).

Table 10 demonstrates that local parsing errors remain strictly confined within their respective modules (e.g., OCR corruption affects only S_{sem}). While macro-level partition shifts (subplot jitter) exert a broader influence, the overall human correlation degradation remains minimal ($\Delta\rho_{\text{human}} = -0.042$), confirming that the composite metric is highly resilient to parsing imperfections.

G.2 Controlled-Budget Inference and Fairness Evaluation

To ensure a strictly fair comparison with respect to inference-time compute, we evaluate models on ChartMimic under uniform execution budgets, restricting the refinement loop to 0 (single-shot / Pass@1), 1, and 2 reflection rounds.

As presented in Table 11:

1. **Single-Shot Strength:** In single-shot generation (Pass@1), ViCo-8B achieves an 88.5% pass

Table 10: Sensitivity analysis of HHLG sub-metrics and human judgment correlation ($\Delta\rho_{\text{human}}$) under controlled upstream parsing noise.

Parsing Perturbation	**Extent**	**Sub-metric Impact** ($\Delta S_{\text{sem}}/\Delta S_{\text{lay}}/\Delta S_{\text{style}}/\Delta S_{\text{color}}$)	$\boldsymbol{\Delta}\rho_{\text{human}}$
OCR token corruption	10%	$(-0.10, -0.02, \ 0.00, \ 0.00)$	-0.015
Visual-node dropout	10%	$(\ 0.00, -0.08, -0.01, -0.07)$	-0.035
Mask erosion/dilation	10%	$(\ 0.00, -0.01, \ 0.00, -0.04)$	-0.012
Text-box jitter	5%	$(\ 0.00, -0.03, \ 0.00, \ 0.00)$	-0.008
Subplot-boundary jitter	5%	$(-0.02, -0.06, -0.05, -0.03)$	-0.042

Table 11: Code execution pass rates (%) under strictly controlled inference budgets on ChartMimic.

Model	**Pass@1 (Single-shot)**	**Reflect×1**	**Reflect×2**
Qwen3-VL-8B	65.6	63.4	65.2
Qwen3-VL-30B	89.5	90.5	90.8
Qwen3-VL-235B	**95.5**	**98.3**	**99.5**
ChartCoder	90.9	83.2	79.5
ViCo-8B	88.5	96.7	99.2

Table 12: Performance comparison between holistic trajectory-level RL baselines and ViCo multi-step counterfactual RL (MSRL) on RealChart2Code and ChartMimic.

RL Method	**RealChart2Code**		**ChartMimic**	
	Pass (%)	Score	Pass (%)	Score
Warmup + GRPO (Terminal reward)	49.7	3.0	89.2	69.7
Warmup + PPO (Terminal reward)	57.1	3.0	89.4	77.1
ViCo (Warmup + MSRL)	**98.8**	**3.4**	**99.2**	**81.5**

rate, significantly outperforming base Qwen3-VL-8B (65.6%) and nearing the 30B model (89.5%), showing that reflective warm-up enhances base code synthesis.

2. **Monotonic Multi-Turn Scaling:** While uncalibrated baselines (such as ChartCoder) suffer from regressive degradation across multiple edits (90.9% → 79.5%), ViCo systematically resolves runtime errors over successive rounds, scaling to 99.2% at Reflect×2 and matching the 235B model.

G.3 Comparison with Trajectory-Level RL Baselines

To validate the effectiveness of our step-wise counterfactual credit assignment against holistic trajectory-level optimization, we compare ViCo MSRL against trajectory-level PPO and GRPO baselines. All methods are initialized from the identical MCTS warm-up checkpoint and trained with equal compute budgets and HHLG reward feedback.

As shown in Table 12, standard trajectory-level GRPO and PPO struggle in alternating reflection-coding environments due to severe credit blurring over long horizons, achieving suboptimal pass rates (49.7% and 57.1% on RealChart2Code). In contrast, ViCo's counterfactual advantage decomposition provides dense, step-specific training signals, substantially boosting both compilation reliability and visual fidelity.

Table 13: Training resource profiling for the 4B parameter model on 8×NVIDIA A800 GPUs.

Method	**GPU Setup**	**GPU Hours (h)**	**Peak Mem. (GiB)**	**Throughput (/s)**
Warmup SFT	8×A800	17.5	554.6	5.94 samples
Multi-turn GRPO	8×A800	14.2	527.4	10.80 rollouts
Multi-turn PPO	8×A800	17.2	551.2	1.42 samples
ViCo MSRL	8×A800	24.4	523.4	0.84 samples

Table 14: Training resource profiling for the 8B parameter model on 8×NVIDIA A800 GPUs.

Method	**GPU Setup**	**GPU Hours (h)**	**Peak Mem. (GiB)**	**Throughput (/s)**
Warmup SFT	8×A800	18.0	571.6	5.80 samples
Multi-turn GRPO	8×A800	18.3	525.5	6.92 rollouts
Multi-turn PPO	8×A800	18.8	573.5	1.39 samples
ViCo MSRL	8×A800	25.6	535.7	0.79 samples

G.4 Computational Cost, Memory, and Throughput Profiling

We benchmark the training efficiency, throughput, and hardware footprints of ViCo against standard multi-turn RL baselines on an 8×NVIDIA A800 GPU node across both 4B and 8B model scales in Tables 13 and 14.

Two key empirical findings emerge:

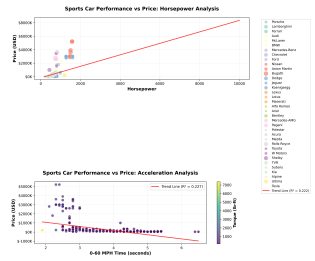
1. **Academic Resource Feasibility:** On a standard single 8×A800 node, full two-stage training converges within 25.6 GPU hours for the 8B model, bypassing heavy proprietary judges and operating entirely within academic laboratory limits.

2. **Favorable Cost-Benefit Tradeoff:** ViCo MSRL requires only an additional 6.8 GPU hours over vanilla multi-turn PPO for the 8B model. This modest computational increase delivers decisive gains in execution pass rates (+41.7% on RealChart2Code) and visual fidelity.

Case Study: rkiattisak/sports car prices

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



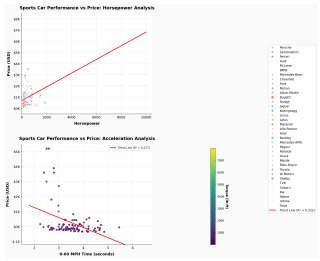
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a line; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure (gridspec).
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Style**: Configure colors, grids, labels to match the target.
5. **Implement**: Write matplotlib code to render the chart.

↓

OUTPUT v0 • after Code v0



Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. **布局优化**：原图采用左侧双图堆叠、右侧上方长图例、右侧下方颜色条的布局。使用 `fig.add_axes` 进行绝对坐标布局可以更精确地还原这种非标准的 GridSpec 结构。
2. **数据与样式细节**：

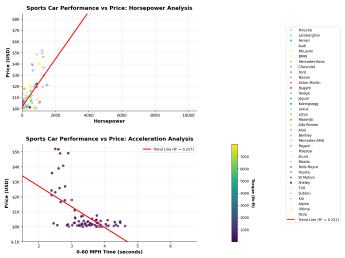
   Y轴格式：原图上图 Y 轴刻度为 `\$0K, \$2000K` 这种带 000 的奇怪格式（可能是 `\$2.0M` 的误写，但视觉上像 K），下图为 `\$0K, \$1K`。为了视觉美观和一致性，统一使用 `\$0K, \$2K, \$4K` 格式，并加粗坐标轴标签。

   散点颜色与大小：上图散点颜色丰富（对应品牌），下图散点统一为深紫色（对应颜色条低值区）。

   图例：手动构建右侧长图例，包含约 36 个品牌及一条趋势线，使用 `Patch` 或 `Line2D` 代理以匹配颜色和大小

↓

OUTPUT v1 • after Code v1



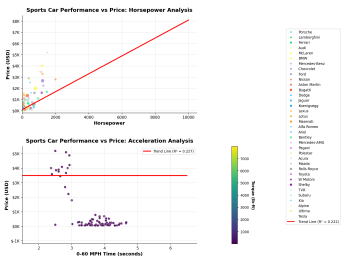
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. **趋势线调整**：原图的上图趋势线斜率较平缓（约从 (0,0) 到 (10000, 8000)），下图趋势线斜率适中（约从 (2, 1500) 到 (5, -1000)）。当前代码通过 `polyfit` 计算的趋势线斜率与原图视觉差异较大（上图过陡，下图过陡）。改进方法：直接硬编码线性方程来绘制红线，以确保…
2. **数据分布微调**：根据调整后的趋势线，重新校准模拟散点数据的分布范围，使散点更合理地围绕趋势线分布，同时保留原图特征性的离群点（如上图 X~1000, Y~5000 的点，下图 X~2.5, Y~5000 的点）。
3. **细节对齐**：进一步微调坐标轴的极限 (`xlim, ylim`)，使刻度线位置与原图完全一致。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 布局优化：原图采用左侧双图堆叠、右侧上方长图例、右侧下方颜色条的布局。使用 fig.add_axes 进行绝对坐标布局可以更精确地还原这种非标准的 GridSpec 结构。
3. **After Reflection v2:** 趋势线调整：原图的上图趋势线斜率较平缓（约从 (0,0) 到 (10000, 8000)），下图趋势线斜率适中（约从 (2, 1500) 到 (5, -1000)）。当前代码通过 polyfit 计算的趋势线斜率与原图视觉差异较大（上图过陡，下图过陡）。改进方法：直接硬编码线性方…

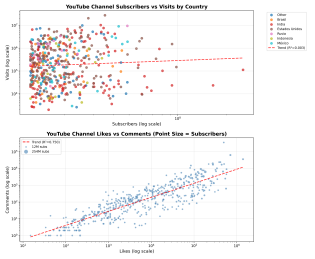
Termination: After 2 rounds of reflection, the model determined the

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: computingvic/top1000youtubers

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



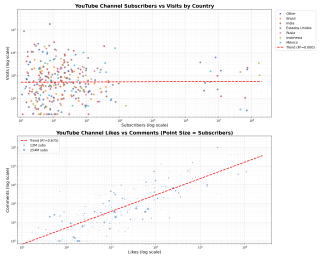
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a subplot; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure (figsize=(16, 12)).
3. **Design data**: Generate mock dataset matching observed visual pattern — 300 points.
4. **Style**: Configure colors, grids, labels, fonts to match the target.
5. **Implement**: Write matplotlib code to render the chart.

↓

OUTPUT v0 • after Code v0



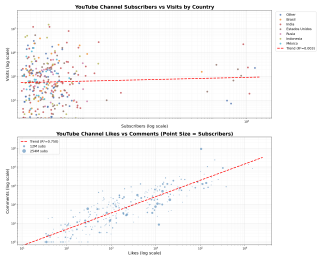
Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. 调整顶部图订阅数分布：用更偏态的 Beta 分布让绝大多数点落在 10^{1}–10^{4}，仅保留约 12 个 $10^{6.5}$–$10^{8.3}$ 的高订阅离群点。
2. 给顶部图 visits 增加 0.04(log10(subs)-1) 的正向漂移，使趋势线呈轻微正斜率；并将图例 R^2 文本固定为原图的 0.003。
3. 顶部图显式设置 x 主刻度为 [1e8]、y 主刻度为 [1e4,1e5,1e6,1e7]，复刻原图的刻度标注方式。
4. 底部图将噪声标准差从 0.75 降到 0.5，使 R^2 接近 0.75（标签固定为 0.750 以精确匹配原图）。
5. … (1 more points omitted for brevity)

↓

OUTPUT v1 • after Code v1



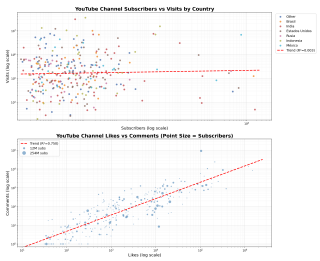
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. 加宽顶部图订阅数主簇的水平分布：将 ls 改为 1 + 5.5 • Beta(1.7, 4.3)，使主簇自然延伸到 10^{3}–10^{5}，保留 12 个 $10^{6.5}$–$10^{8.3}$ 的高订阅离群点。
2. 抬高顶部图 visits 基线：lv 基线从 3.4 提到 3.9，使趋势线左端落在约 1.5×10^{5}、右端约 3×10^{5}，与原图一致，同时增加 10^{6}–10^{7} 区域的点密度。
3. 保持裁剪范围 (3.3, 7.8) 以保留原图底部约 2000 访问量的少量低点和顶部 10^{7+} 的高点。
4. 底部图保持不变。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 调整顶部图订阅数分布：用更偏态的 Beta 分布让绝大多数点落在 10^{1}–10^{4}，仅保留约 12 个 $10^{6.5}$–$10^{8.3}$ 的高订阅离群点。
3. **After Reflection v2:** 加宽顶部图订阅数主簇的水平分布：将 ls 改为 1 + 5.5 • Beta(1.7, 4.3)，使主簇自然延伸到 10^{3}–10^{5}，保留 12 个 $10^{6.5}$–$10^{8.3}$ 的高订阅离群点。

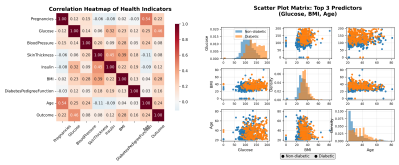
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: vikasukani/diabetes

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



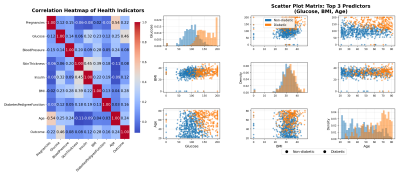
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a correlation ; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure (gridspec).
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Style**: Configure colors, grids, labels, fonts to match the target.
5. **Implement**: Write matplotlib code to render the chart.

↓

OUTPUT v0 • after Code v0



Step 2: Reflection v1 (after Code v0)

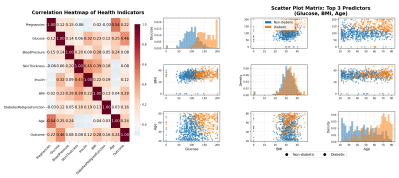
Model compares v0 vs original, suggests improvements

对比发现的主要差异与改进：

1. **热力图配色映射错误（最关键）**：原图对角线 1.00 为深酒红、0 附近为白、负值为浅蓝、0.5 左右为中红，这是以 0 为中心的 diverging 配色（RdBu_r 族）。我之前用 `coolwarm` 且未设 `center=0`，导致 0 被映射成蓝色、白点偏到 0.45，…
2. **散点密度/点大小**：原图散点更细小密集，略调小 s 使观感一致。
3. 其余（0 缺失簇、分组颜色、grid、legend、标题、轴标签）已基本吻合，保留。

↓

OUTPUT v1 • after Code v1



Step 3: Reflection v2 (after Code v1)

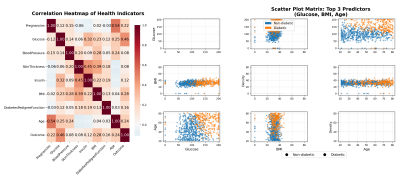
Model compares v1 vs original, refines further

对比当前图与原图，热力图配色（RdBu_r + center=0）、文字颜色、散点分组、0 缺失簇、hist 对角线、legend、标题等均已高度吻合。剩余可改进点：

1. **坐标轴范围不一致**：原图三个变量轴范围固定为 Glucose 0–200、BMI 0–80、Age 20–80（BMI 轴右半基本空白，点云集中在左下）；当前图 BMI 轴仅到 ~45，导致点云铺满子图，与原图留白分布不符。改进：对每个子图按变量强制 `set_xlim/set_ylim` 到上述范围，使…
2. 其余细节（点大小、alpha、grid）已接近，保持不变。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 对比发现的主要差异与改进： 1.
3. **After Reflection v2:** 对比当前图与原图，热力图配色（RdBu_r + center=0）、文字颜色、散点分组、0 缺失簇、hist 对角线、legend、标题等均已高度吻合。剩余可改进点： 1.

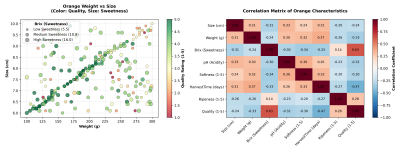
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: shruthiiiee/orange quality

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



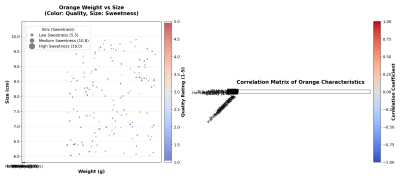
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a scatter plot; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure.
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Style**: Configure colors, labels to match the target.
5. **Implement**: Write matplotlib code to render the chart.

↓

OUTPUT v0 • after Code v0



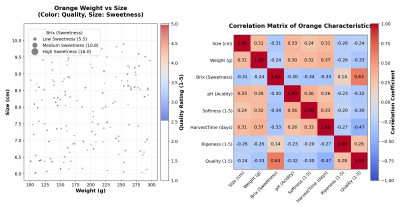
Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. **取消共享X轴**：删除 `plt.subplots` 中的 `sharex=True` 参数。这是因为散点图的X轴范围是 90-310，而 `imshow` 绘制热力图时使用的内部坐标是 0-7。共享X轴导致热力图被强行挤在 90-310 的坐标区间内，从而发生严…
2. **调整对齐方式**：可以使用 `sharey=True` 或者仅仅调整 `height_ratios` 和 `figsize` 来保证两个图表的高度和位置协调一致。
3. **确保热力图正确渲染**：取消 X 轴共享后，`imshow` 将使用默认的 0-7 坐标系，热力图就能正常显示为一个正方形矩阵，标签和数值也能正确放置。

↓

OUTPUT v1 • after Code v1



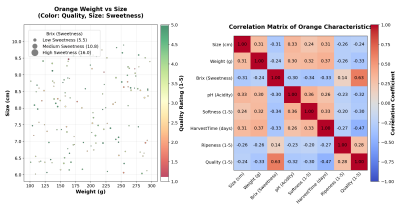
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. **更换左图色板**：将散点图的 `cmap` 从 `coolwarm` 更改为 `RdYlGn`（红-黄-绿），以精确匹配原图中质量评级对应的颜色特征（1.0为红/粉，5.0为绿）。
2. **调整数据分布**：修改散点图 `quality` 值的随机生成逻辑，使用 Beta 分布等使其偏向 1-5 区间的较高值（例如 3.5-5.0），从而在视觉上产生更多绿色和黄色的点，还原原图的颜色密度。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 取消共享X轴：删除 plt.subplots 中的 sharex=True 参数。这是因为散点图的X轴范围是 90-310，而 imshow 绘制热力图时使用的内部坐标是 0-7。共享X轴导致热力图被强行挤在 90-310 的坐标区间内，从而发生严重变形。
3. **After Reflection v2:** 更换左图色板：将散点图的 cmap 从 coolwarm 更改为 RdYlGn（红-黄-绿），以精确匹配原图中质量评级对应的颜色特征（1.0为红/粉，5.0为绿）。

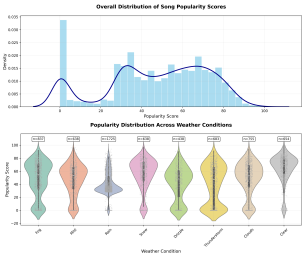
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: receplyasolu/6k weather labeled spotify

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



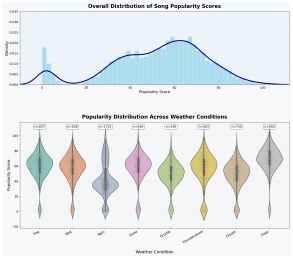
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a distribution ; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure (gridspec).
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Style**: Configure colors, grids, labels, fonts to match the target.
5. **Implement**: Write matplotlib code to render the chart.

↓

OUTPUT v0 • after Code v0



Step 2: Reflection v1 (after Code v0)

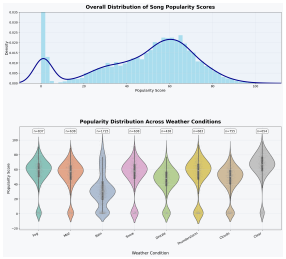
Model compares v0 vs original, suggests improvements

1. 提高低分 bump 占比并收窄其分布（N(1,1.5) 截断到 ⩾0），使 0–2 bin 形成高尖峰，同时因整体 KDE 带宽较宽，0 处 KDE 峰保持适中宽度。
2. 将主体高斯中心重新分散：Rain 左移至 30 形成左峰，Drizzle/Clouds 置于 50/52 形成中间平台，Fog/Mist/Snow/Thunderstorm/Clear 置于 58–68 形成右峰，从而在合并直方图中得到宽双肩+平台结构。
3. 增大主体标准差（≈11，Rain≈9）让峰更宽、与参考的平台感一致。
4. 底部散点改为更浅灰、更低 alpha 以贴近参考；适当增大两子图间距。

其余布局、配色、标注、坐标范围保持不变。

↓

OUTPUT v1 • after Code v1



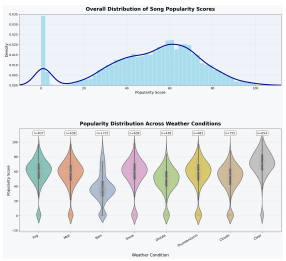
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. 降低低分 bump 占比并设其分布为 N(1,1.2)，使 0–2 柱约为 0.030–0.033，仍是最高尖峰但不再顶格。
2. 主体改为"宽胖双肩"：标准差统一增大到 13（摊宽成宽峰+平台），各 condition 中心对齐参考底部主峰（Fog63/Mist60/Rain35/Snow62/Drizzle52/Thunder62/Clouds55/Clear72），Rain 单独用较小 sd=9 抬升左肩，并保留 Rain 在 70 的细尖（sub, w=0.20）。
3. 其余布局、配色、抖动散点、n 标注、坐标范围保持不变。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 提高低分 bump 占比并收窄其分布（N(1,1.5) 截断到 ⩾0），使 0–2 bin 形成高尖峰，同时因整体 KDE 带宽较宽，0 处 KDE 峰保持适中宽度。
3. **After Reflection v2:** 降低低分 bump 占比并设其分布为 N(1,1.2)，使 0–2 柱约为 0.030–0.033，仍是最高尖峰但不再顶格。

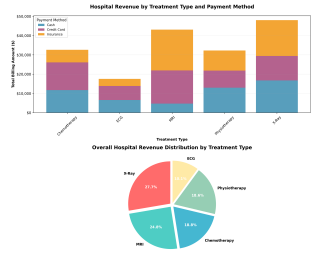
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: kanakbaghel/hospital management

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



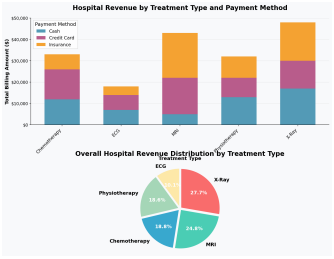
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a distribution ; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure.
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Implement**: Write matplotlib code to render the chart.
5. **Step 5**: Refine rendering details.

↓

OUTPUT v0 • after Code v0



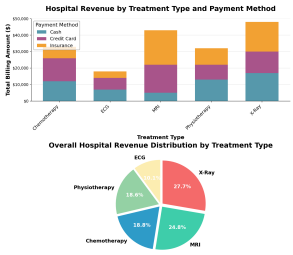
Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. 使用 `GridSpec` 或 `subplots` 的 `hspace` 参数增加上下子图的垂直间距，确保“Treatment Type”标签与下方标题不冲突。
2. 将饼图数据按照顺时针顺序排列：`['X-Ray', 'MRI', 'Chemotherapy', 'Physiotherapy', 'ECG']`，并设置 `startangle=90` 和 `counterclock=False`。
3. 精确调整所有图表元素的颜色（柱状图的三种支付方式，饼图的五个治疗类型）以匹配原图。
4. 使用 `matplotlib.ticker.FuncFormatter` 格式化Y轴刻度，添加 `\$` 和千分位逗号。
5. … (1 more points omitted for brevity)

↓

OUTPUT v1 • after Code v1



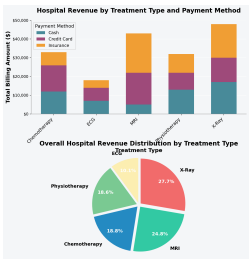
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. 将 `fig` 和 `ax` 的背景色修改为浅灰蓝色（例如 `#f4f6f8`），以完美匹配原图的底色。
2. 将柱状图的网格线颜色设置为白色（`#ffffff`）并调整透明度，使其在浅灰蓝背景上清晰可见。
3. 调整图例参数，添加圆角边框效果（通过 `fancybox=True` 和 `shadow=False`），并微调其字体大小和位置。
4. 减小 `GridSpec` 或 `subplots` 中的 `hspace` 值，使上下子图更紧凑；微调标题的 `pad` 参数以优化垂直间距。
5. … (1 more points omitted for brevity)

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 使用 GridSpec 或 subplots 的 hspace 参数增加上下子图的垂直间距，确保“Treatment Type”标签与下方标题不冲突。
3. **After Reflection v2:** 将 fig 和 ax 的背景色修改为浅灰蓝色（例如 f4f6f8），以完美匹配原图的底色。

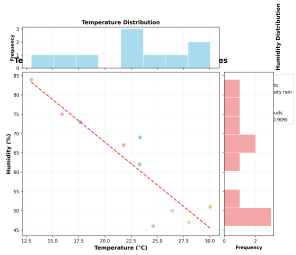
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: aslawant/city weather

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



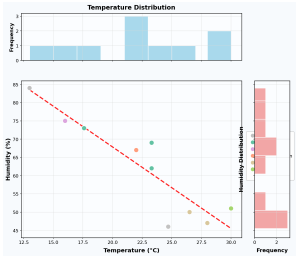
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a pair plot; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure (gridspec).
3. **Design data**: Generate mock dataset matching observed visual pattern — 10 points.
4. **Style**: Configure colors, grids, labels, fonts to match the target.
5. **Implement**: Write matplotlib code to render the chart.

↓

OUTPUT v0 • after Code v0



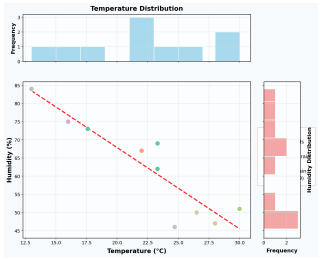
Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. Move the "Humidity Distribution" ylabel to the right side of the right histogram using `ax_r.yaxis.set_label_position("right")` and adjust `labelpad` so it si…
2. Increase the legend font size and marker size, and fine-tune its `bbox_to_anchor` so it overlaps the right histogram similarly to the reference, with the right edge exte…
3. Keep all data coordinates, trend line, colors, grids, and titles as they already match well.

↓

OUTPUT v1 • after Code v1



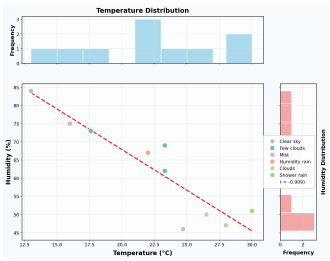
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. 调整 axes 层级：将 `ax_sc` 的 zorder 设为高于 `ax_r`，并给 legend 显式 `set_zorder(10)`，确保图例 markers 绘制在右侧直方图之上、不被遮挡。
2. 增大 figure 宽度并微调 `wspace`，让右侧直方图与 ylabel 右移，使图例 markers 列与 ylabel 分离，复现原图"markers 可见 + 文本右端裁切 + ylabel 在最右"的布局。
3. 将图例 `bbox_to_anchor` 调整为 `(0.99, 0.5)`、`loc='center left'`、`fontsize=12`，使 markers 落在直方图左部可见区，文本右端自然被裁切，贴近原图效果。
4. 数据、直方图 bin、趋势线、颜色、标题等已与原图一致，保持不变。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** Move the "Humidity Distribution" ylabel to the right side of the right histogram using ax_r.yaxis.set_label_position("right") and adjust lab…
3. **After Reflection v2:** 调整 axes 层级：将 ax_sc 的 zorder 设为高于 ax_r，并给 legend 显式 set_zorder(10)，确保图例 markers 绘制在右侧直方图之上、不被遮挡。

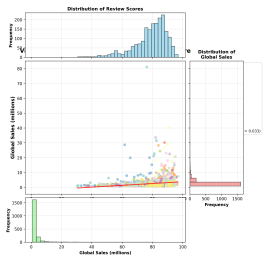
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: thedevastato/global video game sales

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



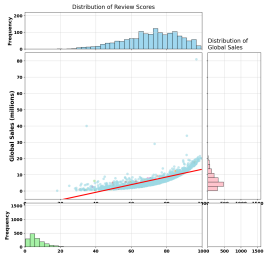
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a scatter plot; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure (gridspec).
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Implement**: Write matplotlib code to render the chart.
5. **Step 5**: Refine rendering details.

↓

OUTPUT v0 • after Code v0



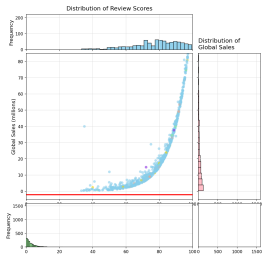
Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. **全局样式**：移除字体加粗设置，减小全局坐标轴标签和刻度的字号，调整子图间距（hspace/wspace），解决文字重叠和排版过大的问题。
2. **数据分布与趋势**：重新设计散点图的数据生成逻辑。将 `x_data` 调整为集中在 70-100 之间的右偏分布；将 `y_data` 改为指数增长模式加噪声，使右侧数据呈现明显的陡峭上升趋势，且整体 Y 值偏低（大部分在 15 以下）。
3. **特征离群点**：在数据中手动注入原图中的标志性离群点（如 (75, 29), (35, 40), (95, 81) 等），以精确还原视觉特征。
4. **散点颜色**：修改颜色映射，使绝大多数散点（约95%）显示为浅蓝色，仅极少数随机点显示为其他颜色（黄、红等），以匹配原图的纯净感。
5. … (2 more points omitted for brevity)

↓

OUTPUT v1 • after Code v1



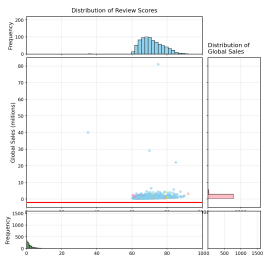
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. **调整散点图数据分布趋势**：原图中散点图的增长曲线相对平缓，大部分数据点集中在右下角（X: 70-100, Y: 0-20），仅有几个离群点较高。当前生成的图中，数据点呈陡峭的指数爆炸式增长，大量堆积在右上角，严重偏离原图。
2. **改进方法**：修改 `y_data` 的生成公式，降低指数项的系数和基础值，使整体趋势变缓；精确调整手动注入的离群点坐标（如将最高点限制在 Y=80 左右，次高点在 Y=40 左右），以完美复现原图的视觉特征。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

1. **Initial attempt (v0):** First code generation based on direct image observation.
2. **After Reflection v1:** 全局样式：移除字体加粗设置，减小全局坐标轴标签和刻度的字号，调整子图间距（hspace/wspace），解决文字重叠和排版过大的问题。
3. **After Reflection v2:** 调整散点图数据分布趋势：原图中散点图的增长曲线相对平缓，大部分数据点集中在右下角（X: 70-100, Y: 0-20），仅有几个离群点较高。当前生成的图中，数据点呈陡峭的指数爆炸式增长，大量堆积在右上角，严重偏离原图。

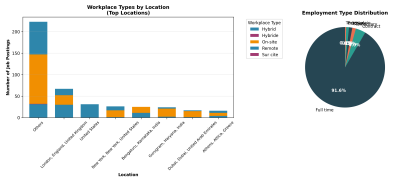
Termination: After 2 rounds of reflection, the model determined the generated code required no further modifications.

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: moyukhbiswas/job postings

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



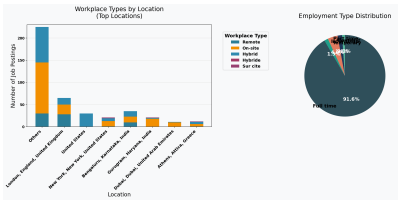
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a distribution; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure.
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Implement**: Write matplotlib code to render the chart.
5. **Step 5**: Refine rendering details.

↓

OUTPUT v0 • after Code v0



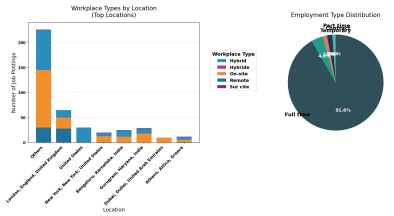
Step 2: Reflection v1 (after Code v0)

Model compares v0 vs original, suggests improvements

1. 调整左图图例位置至 `bbox_to_anchor=(1.05, 0.6)`，并手动指定 `handles` 和 `labels` 以匹配原图顺序（Hybrid 在最上）。
2. 增加柱状图 `width` 参数至 0.75 左右。
3. 重新校准颜色：Remote 使用更深的蓝色 `#1f729e`，Hybrid 使用较亮的蓝色 `#2b8cbe`。
4. 全局设置 `font.weight` 为 `bold`，但在设置标题时单独指定 `fontweight='normal'`。
5. … (2 more points omitted for brevity)

↓

OUTPUT v1 • after Code v1



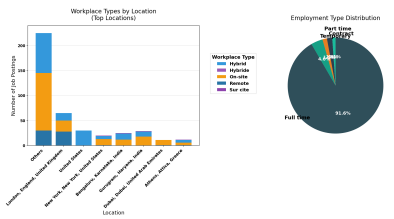
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

1. **修正数据**：调整 Others (去掉顶部 Hybride)、United States (纯 Hybrid)、Dubai (去掉 Hybrid)、Bengaluru/Gurugram (Remote 设为 0) 的数据，使其与原图视觉高度和颜色分布一致。
2. **优化颜色**：更新颜色调色板，Hybrid 使用 #3498db，Remote 使用 #2874a6，Hybride 使用 #9b59b6，Sur cite 使用 #8e44ad，On-site 保持 #f39c1…
3. **增加宽度**：将柱状图 `width` 参数从 `0.75` 增加到 `0.8`，使其更饱满。
4. **微调标签**：微调饼图顶部 `"Part time", "Contract", "Temporary"` 的 `annotate` 坐标，使重叠效果更自然。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

- **Initial attempt (v0):** First code generation based on direct image observation.
- **After Reflection v1:** 调整左图图例位置至 `bbox_to_anchor=(1.05, 0.6)`，并手动指定 `handles` 和 `labels` 以匹配原图顺序（Hybrid 在最上）。
- **After Reflection v2:** 修正数据：调整 Others (去掉顶部 Hybride)、United States (纯 Hybrid)、Dubai (去掉 Hybrid)、Bengaluru/Gurugram (Remote 设为 0) 的数据，使其与原图视觉高度和颜色分布一致。

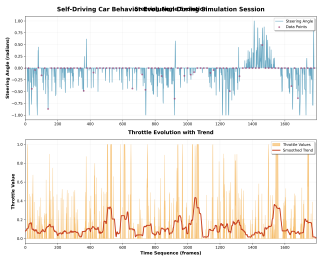
Termination: After 2 rounds of reflection, the model determined the generated code required no further

The model believes that the code needs to be further improved, but the rounds are capped.

Case Study: aslanahmedov/self driving carbehavioural clon

4×2 layout: each row = model's observation (left) & reasoning (right)

INPUT • Original Target



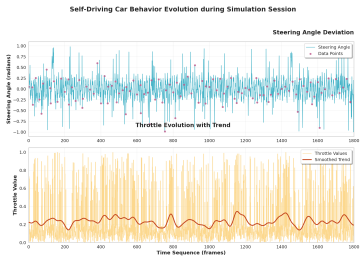
Step 1: Initial Planning (before code)

Model analyzes target image and outlines plan

1. **Analyze**: Identify the chart as a subplot; locate subplots, axes, and visual elements.
2. **Plan layout**: Design the matplotlib structure.
3. **Design data**: Generate mock dataset matching observed visual pattern.
4. **Implement**: Write matplotlib code to render the chart.
5. **Step 5**: Refine rendering details.

↓

OUTPUT v0 • after Code v0



Step 2: Reflection v1 (after Code v0)

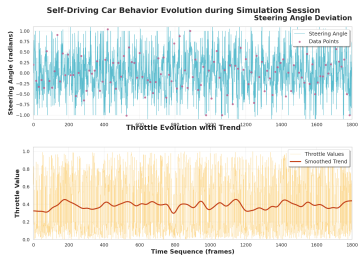
Model compares v0 vs original, suggests improvements

1. **重构标题系统**：使用 `fig.suptitle` 设置顶部标题；使用 `ax1.text` 或 `ax1.set_title(loc='right')` 将 "Steering Angle Deviation" 定位至右上角；增加子图间距 `hspa…`
2. **重做数据模拟**：

   - 上图：增加数据点数量（如 1800 个点），使用叠加的高频随机噪声和随机尖峰（Spike）来模拟极端的转向偏差。
   - 下图：生成密集的随机数据，大部分值低，少部分随机跳高，使用 `drawstyle='steps-post'` 或极高密度的折线来模拟原图的密集填充感。平滑趋势线使用高斯滤波处理密集数据。

↓

OUTPUT v1 • after Code v1



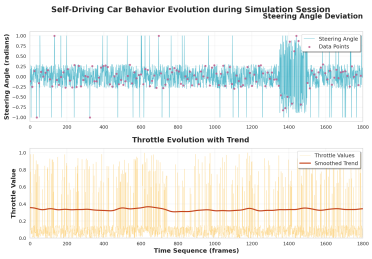
Step 3: Reflection v2 (after Code v1)

Model compares v1 vs original, refines further

- **重构标题排版**：使用 `fig.text` 绘制顶部标题，并利用 `ax1.set_title` 调整 y 轴偏移和 x 轴对齐（`loc='right'`），使 "Steering Angle Deviati…
- **精细控制上图数据**：生成基础的高频低幅值噪声（0~0.3）。对 1350:1500 索引区间单独生成高频中高幅值噪声（0.4~0.9）。最后随机插入少量触及 -1.0 和 1.0 的极值尖峰。
- **优化下图数据生成**：将基础油门值设定为极低的 0~0.1。大幅减少高值脉冲的数量（约 200 个），使其呈现偶发跳高的细线形态，从而还原原图底部密集、上部稀疏的视觉感受。

↓

OUTPUT v2 • FINAL (after Code v2)



Final Result & Iteration Summary

After applying two rounds of reflection

Iteration trajectory:

- **Initial attempt (v0):** First code generation based on direct image observation.
- **After Reflection v1:** 重构标题系统：使用 `fig.suptitle` 设置顶部标题；使用 `ax1.text` 或 `ax1.set_title(loc='right')` 将 "Steering Angle Deviation" 定位至右上角；增加子图间距 `hspace`，并使用 `fig.text` 在垂直居中位…
- **After Reflection v2:** 重构标题排版：使用 `fig.text` 绘制顶部标题，并利用 `ax1.set_title` 调整 y 轴偏移和 x 轴对齐（`loc='right'`），使 "Steering Angle Deviation" 紧贴并部分重叠在主标题下方。中间标题通过 `fig.text` 垂直居中放置于两

The model believes that the code needs to be further improved, but the rounds are capped.