VinciCoder: Unifying Multimodal Code Generation via Coarse-to-fine Visual Reinforcement Learning

Anonymous ACL submission

Abstract

While recent specialized multimodal code generation models excel in tasks like chart-to-code generation, their reliance on single-task training limits generalization and hinders the development of **VI**sio**N** **C**ode **I**ntelligence. In this work, we introduce **VinciCoder**, a unified framework designed for generalized multimodal code generation. We first curate a large-scale SFT corpus comprising 1.3M direct generation pairs and 300k visual-based refinement tasks. This composition fosters self-refinement capabilities, enabling the model to directly rectify code to align with input images. Subsequently, we propose coarse-to-fine Visual Reinforcement Learning (ViRL) to overcome the brittleness of textual metrics in handling semantically equivalent but syntactically diverse code. By quantifying visual similarity across multi-scale patches, ViRL provides an implementation-agnostic reward mechanism that ensures high-fidelity alignment between rendered outputs and input visuals. Extensive experimental results across diverse benchmarks demonstrate that VinciCoder achieves superior performance, while comprehensive ablation studies validate the effectiveness of our proposed ViRL strategy.

1 Introduction

Recent advancements in Large Language Models (LLMs), such as Gemini-2.5 (Comanici et al., 2025) and Qwen3-Coder (Yang et al., 2025a), have achieved significant breakthroughs in code generation, demonstrating a robust capability to follow complex instructions and synthesize executable code across various programming languages. Beyond textual descriptions, a growing body of research (Wan et al., 2024; Jiang et al., 2025b) explores multimodal code generation from visual inputs like charts and webpages, which are inherently more information-dense than natural language and thus present greater challenges for model comprehension and synthesis.

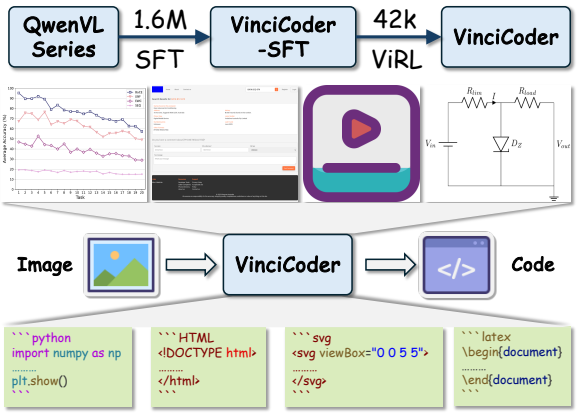


Figure 1: The VinciCoder Framework. Built upon the Qwen-VL series via a progressive SFT-ViRL pipeline, VinciCoder bridges visual perception and code synthesis across diverse modalities.

However, current research predominantly develops task-specific vision-language models (VLMs) such as ChartCoder (Zhao et al., 2025b) for charts and Web2Code (Yun et al., 2024) for web-to-HTML. While effective in their respective domains, these models rely on narrow training scopes that restrict generalization and purely supervised fine-tuning (SFT) that fails to ensure both high code executability and visual fidelity. Furthermore, while Reinforcement Learning with Verifiable Reward (RLVR) has seen broad cross-domain adoption, its application to code generation is hindered by brittle, rule-based rewards that induce instability in diverse, open-ended scenarios due to their sensitivity to semantically equivalent but syntactically diverse code. Consequently, developing a unified multimodal code generation framework (Jiang et al., 2025a; Sun et al., 2025) integrated with style-agnostic RL has emerged as a vital yet challenging research direction.

To address these challenges, we introduce VinciCoder, a unified VLM that employs a two-stage training strategy for multimodal code generation. In the SFT stage, we curate a large-scale corpus of 1.3M direct generation samples and pioneer the

integration of a non-trivial visual-based refinement task using 300k additional samples. Unlike conventional editing, this task requires the model to rectify flawed code snippets containing logical errors or partial renderings, ensuring their visual outputs precisely align with target images. By incorporating this data, we equip VinciCoder with multi-turn debugging and refining capabilities to iteratively improve code quality. Subsequently, we propose Visual Reinforcement Learning (ViRL) to enhance code executability and visual fidelity. By pivoting the reward mechanism to the visual domain, ViRL utilizes direct perceptual similarity as a style-agnostic signal, ensuring stable optimization that is indifferent to specific coding implementations. To robustly compare multi-resolution images, our ViRL framework employs a coarse-to-fine reward mechanism that simultaneously captures global structural layouts from downsampled views and fine-grained local details from segmented patches.

We conduct extensive experiments across diverse multimodal benchmarks, comparing VinciCoder against leading contemporary models. The results demonstrate the effectiveness of our two-stage strategy, as VinciCoder-SFT already establishes a strong baseline that surpasses existing models. The subsequent ViRL further enhances performance, setting a new frontier across all tasks. Furthermore, ablation studies on cold-start ViRL settings (w/o SFT) confirm that our proposed ViRL strategy consistently improves model performance independently. Beyond direct generation, the visual refinement task empowers VinciCoder to debug and refine code snippets using ground-truth visual feedback. To our knowledge, VinciCoder is the first unified model to leverage RL for domain-agnostic visual fidelity in multimodal code generation.

- We introduce VinciCoder, a unified VLM for generalized multimodal code generation. By pioneering a non-trivial visual-based refinement task, we empower the model with self-refinement capabilities to rectify code discrepancies.
- We propose coarse-to-fine ViRL, a framework that enables stable optimization for semantically equivalent but syntactically diverse code by shifting from brittle textual rewards to implementation-agnostic visual similarity.
- Extensive evaluation results across diverse benchmarks demonstrate the superior performance of VinciCoder. Ablation results further demonstrate that our method significantly enhances both code executability and visual fidelity.

2 Related Works

2.1 MLLMs for Code Generation

Multimodal code generation has gained significant traction, particularly in synthesizing code for visual artifacts such as charts, webpages, SVGs, and scientific plots. Current research in multimodal code generation spans several specialized domains. In charts and scientific plots, works like ChartCoder (Zhao et al., 2025b) and MathCoder-VL (Wang et al., 2025a) focus on large-scale corpora and real-world datasets (e.g., Datikz (Belouadi et al., 2025)), while others integrate RL (Chen et al., 2025a) to enhance visual fidelity. Similarly, web-to-code research leverages both synthetic pairs (Yun et al., 2024) and real-world data (Gui et al., 2025), with models like LayoutCoder (Wu et al., 2025) introducing layout-aware frameworks for structural accuracy. In the SVG domain, StarVector (Rodriguez et al., 2025a) and OmniSVG (Yang et al., 2025d) utilize paired icons and illustrations to bridge the gap between images and scalable graphics. Furthermore, approaches like Cosyn (Yang et al., 2025e) utilize LLMs to synthesize code across multiple visual domains, enabling the rendering of informative images from textual descriptions. However, these methods are task-specific, as they are constrained to homogeneous visual patterns and singular programming languages. While recent research has shifted toward unified models (Jiang et al., 2025a; Sun et al., 2025), these efforts rely almost exclusively on SFT, which is insufficient for ensuring both code executability and visual fidelity.

2.2 Reinforcement Learning for MLLM

Inspired by the success of Group Relative Policy Optimization (GRPO) in post-training LLMs (Shao et al., 2024; Guo et al., 2025), RL has garnered significant attention from the research community. Recently, a growing body of work has focused on applying RL to MLLMs to enhance their vision-language reasoning capabilities. Pioneering efforts, such as Vision-R1 (Huang et al., 2025), VLM-R1 (Shen et al., 2025), and R1-OneVision (Yang et al., 2025c), initially utilize Chain-of-Thought (CoT) data for SFT to establish a reasoning baseline and employ RL to refine the model policy towards generating coherent answers. This RL framework has proven highly versatile, with successful adaptations

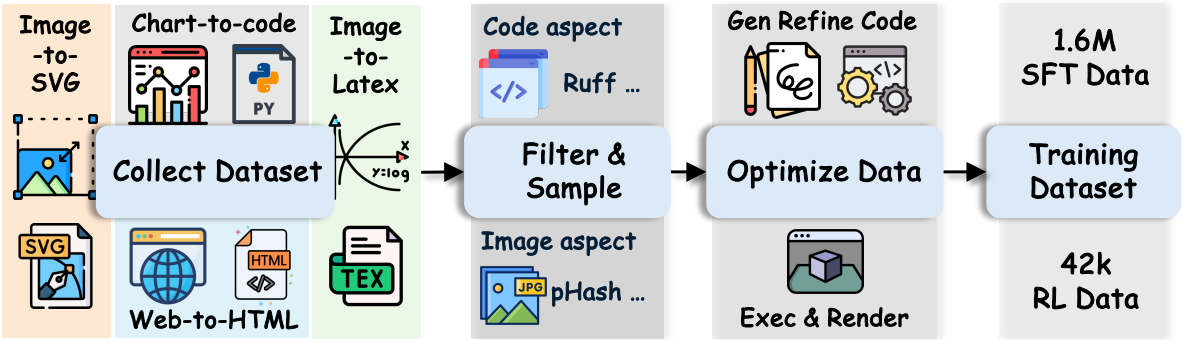


Figure 2: Overview of the VinciCoder data construction pipeline. Following initial filtering and diversity-aware sampling of open-source datasets, the corpus is enhanced through two parallel streams: refining existing samples via execution and optimization, and generating novel samples for the visual refinement task. This dual strategy produces high-quality data pairs for both SFT and RL training stages.

to a range of fundamental visual tasks, including grounding (Zhou et al., 2025; Liu et al., 2025b), question answering (QA) (Tan et al., 2025a; Chen et al., 2025b), and segmentation (Liu et al., 2025a). Besides constructing rewards from generated text contents, another research direction involves leveraging visual feedback to formulate reward functions. For instance, RRVF (Chen et al., 2025d) and RLRF (Rodriguez et al., 2025b) utilize MLLMs and pretrained Vision Transformers (ViTs) to score generated images, thereby providing a reward signal for RL training.

3 Method

3.1 Task Definition

The standard approach for multimodal code generation is to generate code from visual and textual inputs directly. Given an Image and a Text instruction, an MLLM is tasked with generating the corresponding Code.

$$\text{Code} = \text{MLLM}(\text{Image}, \text{Text}) \quad (1)$$

Besides direct generation, we introduce a visual-based refinements task, which tasks the model with refining an initial, potentially flawed code draft $\text{Code}_{\text{draft}}$ to the refined version $\text{Code}_{\text{refined}}$.

$$\text{Code}_{\text{refined}} = \text{MLLM}(\text{Image}, \text{Code}_{\text{draft}}, \text{Text}) \quad (2)$$

3.2 Data Construction

To construct large-scale training data, we curate datasets from various open-source datasets and generate missing image-code types. All code is re-executed to render the corresponding images.

3.2.1 SFT Data

Chart-to-code. For our chart-to-code task, we curate training data from ChartCoder (Zhao et al., 2025b) and MSRL (Chen et al., 2025a), starting with rectifying syntax errors in Chart2Code-160k via automated linters.To ensure diversity, we extract 100k distinct samples from MSRL using perceptual hashing (pHash) and mini-batch K-means clustering. We further construct the code refinement dataset by employing a specialized VLM, trained on our initial corpus, to generate first-pass code for 100k non-overlapping MSRL samples. This dataset pairs imperfect generations with their corresponding target images and ground-truth code, facilitating the training of multi-turn debugging capabilities. We provide the refinement data format in Figure 9.

Web-to-HTML. For the web-to-HTML task, we curate training data from MCD (Jiang et al., 2025a), Web2M (Gui et al., 2025), and Web2Code (Yun et al., 2024). For the Web2M collection, we filter for English-language entries and exclude samples containing hyperlinks or embedded images, yielding a refined dataset of 60k entries. Following a pipeline consistent with our chart-to-code approach, we train a dedicated web-to-code VLM on the combined MCD and filtered Web2M data. This model is then used to generate initial code for 100k instances sampled from the Web2Code dataset to form the final refinement set.

Image-to-SVG. For the image-to-SVG task, our primary corpus comprises 360k image-SVG pairs from the UniSVG ISVGEN subset (Li et al., 2025b). To construct the refinement dataset, we employ a dedicated VLM trained on ISVGEN to generate code drafts for 100k samples from the same subset. This approach is motivated by observed training instabilities, where high and fluctuating final training loss suggests the model has not yet mastered the training set. Consequently, these samples remain non-trivial targets for refinement,

Scientific Plots	Code Types	Statistics
Document	Latex/HTML	71k
Molecule	RDKiT/Indigo	50k
Diagram	Latex/HTML/Mermaid	48k
Table	Latex/HTML	32k
Graphic	SVG/Asymptote	27k
Circuit	Latex	10k

Table 1: Details about scientific plots and corresponding code types in the SFT dataset.

allowing the model to learn error correction even on previously seen data.

Image-to-Latex. For the image-to-LaTeX task, we curate data from the DaTikZ-v3 (Belouadi et al., 2024) and Cosyn-400k (Yang et al., 2025e). To ensure consistent and valid outputs, we standardize the code by encapsulating all instances within a standalone TikZ environment. This procedure is specifically designed to generate tightly-cropped figures, thereby preventing the model from producing full-page A4 PDF layouts. Following this, we execute each sample to validate its integrity, filtering out instances that either result in rendering errors or produce multi-page outputs.

Scientific Plots-to-code. In addition to the aforementioned domains, we extend our investigation to complex scientific visualizations. This expansion covers a broad spectrum of graphical representations, including molecular structures, electronic schematics, general diagrams, document layouts, and tabular data. The underlying code for these figures utilizes both general-purpose languages and specialized domain-specific languages (DSLs) such as Mermaid and Asymptote. Our dataset is primarily constructed from the Cosyn-400k collection and various open-source text-to-Mermaid datasets. We further augment this corpus with 40k molecular image-code pairs, which were generated by rendering SMILES strings sourced from the USPTO database. Table 1 provides a detailed breakdown of the dataset, summarizing the distribution of categories and their corresponding statistics.

3.2.2 RL Data

For the RL phase, we construct a new dataset spanning five distinct domains, ensuring complete mutual exclusivity with our SFT corpus. The curation process for each domain is detailed as follows: (i) Chart-to-code: We directly utilize the 11k image-code pairs from the second RL-stage subset of the MSRL dataset (Chen et al., 2025a). (ii) Web-to-HTML: We sample 9k examples from Web2Code (Yun et al., 2024) and employ Gemini-2.5-Flash (Comanici et al., 2025) to refine the HTML code, using varied reference tags and diverse instructions to enhance the visual complexity and quality of the generated webpages. (iii) Image-to-SVG: Using our trained SFT model as a quality filter, we curate the SVG-icons dataset (Rodriguez et al., 2025a) by identifying hard examples. We retain 10k high-fidelity samples that the current model fails to reconstruct accurately, specifically targeting complex cases that remain challenging for generative synthesis. (iv) Image-to-LaTeX: We extract samples from the arxiv-woc-680k category of the ImgCode-8.6M dataset (Wang et al., 2025a). To rectify issues such as inconsistent output scaling and low content diversity, we implement a hybrid filtering pipeline that combines rule-based heuristics with model-based verification, yielding a final curated set of 10k examples. (v) Chemical Images: We synthesize a dedicated collection of 2k molecular images by rendering unique SMILES strings through both RDKit (Landrum, 2013) and Indigo (EPAM Systems, 2025), ensuring diverse rendering styles for the RL phase.

Image	Code	Data Statistics	
		SFT Data	RL Data
Chart	Python	412k	11k
Webpage	HTML	355k	9k
Image	SVG	463k	10k
Image	Latex	108k	10k
Scientific Plots	Multiple	238k	2k

Table 2: Details about SFT and RL data statistics.

The comprehensive composition of our training datasets is detailed in Table 2. Specifically, the chart-to-code, web-to-HTML, and image-to-SVG subsets are augmented with 100k, 92k, and 103k refinement samples, respectively, while the rest consists exclusively of direct generation data.

3.3 Model Training

3.3.1 Supervised Finetuning (SFT)

The SFT phase is essential for establishing syntactic priors, preventing the model from over-optimizing for visual fidelity at the expense of functional correctness. In niche or less-trained tasks, such as chemical tasks, direct ViRL without this initialization often triggers reward hacking, prioritizing perception similarity over code integrity. Consequently, we first employ a standard autoregressive objective to train the model

$$\mathcal{L}(\theta) := -\mathbb{E}_{(x,y)\sim\mathcal{D}_{\text{SFT}}} \sum_{t=1}^{T} \log P\left(y_t \mid x, y_{<t}; \theta\right), \quad (3)$$

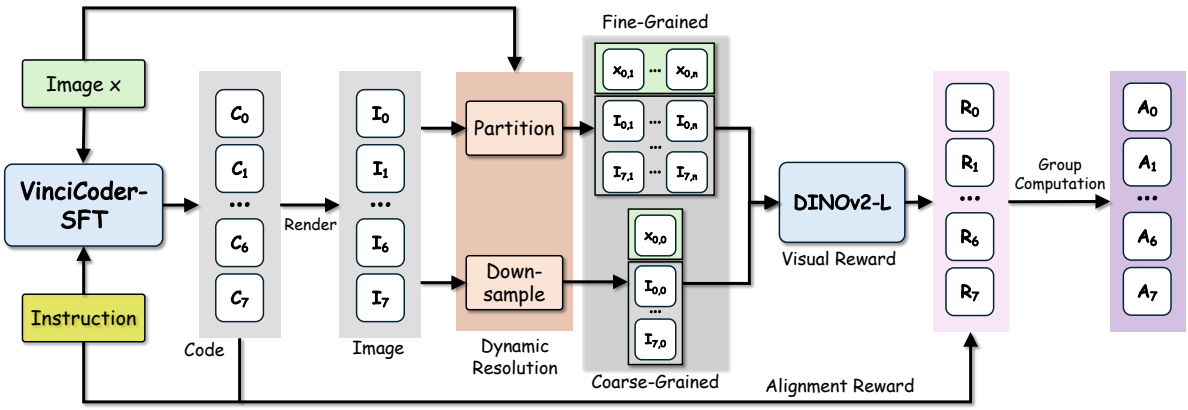


Figure 3: Overview of the ViRL strategy. During training, the alignment reward assesses the structural and linguistic consistency between the generated code and input instructions. Subsequently, for executable rollouts, the coarse-to-fine visual reward is computed by comparing DINOv2 embeddings of global thumbnails and local patches against the target image to evaluate visual alignment.

(x, y) is the query and target response.

3.3.2 Visual Reinforcement Learning (ViRL)

Although SFT achieves strong performance on text-based similarity, this approach faces fundamental limitations in capturing visual-semantic alignment. First, the next-token prediction objective is inherently local and provides no supervisory signals for global functional properties such as code executability. Second, SFT lacks explicit visual grounding as the model remains blind to the rendered output during training. This deficiency is critical because the code-to-visual mapping is highly sensitive, where minor syntax perturbations can induce catastrophic shifts in the rendered images.

To address these issues, we propose ViRL, a reinforcement learning strategy that incorporates coarse-to-fine visual rewards and alignment rewards to optimize visual fidelity and instruction-following consistency, respectively. During the ViRL stage, we employ the Group Relative Policy Optimization (GRPO) (Shao et al., 2024) to optimize the model. The total reward R is a weighted combination of the coarse-to-fine visual reward R_v and the language alignment reward R_l:

$$R = \omega_v R_v + \omega_l R_l \tag{4}$$

where ω_v and ω_l are hyperparameters controlling the relative importance. Figure 3 illustrates the integrated framework of our ViRL strategy.

Coarse-to-fine Visual Reward R_v. Directly downsampling high-resolution images is a suboptimal approach for computing visual similarity, as it inherently discards fine-grained details and may mask significant visual discrepancies. To mitigate this, we propose a coarse-to-fine visual reward mechanism R_v. Given a rendered image I_r and its corresponding ground-truth I_s, we first align their dimensions by resizing the former to the latter. The mechanism then evaluates similarity across two complementary scales: (1) Fine-grained scale, which employs a dynamic partitioning strategy to divide the image into non-overlapping 448×448 patches while preserving the original aspect ratio; and (2) Coarse-grained scale, where a downsampled thumbnail of the entire image is generated to provide global-level context. Similarity scores are computed between corresponding tiles and thumbnails using DINOv2 (Oquab et al., 2023). The final visual reward is the arithmetic mean of the global score and all patch-based scores:

$$R_v = \frac{1}{N+1}\left(R_{\text{global}} + \sum_{i=1}^{N} R_{\text{fine},i}\right) \tag{5}$$

where N denotes the number of fine-grained patches. In instances where the generated code fails to render successfully, R_v is assigned a value of 0. This robust design ensures that fine and coarse rewards naturally converge for low-resolution images, enabling a unified pipeline to handle diverse resolutions.

Alignment Reward R_l. To address the issue where the SFT model generates code in a language inconsistent with the prompt, we introduce an alignment reward to improve instruction fidelity. This reward is determined by checking the consistency between the target language specified in the instruction and the language identifier extracted from the generated code blocks. We use a predefined mapping to handle aliases, such as mapping `tikz` to `latex`. A binary reward of 1 is assigned if the generated language matches a valid alias, ensur-

Model	ChartMimic-direct-v2			UniSVG-ISVGEN			Design2Code		Image2Struct-plot		ChemDraw	
	Exec.Rate	Low-L	High-L	Low-L	High-L	Score	Low-L	High-L	Ren.Succ.	EMS	Exec.Rate	Tani.Sim.
Closed-Source Models												
Gemini-2.5-Pro	97.3	88.7	83.8	53.6	80.3	69.6	90.8	91.4	74.3	52.5	77.3	2.8
Claude-4.5-Sonnet	97.8	89.6	82.9	61.0	83.4	74.6	90.4	90.8	72.7	50.2	95.3	41.7
GPT-5	94.8	81.9	78.3	60.8	88.3	77.3	90.6	91.0	78.7	57.4	93.8	52.1
Open-Source Models												
Qwen2.5-VL-72B	88.5	72.7	79.1	47.7	76.0	64.7	86.9	88.7	62.0	41.7	75.8	28.0
InternVL3.5-38B	79.0	60.0	71.8	51.9	77.3	67.1	87.8	88.4	72.6	49.5	55.5	31.4
Qwen3-VL-32B	83.0	66.9	77.5	68.0	86.0	78.8	**88.6**	**89.8**	75.7	53.3	37.5	48.8
InternVL3.5-14B	73.2	52.8	55.4	52.0	75.0	65.9	86.1	87.8	73.0	50.2	71.9	39.3
InternVL3-14B	72.3	51.3	54.1	51.4	75.5	65.8	85.8	87.5	73.3	52.2	71.1	40.2
InternVL3.5-8B	66.7	46.0	48.3	55.0	78.0	68.6	85.8	87.3	58.3	40.5	49.2	7.8
InternVL3-8B	63.3	43.8	46.1	54.5	77.4	68.2	85.3	87.6	57.7	38.6	42.2	6.2
JanusCoderV-7B	80.6	65.8	72.7	63.6	70.1	67.5	85.7	87.8	-	-	-	-
Qwen2.5-VL-7B	68.7	42.2	40.1	47.5	73.8	63.3	83.4	87.6	42.7	25.5	21.1	11.7
VinciCoder-7B	91.2	78.3	79.8	77.0	92.0	86.0	88.2	89.1	**84.7**	**60.9**	87.5	56.0
Qwen3-VL-8B	78.3	62.5	67.8	53.0	77.0	67.4	85.5	87.2	47.7	33.0	78.9	41.2
VinciCoder-8B	**91.6**	**78.9**	**80.6**	**77.1**	**94.1**	**87.3**	88.4	89.3	77.3	57.8	**88.3**	**62.6**

Table 3: Main results on multimodal code generation benchmarks. Gray and blue rows represent closed-source baselines and VinciCoder, respectively. The best performance among open-source models is in **bold**. JanusCoderV-7B results for Image2Struct and ChemDraw are omitted due to a lack of task-specific training.

ing the model follows the specified programming language.

4 Experiments

4.1 Implementation Details

We perform SFT on the curated dataset for one epoch, employing Qwen2.5-VL-7B-Instruct (Bai et al., 2025b) and Qwen3-VL-8B-Instruct (Bai et al., 2025a) as the base models. This stage is conducted on 24 H800 GPUs with a global batch size of 96. During the RL phase, we utilize GRPO to fine-tune the SFT model with a global batch size of 256. The hyperparameters ω_v and ω_l are set to 0.9 and 0.1, respectively. For each query, GPRO generate 8 rollouts. For resource allocation, 16 GPUs are dedicated to the policy model while 4 GPUs are reserved for reward scoring, which utilizes DINOv2-L to generate visual embeddings.

4.2 Evaluation Settings

Evaluation Benchmarks. We evaluate VinciCoder against leading closed-source (Comanici et al., 2025; Anthropic, 2025; OpenAI, 2025) and open-source models (Bai et al., 2025b; Zhu et al., 2025; Wang et al., 2025b; Sun et al., 2025) across five domains. Specifically, we utilize ChartMimic (Yang et al., 2024) for charts, Design2Code (Si et al., 2024) for webpages, UniSVG (Li et al., 2025b) for SVG, and Image2Struct (Roberts et al., 2024) for LaTeX generation. For molecules, we evaluate ChemDraw performance on the Cosyn-400k (Yang et al., 2025e) test set, reporting the execution rate and average Tanimoto similarity (Tani. Sim.) of the generated SMILES.

4.3 Main Results

As detailed in Table 3, VinciCoder establishes a new state-of-the-art among open-source solutions, outperforming existing models across the majority of benchmarks. Notably, it maintains a substantial lead over both comparable-scale competitors and the recent unified multimodal model JanusCoderV.

The performance gap between the SFT and final models validates the effectiveness of our two-stage training process. The initial SFT stage alone yields substantial gains, with VinciCoder-SFT significantly surpassing its base model across nearly all benchmarks. This result underscores the high quality of our large-scale SFT dataset. The subsequent ViRL stage further enhances performance across almost all metrics, especially in visual similarity and execution rate. This improvement stems from two mechanisms: RL penalizes invalid outputs with zero rewards to boost execution rates, while the ViRL strategy optimizes perceptual alignment between rendered and target images to improve visual similarity. We visualize reward progressions during the ViRL stage in Figure 4. In addition to generalist models, comparisons against task-specific baselines are provided in the Appendix B.2.

To evaluate the iterative debugging and refinement capabilities of our model, we conduct a two-turn experiment on ChartMimic and UniSVG. We first categorize the first-turn outputs into non-

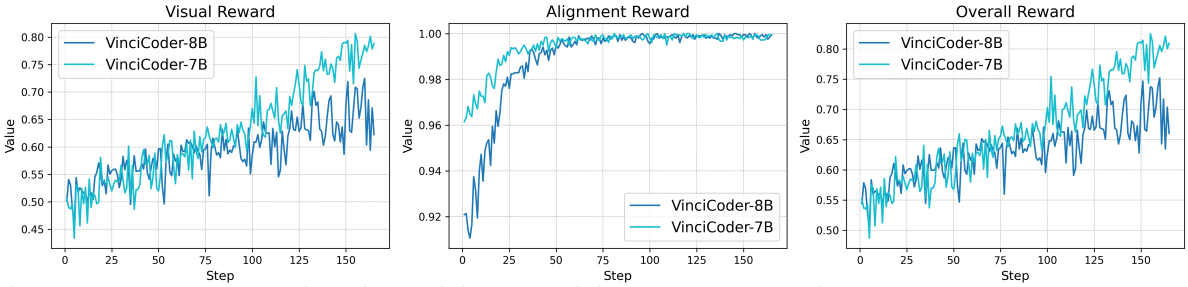


Figure 4: Reward curves during ViRL training. As training progresses, the visual reward shows a steady upward trend, while the alignment reward rapidly converges to its maximum value.

Dataset	Debug (Non-Exec.) Fixed / Total (Fix Rate)	Refine (Exec.) 1st → 2nd (Δ High-L)
ChartMimic	12 / 66 (18.2%)	88.6 → 89.8 (+1.2)
UniSVG	8 / 85 (9.4%)	91.0 → 92.3 (+1.3)

Table 4: Multi-turn refinement results. We report the fix rate and the high-level score improvement for non-executable and executable code respectively.

Model	ChartM.	UniSVG	D2C	I2S	ChemD.
Base	50.3	63.3	85.5	34.1	16.4
+ ViRL	72.5 ↑22.2	70.3 ↑7.0	87.7 ↑2.2	57.6 ↑23.5	2.4 ↓14.0
+ SFT	81.1 ↑30.8	85.9 ↑22.6	86.7 ↑1.2	65.8 ↑31.7	70.4 ↑54.0
VinciCoder	83.1	86.0	88.7	72.8	71.8

Table 5: Ablation study on training stages. We report the average of all metrics for each benchmark, except for UniSVG, which is evaluated using the overall score.

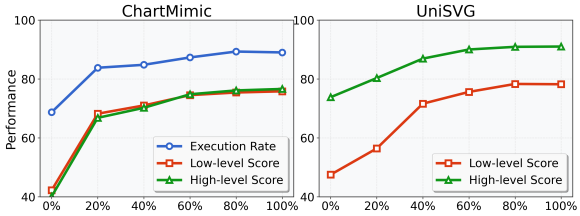


Figure 5: Ablation experiments about model performance under various SFT data scales.

Model	ChartM.	UniSVG	D2C	I2S	ChemD.
Full SFT	81.1	85.9	86.7	65.8	70.4
+ ViRL	83.1 ↑2.0	86.0 ↑0.1	88.7 ↑2.0	72.8 ↑7.0	71.8 ↑1.4
20% SFT	72.9	70.8	86.0	49.0	56.1
+ ViRL	78.8 ↑5.9	80.9 ↑10.1	88.2 ↑2.2	66.2 ↑17.2	58.5 ↑2.4

Table 6: Impact of SFT scale on ViRL performance. Relative improvements from ViRL are most pronounced at a 20% SFT scale, underscoring its ability to enhance less-saturated models.

executable and executable groups to facilitate a granular analysis of different refinement behaviors. Specifically, we assess the repair rate of buggy code within the non-executable group and the degree of visual enhancement for the already executable group. As shown in Table 4, for the non-executable group, our model demonstrates effective debugging skills, fixing 18.2% of the buggy code on ChartMimic to make it runnable. For the executable group, the model focuses on visual refinement, further improving the high-level scores and narrowing the gap with the ground truth. These results show that with the refinement data, our model can effectively correct both functional errors and subtle visual details through multi-turn interactions.

4.4 Ablation Studies

Furthermore, we conduct ablation studies on the VinciCoder-7B, focusing on training strategies, data scaling, and reward function designs.

Training Strategy We first evaluate the individual and combined contributions of the SFT and RL stages. Table 6 confirms that our two-stage SFT-RL strategy achieves the best performance, validating the paradigm of using SFT to establish a strong policy followed by RL optimization. Notably, applying ViRL directly to the base model yields significant gains on ChartMimic and Design2Code, particularly in execution rate and high-level scores. This confirms that visual feedback effectively enhances both code correctness and perceptual fidelity. However, improvements on UniSVG are marginal, and performance even declines on ChemDraw, suggesting that the efficacy of ViRL is constrained by the base model's foundational capacity. However, as we only apply 42k data for RL and 1.6M for SFT, the results demonstrate the efficiency of our proposed ViRL strategy.

Data Scaling. We further investigate the influence of SFT data scale on both immediate performance and the subsequent RL stage. As shown in Figure 5, performance scales consistently with the SFT dataset size before reaching a plateau. This saturation suggests that while our large-scale SFT provides a robust foundation, further gains through supervised learning alone become marginal.

To isolate the contribution of our ViRL strategy, we employ it on the 20% SFT model checkpoint. As detailed in Table 6, ViRL yields a substantial performance leap in this setting. This confirms that

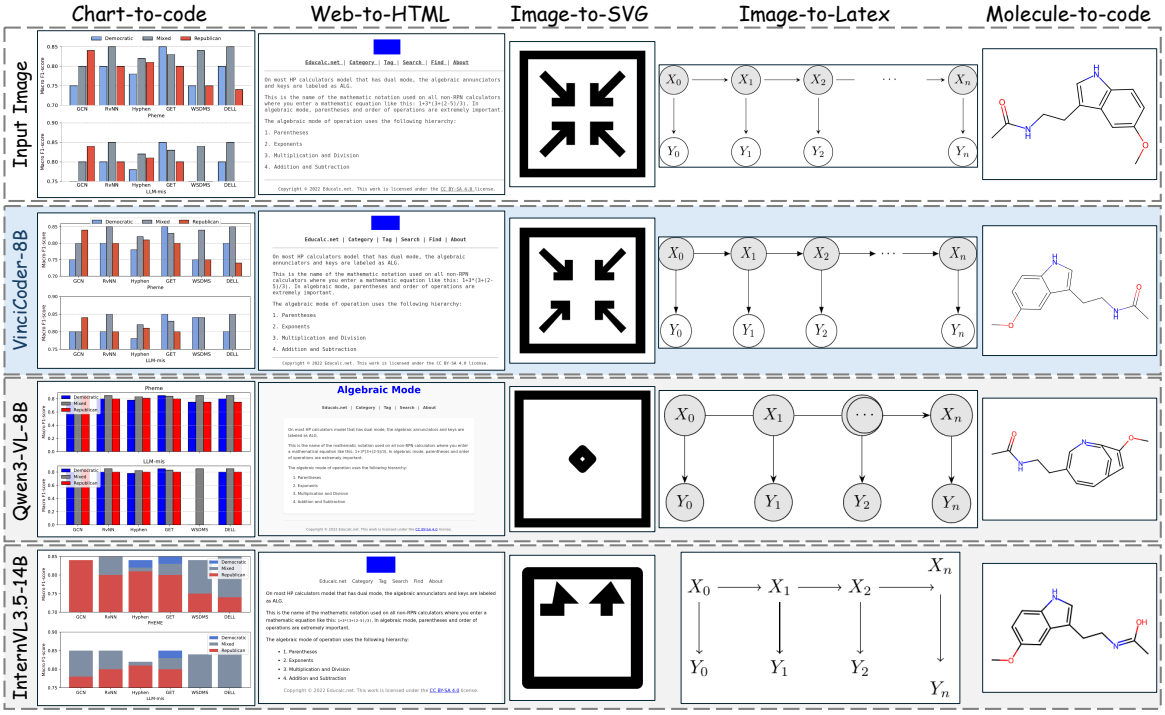


Figure 6: Showcases of images rendered by VinciCoder generated code and other similar scales open-source VLMs.

Ablations	ChartMimic		UniSVG		Design2Code	
	Low-L	High-L	Low-L	High-L	Low-L	High-L
Reward Formulations						
Edit Dist.2	77.1	76.7	75.9	88.5	87.1	87.4
SSIM	75.7	75.3	77.2	89.8	84.5	85.6
InternViT	77.9	78.1	77.1	90.6	86.3	87.9
Coarse-to-fine ViRL						
w/o Coarse	77.8	78.9	76.9	91.8	87.4	88.3
w/o Fine	77.0	78.0	77.1	91.8	86.9	87.6
w/o Align	78.2	79.4	76.2	90.5	87.5	88.3
VinciCoder	78.3	79.8	77.0	92.0	88.2	89.1

Table 7: Ablation studies about the reward function.

the observed gains stem from the ViRL mechanism rather than SFT data scaling. While the 1.6M SFT provides a strong foundation, ViRL shows superior data efficiency in visual alignment, with its effectiveness being most pronounced before the base model reaches its SFT plateau. We provide the comprehensive results in Appendix B.1.

Reward Functions. Last but not least, we conduct ablation studies to validate our coarse-to-fine visual reward design. We evaluate our approach against three baselines: a textual reward based on edit distance, a low-level visual reward based on SSIM, and a semantic-level visual reward based on InternViT-300M (with matched parameters with DINO-L). Furthermore, we isolate the impact of individual components within our coarse-to-fine strategy to verify their specific contributions. As shown in Table 7, our coarse-to-fine reward achieves the most balanced performance. Analysis indicates that self-supervised DINOv2 excels at capturing fine-grained features essential for visual fidelity, whereas image-text aligned ViTs prioritize global semantic alignment. Furthermore, the coarse-to-fine strategy is critical for high-resolution benchmarks like ChartMimic and Design2Code, where omitting either component leads to significant degradation. Conversely, performance on UniSVG remains stable, as its lower resolution allows our method to simplify into a global comparison, rendering the coarse and fine-grained rewards functionally equivalent.

We provide qualitative showcases in Figure 6. Also, additional results and detailed analysis such as failure modes are illustrated in Appendix B.

5 Conclusion

In this paper, we present VinciCoder, a unified multimodal code generation framework optimized via a two-stage SFT-ViRL strategy. Following large-scale SFT on 1.6M multi-domain samples, we introduce ViRL, a reinforcement learning paradigm featuring a coarse-to-fine reward mechanism that aligns code generation with visual perceptual signals. Extensive benchmarks demonstrate that VinciCoder significantly outperforms existing open-source models. Our ablation studies further validate the efficacy of the ViRL paradigm and its ability to achieve high-fidelity visual alignment.

Limitations

Despite improvements in visual quality and code performance, VinciCoder has some limitations. First, visual feedback helps the model organize layout and style, but it cannot teach the model new domain knowledge that is missing from the base policy. For example, if the model does not understand complex chemical structures in ChemDraw, the visual reward might cause it to take shortcuts. In these cases, the model creates basic shapes that look like the target but are logically incorrect. Second, the rendering process and the need to generate many samples for reinforcement learning require a lot of computing power. This makes it difficult to scale the model for very long code files or high resolution images. Finally, the reward system is based on static images. It does not yet consider interactive or moving parts in web based code like HTML and JavaScript.

Ethic Statement

The development of VinciCoder follows ethical guidelines for data usage and model deployment. Our project uses established open source models and libraries to ensure that our research is transparent and easy to reproduce. All training data includes 1.6M SFT samples and 42k RL samples collected from public sources according to their original licenses. We used strict filters to remove personally identifiable information from the training process. Although VinciCoder improves multimodal code generation, it is designed for research and education. Users must know that automated code generation can produce incorrect results. Therefore, we recommend human review for safety critical software. We do not support the use of this model to create malicious code or violate intellectual property rights. By using and supporting the open source community, we hope to encourage responsible progress in vision language modeling.

References

Anthropic. 2025. Introducing claude sonnet 4.5. Web Page.

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. 2025a. Qwen3-vl technical report. *Preprint*, arXiv:2511.21631.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, and 1 others. 2025b. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.

Jonas Belouadi, Eddy Ilg, Margret Keuper, Hideki Tanaka, Masao Utiyama, Raj Dabre, Steffen Eger, and Simone Paolo Ponzetto. 2025. Tikzero: Zero-shot text-guided graphics program synthesis. *arXiv preprint arXiv:2503.11509*.

Jonas Belouadi, Simone Ponzetto, and Steffen Eger. 2024. Detikzify: Synthesizing graphics programs for scientific figures and sketches with tikz. *Advances in Neural Information Processing Systems*, 37:85074–85108.

Lei Chen, Xuanle Zhao, Zhixiong Zeng, Jing Huang, Liming Zheng, Yufeng Zhong, and Lin Ma. 2025a. Breaking the sft plateau: Multimodal structured reinforcement learning for chart-to-code generation. *arXiv preprint arXiv:2508.13587*.

Lei Chen, Xuanle Zhao, Zhixiong Zeng, Jing Huang, Yufeng Zhong, and Lin Ma. 2025b. Chart-r1: Chain-of-thought supervision and reinforcement for advanced chart reasoner. *arXiv preprint arXiv:2507.15509*.

Siqi Chen, Xinyu Dong, Haolei Xu, Xingyu Wu, Fei Tang, Hang Zhang, Yuchen Yan, Linjuan Wu, Wenqi Zhang, Guiyang Hou, and 1 others. 2025c. Svgenius: Benchmarking llms in svg understanding, editing and generation. *arXiv preprint arXiv:2506.03139*.

Yang Chen, Yufan Shen, Wenxuan Huang, Sheng Zhou, Qunshu Lin, Xinyu Cai, Zhi Yu, Jiajun Bu, Botian Shi, and Yu Qiao. 2025d. Learning only with images: Visual reinforcement learning with reasoning, rendering, and visual feedback. *arXiv preprint arXiv:2507.20766*.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

EPAM Systems. 2025. Indigo toolkit. Accessed: 2025-10-18.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, Dongping Chen, Siyuan Wu, Xing Zhou, and 1 others. 2025. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM on Web Conference 2025*, pages 1834–1845.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, and 1 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. 2025. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *arXiv preprint arXiv:2503.06749*.

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. *arXiv preprint arXiv:2508.09945*.

Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R Lyu, and Xiangyu Yue. 2025b. Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. *arXiv preprint arXiv:2507.22827*.

Greg Landrum. 2013. Rdkit documentation. *Release*, 1(1-79):4.

Bozheng Li, Miao Yang, Zhenhan Chen, Jiawang Cao, Mushui Liu, Yi Lu, Yongliang Wu, Bin Zhang, Yangguang Ji, Licheng Tang, and 1 others. 2025a. Opusanimation: Code-based dynamic chart generation. *arXiv preprint arXiv:2510.03341*.

Jinke Li, Jiarui Yu, Chenxing Wei, Hande Dong, Qiang Lin, Liangjing Yang, Zhicai Wang, and Yanbin Hao. 2025b. Unisvg: A unified dataset for vector graphic understanding and generation with multimodal large language models. *arXiv preprint arXiv:2508.07766*.

Ryan Li, Yanzhe Zhang, and Diyi Yang. 2024. Sketch2code: Evaluating vision-language models for interactive web design prototyping. *arXiv preprint arXiv:2410.16232*.

Yuqi Liu, Bohao Peng, Zhisheng Zhong, Zihao Yue, Fanbin Lu, Bei Yu, and Jiaya Jia. 2025a. Seg-zero: Reasoning-chain guided segmentation via cognitive reinforcement. *arXiv preprint arXiv:2503.06520*.

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

OpenAI. 2025. Introducing gpt-5. Web Page.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, and 1 others. 2023. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*.

Josselin S Roberts, Tony Lee, Chi H Wong, Michihiro Yasunaga, Yifan Mai, and Percy Liang. 2024. Image2struct: Benchmarking structure extraction for vision-language models. *Advances in Neural Information Processing Systems*, 37:115058–115097.

Juan A Rodriguez, Abhay Puri, Shubham Agarwal, Issam H Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2025a. Starvector: Generating scalable vector graphics code from images and text. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 16175–16186.

Juan A Rodriguez, Haotian Zhang, Abhay Puri, Aarash Feizi, Rishav Pramanik, Pascal Wichmann, Arnab Mondal, Mohammad Reza Samsami, Rabiul Awal, Perouz Taslakian, and 1 others. 2025b. Rendering-aware reinforcement learning for vector graphics generation. *arXiv preprint arXiv:2505.20793*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Haozhan Shen, Peng Liu, Jingcheng Li, Chunxin Fang, Yibo Ma, Jiajia Liao, Qiaoli Shen, Zilun Zhang, Kangjia Zhao, Qianqian Zhang, and 1 others. 2025. Vlm-r1: A stable and generalizable r1-style large vision-language model. *arXiv preprint arXiv:2504.07615*.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2code: Benchmarking multimodal code generation for automated front-end engineering. *arXiv preprint arXiv:2403.03163*.

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

Huajie Tan, Yuheng Ji, Xiaoshuai Hao, Minglan Lin, Pengwei Wang, Zhongyuan Wang, and Shanghang Zhang. 2025a. Reason-rft: Reinforcement fine-tuning for visual reasoning. *arXiv preprint arXiv:2503.20752*.

Wentao Tan, Qiong Cao, Chao Xue, Yibing Zhan, Changxing Ding, and Xiaodong He. 2025b. Chartmaster: Advancing chart-to-code generation with real-world charts and chart similarity reinforcement learning. *arXiv preprint arXiv:2508.17608*.

Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R Lyu. 2024. Automatically generating ui code from screenshot: A divide-and-conquer-based approach. *arXiv preprint arXiv:2406.16386*.

Ke Wang, Junting Pan, Linda Wei, Aojun Zhou, Weikang Shi, Zimu Lu, Han Xiao, Yunqiao Yang, Houxing Ren, Mingjie Zhan, and 1 others. 2025a. Mathcoder-vl: Bridging vision and code for enhanced multimodal mathematical reasoning. *arXiv preprint arXiv:2505.10557*.

Weiyun Wang, Zhangwei Gao, Lixin Gu, Hengjun Pu, Long Cui, Xingguang Wei, Zhaoyang Liu, Linglin Jing, Shenglong Ye, Jie Shao, and 1 others. 2025b. Internvl3. 5: Advancing open-source multimodal models in versatility, reasoning, and efficiency. *arXiv preprint arXiv:2508.18265*.

Chengyue Wu, Yixiao Ge, Qiushan Guo, Jiahao Wang, Zhixuan Liang, Zeyu Lu, Ying Shan, and Ping Luo. 2024. Plot2code: A comprehensive benchmark for evaluating multi-modal large language models in code generation from scientific plots. *arXiv preprint arXiv:2405.07990*.

Fan Wu, Cuiyun Gao, Shuqing Li, Xin-Cheng Wen, and Qing Liao. 2025. Mllm-based ui2code automation guided by ui layout information. *Proceedings of the ACM on Software Engineering*, 2(ISSTA):1123–1145.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zhiyao Xu, and Michael R Lyu. 2024. Interaction2code: How far are we from automatic interactive webpage generation? *arXiv e-prints*, pages arXiv–2411.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, and 1 others. 2025a. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*.

Cheng Yang, Chufan Shi, Yaxin Liu, Bo Shui, Junjie Wang, Mohan Jing, Linran Xu, Xinyu Zhu, Siheng Li, Yuxiang Zhang, and 1 others. 2024. Chartmimic: Evaluating lmm's cross-modal reasoning capability via chart-to-code generation. *arXiv preprint arXiv:2406.09961*.

Donglu Yang, Liang Zhang, Zihao Yue, Liangyu Chen, Yichen Xu, Wenxuan Wang, and Qin Jin. 2025b. Chartm^3: Benchmarking chart editing with multimodal instructions. *arXiv preprint arXiv:2507.21167*.

Yi Yang, Xiaoxuan He, Hongkun Pan, Xiyan Jiang, Yan Deng, Xingtao Yang, Haoyu Lu, Dacheng Yin, Fengyun Rao, Minfeng Zhu, and 1 others. 2025c. R1-onevision: Advancing generalized multimodal reasoning through cross-modal formalization. *arXiv preprint arXiv:2503.10615*.

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

Yue Yang, Ajay Patel, Matt Deitke, Tanmay Gupta, Luca Weihs, Andrew Head, Mark Yatskar, Chris Callison-Burch, Ranjay Krishna, Aniruddha Kembhavi, and 1 others. 2025e. Scaling text-rich image understanding via code-guided synthetic multimodal data generation. *arXiv preprint arXiv:2502.14846*.

Sukmin Yun, Rusiru Thushara, Mohammad Bhat, Yongxin Wang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, and 1 others. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Advances in neural information processing systems*, 37:112134–112157.

Chenchen Zhang, Yuhang Li, Can Xu, Jiaheng Liu, Ao Liu, Changzhi Zhou, Ken Deng, Dengpeng Wu, Guanhua Huang, Kejiao Li, and 1 others. 2025a. Artifactsbench: Bridging the visual-interactive gap in llm code generation evaluation. *arXiv preprint arXiv:2507.04952*.

Zhihan Zhang, Yixin Cao, and Lizi Liao. 2025b. Boosting chart-to-code generation in mllm via dual preference-guided refinement. *arXiv preprint arXiv:2504.02906*.

Xuanle Zhao, Xuexin Liu, Haoyue Yang, Xianzhen Luo, Fanhu Zeng, Jianling Li, Qi Shi, and Chi Chen. 2025a. Chartedit: How far are mllms from automating chart analysis? evaluating mllms' capability via chart editing. *arXiv preprint arXiv:2505.11935*.

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. *arXiv preprint arXiv:2501.06598*.

Yuqi Zhou, Sunhao Dai, Shuai Wang, Kaiwen Zhou, Qinglin Jia, and Jun Xu. 2025. Gui-g1: Understanding r1-zero-like training for visual grounding in gui agents. *arXiv preprint arXiv:2505.15810*.

Jinguo Zhu, Weiyun Wang, Zhe Chen, Zhaoyang Liu, Shenglong Ye, Lixin Gu, Hao Tian, Yuchen Duan, Weijie Su, Jie Shao, and 1 others. 2025. Internvl3: Exploring advanced training and test-time recipes for open-source multimodal models. *arXiv preprint arXiv:2504.10479*.

A Benchmark Details

Our evaluation methodology for the ChartMimic, UniSVG, and Design2Code benchmarks follows the official implementations provided in their respective GitHub repositories. Regarding Image2Struct, we observed a minor inconsistency between the metrics in the published paper and the official code. To ensure replicability, we follow the implementation in the GitHub repository. We introduce the our calculation of EMS in Section A.1. For the ChemDraw benchmark, we evaluate performance using the execution rate and Tanimoto similarity, two metrics we introduce in Section A.2.

A.1 Earth Mover's Similarity (EMS)

The Earth Mover's Similarity (EMS) is an efficient, patch-based metric derived from the Earth Mover's Distance (EMD). It quantifies similarity through a two-level process that compares the arrangement of image patches at a global level and the pixel distributions within those patches at a local level.

The calculation begins by converting the reference image x and the generated image $\hat{x}$ to grayscale. The most frequent pixel value in the reference image is identified and designated as the background color, v_{bg}, to focus the analysis on foreground content.

Next, both images are partitioned into a grid of K patches, $\{P_0, \ldots, P_{K-1}\}$. A global signature, $S_{global} = \{(w_i, P_i, c_i)\}_{i=0}^{K-1}$, is created for each image. Each element in the signature represents a patch P_i by its content, its normalized center coordinate c_i, and a weight w_i. To prioritize meaningful content, this weight w_i is set significantly higher for patches containing non-background pixels.

The dissimilarity between the images is calculated using a hierarchical cost matrix C_p, where each element $C_p[i, j]$ defines the cost of matching patch P_i from image x to patch P_j from image $\hat{x}$. This cost aggregates two factors: the dissimilarity within the patches and the spatial distance between them:

$$C_p[i, j] = \text{EMD}_{intra}(P_i, P_j) + \lambda \cdot ||c_i - c_j||_1 \quad (6)$$

Here, EMD_{intra} is the classic EMD computed on the pixel values and their local coordinates within each patch. The term $||c_i - c_j||_1$ is the L_1 (Manhattan) distance between the patch centers, scaled by a factor λ:

$$\lambda = \frac{\sqrt{r \times s}}{W + H} \quad (7)$$

Algorithm 1 Tanimoto Similarity Calculation

```
function CALCULATETANIMOTO(pred_smiles, gt_smiles)
    ▷ Step 1: Convert SMILES strings to RDKit molecule.
    pred_mol ← RDKit.MolFromSmiles(pred_smiles)
    gt_mol ← RDKit.MolFromSmiles(gt_smiles)    ▷ Step 2: Handle invalid SMILES. If either is invalid, similarity is 0.
    if pred_mol is null or gt_mol is null then
        return 0.0
    end if
    ▷ Step 3: Generate molecular fingerprints (e.g., Morgan fingerprints).
    pred_fp ← GetMorganFP(pred_mol, radius = 2)
    gt_fp ← GetMorganFP(gt_mol, radius = 2)
    ▷ Step 4: Calculate Tanimoto similarity between the two fingerprints.
    similarity ← TanimotoSimilarity(pred_fp, gt_fp)
    return similarity
end function
```

where (r, s) are the patch dimensions and (W, H) are the image dimensions.

The total dissimilarity score, $\text{EMD}_{block}(x, \hat{x})$, is obtained by solving the transportation problem defined by the cost matrix C_p. This score is then normalized to produce the final EMS value, which lies in the range $[0, 1]$. Normalization is achieved by dividing by the dissimilarity between the reference image x and a constant image x_{const} (either black or white, whichever is more dissimilar):

$$\text{EMS}(x, \hat{x}) = \max\left(0, 1 - \frac{\text{EMD}_{block}(x, \hat{x})}{\text{EMD}_{block}(x, x_{const})}\right) \quad (8)$$

An EMS score of 1 indicates identical images, while a score of 0 indicates maximum dissimilarity.

A.2 ChemDraw Metrics

Performance on the molecule-to-code task is assessed using two key metrics. The first, execution rate, serves as a primary filter for syntactic correctness, evaluating whether the VLM produces chemically valid code and SMILES strings. The second, Tanimoto similarity, measures the chemical fidelity of the valid generations. This metric utilizes the RDKit library to compare the molecular fingerprints of the generated structure against the ground-truth structure. A detailed description of the calculation in Algorithm 1.

The formula for Tanimoto similarity between two fingerprints (bit vectors) A and B is:

$$T(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{c}{a + b - c}$$

where:

Model	ChartMimic-direct-v2			UniSVG-ISVGEN			Design2Code		Image2Struct		ChemDraw	
	Exec.Rate	Low-L	High-L	Low-L	High-L	Score	Low-L	High-L	Ren.Succ.	EMS	Exec.Rate	Tani.Sim.
VinciCoder-7B Variants												
Full SFT	89.0	75.8	78.6	78.2	91.0	85.9	86.5	87.0	77.0	54.6	85.9	54.9
SFT-ViRL	91.2	78.3	79.8	77.0	92.0	86.0	88.2	89.1	84.7	60.9	87.5	56.0
20%SFT	83.8	68.2	66.8	56.4	80.3	70.8	85.6	86.4	57.0	40.9	75.8	36.4
20%SFT-ViRL	88.3	73.4	74.7	69.4	88.5	80.9	87.7	88.7	77.7	54.8	78.9	38.0
ViRL Only	90.2	58.6	68.8	55.2	80.3	70.3	86.8	88.6	64.3	50.9	4.1	0.7
VinciCoder-8B Variants												
Full SFT	88.3	75.6	78.9	78.4	93.7	87.6	86.8	87.9	72.3	50.7	85.9	59.3
SFT-ViRL	91.6	78.9	80.6	77.1	94.1	87.3	88.4	89.3	77.3	57.8	88.3	62.6

Table 8: Comprehensive results for VinciCoder variants across different benchmarks.

- a is the number of bits set in fingerprint A.
- b is the number of bits set in fingerprint B.
- c is the number of bits set in both A and B (the intersection).

B Further Experiment Results

B.1 Full Results of VinciCoder Variants

We provide full evaluation results for VinciCoder variants to show the impact of different training stages and model scales. As shown in Table 8, we report performance for both 7B and 8B versions at the SFT checkpoint (SFT Only) and the checkpoint after SFT and ViRL training (SFT-ViRL), which correspond to VinciCoder-7B and -8B in Table 3. These results demonstrate how the ViRL strategy improves the model performance after the initial SFT stage.

In addition to the main models, we provide the full ablation results of the 7B variant. This includes the 20 percent SFT model (20%SFT) and its corresponding RL version (20%SFT-ViRL) as well as the ViRL only model (ViRL Only) that is trained without any SFT data.

B.2 Comparison with task-specific models

We compare VinciCoder with other domain-specific models, like chart-to-code and image-to-SVG models. The results in Table 9 show that VinciCoder performs comparably to specialized models such as ChartCoder (Zhao et al., 2025b), Chart2Code (Zhang et al., 2025b), MSRL (Chen et al., 2025a) and ChartMaster (Tan et al., 2025b) on the ChartMimic dataset. More domain-specific models are compared in the Appendix. Moreover, its performance on UniSVG exceeds that of fine-tuned specialist models, which validates the effectiveness of our unified framework for multimodal code generation.

Chart-to-code Models	ChartMimic-direct-v2		
	Exec. Rate	Low-Level	High-Level
ChartCoder	91.4	72.5	74.0
Chart2Code	62.1	42.9	33.3
MSRL	96.5	78.6	83.8
ChartMaster	93.8	78.2	85.1
VinciCoder-7B	91.2	78.3	79.8
VinciCoder-8B	91.6	78.9	80.6

Table 9: Comparing with task-specific chart-to-code models.

Image-to-SVG Models	UniSVG-ISVGEN			
	SSIM ↑	LPIPS ↓	CLIP Score↑	Score
LLaVA 1.5 Tuned	65.4	47.9	80.2	71.6
Llama 3.2 Tuned	72.2	37.8	84.3	77.5
Qwen2.5-VL Tuned	72.5	36.8	83.6	77.3
VinciCoder-7B	77.4	23.3	92.0	86.0
VinciCoder-8B	77.4	23.2	94.1	87.3

Table 10: Comparing with task-specific image-to-SVG models. The metrics of finetuned models are from UniSVG (Li et al., 2025b).

B.3 Effectiveness of Refinement Data

We further evaluate the impact of refinement data by comparing models trained with and without its inclusion during the SFT stage. As shown in Table 4, incorporating these tasks yields consistent improvements across nearly all benchmarks. This observation suggests that exposure to refinement data during SFT not only enables iterative correction but also enhances the model's direct generation performance. These results underscore the effectiveness of refinement data in bridging the gap between initial code generation and final visual alignment.

Model	ChartMimic		UniSVG		Design2Code	
	Low-L	High-L	Low-L	High-L	Low-L	High-L
w/o Refine	76.2	76.4	75.6	89.4	86.1	86.6
w/ Refine	75.8	78.6	78.2	91.0	86.5	87.0

Table 11: Ablation studies about the 300k refinement data. We compare models after the SFT stage.

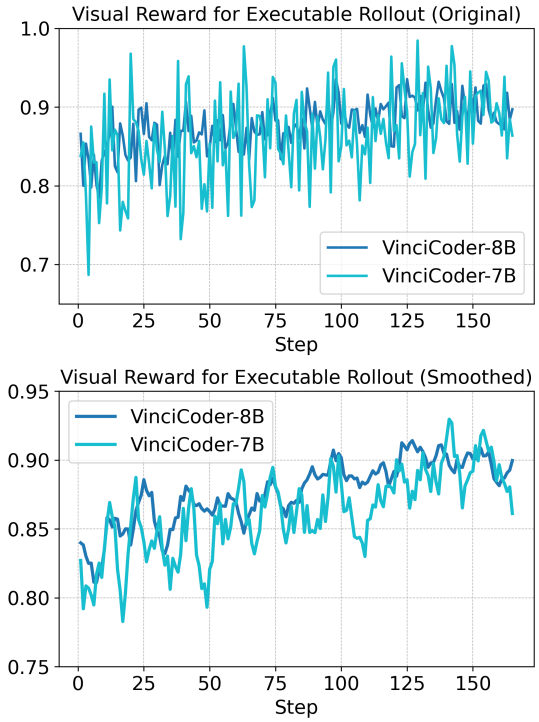


Figure 7: The reward progression of executable rollouts during our ViRL training stage.

B.4 ViRL Results

Figure 4 shows the improvement in visual similarity during RL training. Our reward function assigns a score based on visual similarity only for executable rollouts, while all unexecutable rollouts receive a reward of zero. We want to analyze whether the visual similarity between executable rollouts and input images improves during RL training. We visualize the visual reward of executable rollouts only during the RL training process. The result in Figure 7 denotes that during our training, the visual similarity of executable rollouts improves as well, demonstrating that our proposed ViRL training strategy not only improves the execution rate but also the visual fidelity.

B.5 Fail Cases in ViRL

While ViRL enhances visual fidelity, it faces challenges with unfamiliar code structures. Specifically, the model can generate disorganized code that deviates from the input image and instructions, a behavior particularly evident in the ChemDraw

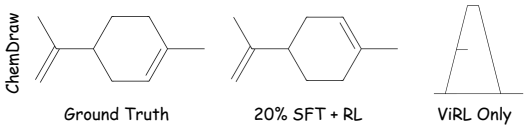


Figure 8: Example of semantically wrong code.

benchmark. As illustrated in Figure 8, the ViRL Only model produces a geometric shape (a simple trapezoid) that bears no semantic relation to the chemical molecular structure in the ground truth. This suggests that without the prior knowledge instilled by SFT, RL alone may oversimplify the complex layout into primitive shapes that are easier to optimize but semantically incorrect. In contrast, the model trained with 20% SFT + RL successfully reconstructs the intricate chemical bonds and rings, underscoring that even a small scale of SFT is essential for enforcing correct syntax and domain-specific code usage.

C Related Works about Multimodal Code Generation Benchmarks

Besides constructing code MLLMs, many multimodal code generation benchmarks have been proposed for evaluation. Previous works generally focus on evaluating direct generation capacity within task-specific domains. For instance, benchmarks such as ChartMimic (Yang et al., 2024) and Plot2Code (Wu et al., 2024) evaluate chart-to-code generation capabilities. Other works, including Design2Code (Si et al., 2024), UniSVG (Li et al., 2025b), and Image2Struct (Roberts et al., 2024), assess the code generation of corresponding visual inputs. Also, some benchmarks evaluate beyond direct generation, including interaction (Xiao et al., 2024; Li et al., 2024) and editing generation (Chen et al., 2025c; Zhao et al., 2025a; Yang et al., 2025b). Recently, with the growing capacities of VLMs, many new and complex benchmarks have been introduced. Artifactsbench (Zhang et al., 2025a) and DCG-Bench (Li et al., 2025a) propose diverse webpage and chart code generation tasks with dynamic visual images and code complexity.

D GRPO Alogrithm

A primary advantage of GRPO is its independence from a separate critic model, a key component in methods like PPO (Schulman et al., 2017). For a given input query x, the algorithm first samples a set of G responses, $\{o_1, o_2, \ldots, o_G\}$, from the current policy π_{old}. Each response receives a reward

R_i and the group-normalized advantage for the i-th response at time step t is:

$$\hat{A}_{i,t} = \frac{R_i - \text{mean}(\{R_j\}_{j=1}^G)}{\text{std}(\{R_j\}_{j=1}^G)}. \tag{9}$$

It then optimizes the new policy π_θ by maximizing the following objective function:

$$\begin{aligned}\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D},\{o_i\}_{i=1}^G\sim\pi_{\theta_{\text{old}}}(\cdot|x)} \\ \left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(r_{i,t}(\theta)\hat{A}_{i,t}\right.\right.\right. \\ \left.\left.\left.\text{clip}\left(r_{i,t}(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_{i,t}\right)\right]\end{aligned} \tag{10}$$

where the probability ratio $r_{i,t}(\theta)$ is defined as:

$$r_{i,t}(\theta) = \frac{\pi_\theta(o_{i,t} \mid x, o_{i,<t})}{\pi_{\theta_{\text{old}}}(o_{i,t} \mid x, o_{i,<t})}. \tag{11}$$

E Refinement Task

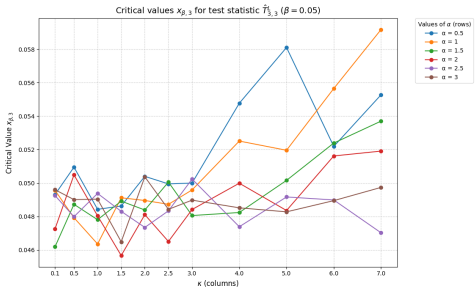
We provide the data format of the refinement task in Figure 9. The incomplete code implementation is generated by our trained, dedicated VLM, and the concrete code implementation is from our curated datasets.

F Evaluation Prompts

In our experiments, all the prompts utilize the official implementations, except for our constructed ChemDraw. For open-source models, we instruct them to generate output SMILES in the format of `'smiles = SMILES string'` and utilize robust parsers to extract the output SMILES for comparison. We visualize the prompt and the input molecule image in Figure 10.

Data Format for Refinement Task

Image (Ground Truth):



User Instruction: You aim to generate a redrawn code of the given chart. Previously, you generated an incomplete code implementation. Let's refine this plotting code. It seems to have inaccuracies and isn't working as expected.
Previous generated code:
Incomplete Code Implementation
Please modify the previous code snippets to ensure the code can render the given image.

Assisstant Instruction:
Concrete Code Implementation

Figure 9: Data showcase of the visual-based refinement task.

ChemDraw Benchmark Prompts

Instruction: Create a Python script that displays a visual representation of the molecule shown in the image. Utilize 'smiles = <SMILES> string' to define the smile and RKiT or Indigo for plot.

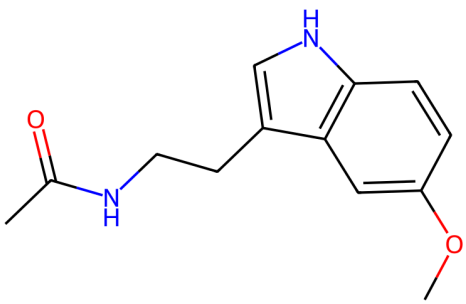


Image:

Figure 10: Prompt for our constructed ChemDraw benchmark.