Unified Visual-to-Code Generation via Symbolic Rewards and Reference-Guided Code Optimization

Anonymous ACL submission

Abstract

Visual-to-Code generation, which transforms scientific plots, vector graphics, and webpages into executable scripts, demands a level of pixel-precise alignment that standard Multimodal Large Language Models (MLLMs) fail to achieve through Supervised Fine-Tuning (SFT) alone. While Reinforcement Learning (RL) offers a theoretical pathway to bridge this gap, its application is hindered by two fundamental obstacles: (1) *Reward Coarseness*, where semantic metrics like CLIP scores fail to penalize fine-grained element deviations, and (2) *Exploration Stagnation*, where the sparse, heterogeneous code search space prevents the policy from bootstrapping valid trajectories. To overcome these limitations, we introduce UniCoder, a unified RL framework that integrates two novel mechanisms. First, we propose **Symbolic Attribute Alignment**, which employs a lightweight auxiliary LLM to parse generated code into discrete visual attributes (e.g., hex colors, coordinate limits), enabling dense, element-wise reward computation. Second, to escape local optima, we devise **Reference-Guided Code Optimization**, a strategy that dynamically injects ground-truth trajectories into low-performing rollout groups, transforming blind exploration into guided policy improvement. Extensive experiments on ChartMimic, UniSVG, Design2Code and ScreenBench benchmarks demonstrate that our 8B-parameter model not only surpasses all open-source baselines but also achieves state-of-the-art performance comparable to proprietary models like GPT-5, establishing a new paradigm for generalized visual-to-code synthesis.

1 Introduction

The capability to translate visual information into executable code stands as a pivotal frontier in Multimodal Large Language Model (MLLM) research. From reverse-engineering scientific charts into Python scripts to converting design mockups into functional webpages, this task bridges the gap between visual intent and digital implementation. Unlike general visual captioning where semantic ambiguity is frequently tolerated, visual-to-code generation demands rigorous precision. The generated code must not only be syntactically executable but also render a visual output that is structurally and stylistically isomorphic to the input.

Recent advancements in MLLMs have been largely driven by large-scale Supervised Fine-Tuning (SFT). While SFT enables models to learn high-level correspondences between image layouts and code structures, it often plateaus in fine-grained alignment. SFT-trained models frequently struggle with precise coordinate reasoning in Scalable Vector Graphics (SVG) or specific attribute extraction in scientific plotting. Consequently, they often default to hallucinated values that fit the training distribution rather than the specific instance.

Reinforcement Learning (RL) offers a promising paradigm to transcend these limitations by optimizing directly for outcome-based feedback. Techniques such as Group Relative Policy Optimization (GRPO) have demonstrated remarkable success in logical reasoning and code generation. However, adapting RL to the visual-to-code domain introduces two fundamental challenges: **(1) The Reward Alignment Problem:** Constructing a reward signal that is both dense and accurate is notoriously difficult. Pixel-level metrics are overly sensitive to high-frequency rendering artifacts where a minor padding shift can penalize a structurally correct solution. Conversely, semantic metrics are often too coarse to penalize hallucinations in text labels. **(2) Exploration Stagnation:** The search space for executable code is sparse and brittle. A model facing a complex visual input often suffers from a "cold start" problem where it fails to generate any executably valid trajectories during early training. This leads to gradient variance collapse and

*Equal contribution.
†Corresponding author.

learning stagnation.

To address these challenges, we propose a unified reinforcement learning framework designed to master diverse visual-to-code tasks simultaneously. We introduce **Symbolic Attribute Alignment**, a mechanism that leverages a lightweight and auxiliary LLM to parse generated code and extract discrete visual attributes such as fill colors and axis limits. This enables the calculation of precise and element-level rewards that decouple structural correctness from rendering noise. Furthermore, we propose **Reference-Guided Code Optimization** to mitigate exploration stagnation. This strategy dynamically injects ground-truth reference code into the policy's rollout buffer during low-performance episodes. This effectively bootstraps the exploration process by transforming the optimization landscape from sparse blind search to guided imitation.

Our main contributions are summarized as follows:

- We identify the critical bottlenecks of reward coarseness and exploration stagnation when applying RL to multimodal code generation.
- We propose UniCoder, a novel framework integrating an LLM-based Attribute Extractor for fine-grained reward modeling and a Reference Injection strategy for stable policy exploration.
- We demonstrate that our 8B-parameter model achieves state-of-the-art performance among open-source models on multiple diverse benchmarks including ChartMimic, UniSVG, Design2Code and ScreenBench. Our method significantly closes the gap with proprietary models like GPT-5.

2 Related Work

2.1 Multimodal Large Language Models for Visual-to-Code Generation

The emergence of Multimodal Large Language Models (MLLMs) has significantly advanced the field of joint visual-text reasoning. Early explorations such as LLaVA(Liu et al., 2023b) laid the groundwork for aligning visual and textual representations by introducing a simple linear projection layer to map CLIP visual features into the LLaMA word embedding space. Qwen-VL(Bai et al., 2023) and DeepSeek-VL(Lu et al., 2024) were fine-tuned to further enhance high-resolution processing and OCR capabilities essential for reading code-related visual elements. Contemporary foundation models like GPT-5.1 and Gemini 3 Pro have dramatically expanded the frontiers of multimodal understanding, excelling in tasks that require generating high-fidelity code from complex imagery. Beyond general-purpose reasoning, increasing attention has been dedicated to specialized domains. **Python Plots**: Early works like DePlot(Liu et al., 2023a) converted charts into textual tables for downstream reasoning. More recently, Plot2Code(Wu et al., 2024) and ChartMimic(Yang et al., 2025) introduced benchmarks requiring models to generate executable Python code (e.g., Matplotlib) that strictly reproduces the visual appearance of reference charts. **SVG**: DeepSVG pioneered deep generative models for vector icons using hierarchical transformers to predict path commands. Building on this, IconShop(Wu et al., 2023) and StarVector(Rodriguez et al., 2025) explored text-guided SVG generation. For large-scale icon-to-SVG translation evaluation, UniSVG(Li et al., 2025) established a comprehensive framework highlighting the difficulty MLLMs face in maintaining topological correctness and geometric precision without explicit reinforcement. **Webpage**: WebSight(Bhathal and Gupta, 2025) managed to utilize millions of synthetic screenshot-code pairs for fine-tuning. Design2Code(Si et al., 2025) provided a rigorous benchmark for real-world frontend generation, demonstrating that while SFT models can capture high-level layouts, they often struggle with fine-grained alignment and complex DOM tree nesting. Recent attempts such as LayoutCoder integrated multi-agent framework to further enhance webpage fidelity for UI-to-Code generation.

2.2 Reinforcement Learning for Multimodal Alignment

Group Relative Policy Optimization (GRPO) (Shao et al., 2024) has rapidly emerged as a paradigm-shifting approach for post-training alignment, demonstrating exceptional efficacy in unlocking complex reasoning capabilities, particularly for code generation tasks. Recent studies further show that group-based policy optimization can benefit from external gold references when sampled responses provide weak contrastive signals (Wu and Liu, 2025), suggesting the importance of reliable guidance in low-reward regimes. However, applying such advanced optimization to domain-specific visual synthesis remains constrained by the quality of reward signals. Existing approaches in this field

predominantly rely on coarse-grained objectives: Leading approaches for SVG generation like VectorFusion(Jain et al., 2022) and StarVector(Rodriguez et al., 2025) predominantly rely on basic semantic distillation (e.g., CLIP Score) or coordinate reconstruction . ScreenCoder(Jiang et al., 2025) employed a pixel-level Mean Squared Error (MSE) as the primary reward signal during RL stage for Webpage-to-Code generation. These heuristic metrics, however, fail to capture element-level nuances due to the non-linear mapping between symbolic code and visual output, where subtle deviations in the code structure can trigger cascading layout failures.

3 Motivation

Developing a generalist agent for visual-to-code generation requires us to overcome two fundamental bottlenecks. These challenges stem from the unique, non-linear nature of translating visual intent into executable logic. We argue that standard reinforcement learning paradigms are ill-suited for this domain without specific structural adaptations.

3.1 The Reward Alignment Paradox

The first obstacle lies in the difficulty of defining a reward signal that is both chemically dense and semantically accurate. Unlike general text generation where semantic equivalence is broad, visual-to-code generation requires a rigorous isomorphism between the input image and the rendered output. We observe that conventional reward mechanisms fail to capture this relationship due to two distinct failure modes.

Semantic Coarseness of Visual Embeddings. Metrics derived from pre-trained vision-language models, such as the CLIP Score, are optimized for global semantic alignment rather than fine-grained structural precision. As illustrated in Figure 1(a) and (b), we observe that these metrics exhibit a dangerous tolerance for "semantic hallucinations." A generated chart may receive a high similarity score because it successfully mimics the color scheme and general layout of the ground truth. However, it may simultaneously fail to render the correct data points or text labels. This creates a reward signal that encourages the model to cheat. The policy learns to prioritize aesthetic style over data fidelity.

Brittleness of Pixel-Level Objectives. Conversely, attempting to enforce precision via pixel-level metrics like Mean Squared Error (MSE) introduces the opposite problem. The rendering process of code is highly non-linear and sensitive to high-frequency noise. A semantically irrelevant change, such as a minor shift in margin padding or a difference in font anti-aliasing, can result in a catastrophic pixel loss. This noisy feedback confuses the policy. It penalizes structurally correct code for rendering artifacts that are invisible to the human eye. This dichotomy motivates our design of **Symbolic Attribute Alignment**. We posit that the only way to accurately evaluate code is to decouple the *symbolic intent* (what elements should exist) from the *rendering realization* (how pixels are drawn).

3.2 Exploration Stagnation in Heterogeneous Spaces

The second major bottleneck is the difficulty of effective exploration. While Group Relative Policy Optimization (GRPO) has shown success in math and logic tasks, applying it to multimodal code generation reveals a severe "cold start" problem.

The Sparse Reward Landscape. The search space for executable code is notoriously brittle. Unlike natural language where a slightly incorrect sentence is still readable, a slightly incorrect code block often results in a syntax error or a blank render. This creates an extremely sparse reward landscape. During the initial stages of training, the model frequently fails to generate any executable trajectories. Consequently, the gradient variance collapses. As shown in Figure 1(c), this leads to prolonged periods of reward stagnation where the model learns nothing because it receives no positive feedback to reinforce.

Cognitive Burden of Heterogeneity. This stagnation is exacerbated by the requirement to master multiple coding paradigms simultaneously. A unified model must navigate vastly different reasoning modalities. These include the logic-driven imperative programming of Python, the geometry-driven coordinate systems of SVG, and the structure-driven DOM nesting of HTML. We observe that standard random exploration struggles to toggle between these modalities effectively. The model often gets stuck in local optima where it outputs valid code for the wrong domain or hallucinates syntax from one language into another. This observation serves as the foundation for our **Reference-Guided Optimization**. We argue that to traverse such a disjoint

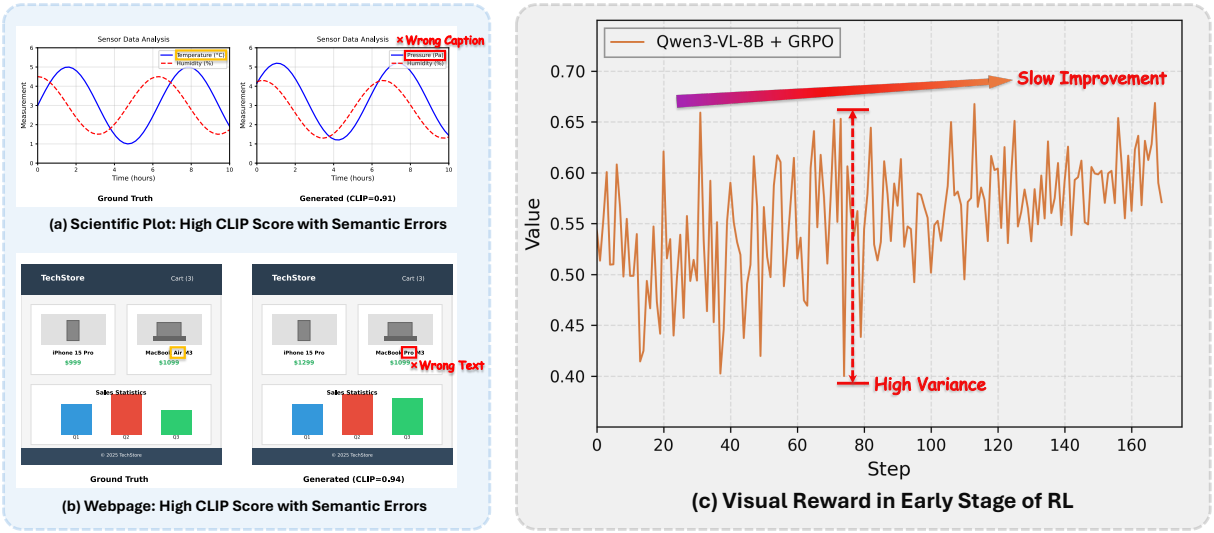

Figure 1: Limitations of CLIP-based visual rewards for Visual-to-Code generation. (a-b) **Semantic Error Cases**: Generated outputs receive high CLIP scores (0.91 and 0.94) despite containing critical errors, such as wrong axis labels and shifted data points in scientific plots, incorrect prices and product names in webpages. These examples demonstrate that CLIP captures global visual similarity but fails to enforce fine-grained element-level alignment. (c) **Exploration Stagnation**: Training with vanilla GRPO shows reward stagnation and high variance, indicating the model struggles to explore effectively across heterogeneous coding paradigms (Python/SVG/HTML).

and punishing search space, the optimization process must be transformed from a blind search into a guided discovery.

4 Method

We propose a unified reinforcement learning framework designed to overcome the reward sparsity and exploration stagnation inherent to multimodal code generation. Our approach, Reference-Guided Group Relative Policy Optimization (RG-GRPO), adapts the standard GRPO paradigm by introducing two critical modifications: (1) a *Symbolic Attribute Alignment* mechanism that densifies the reward signal, and (2) a *Reference Injection* strategy that dynamically alters the exploration distribution to bootstrap learning. The overall framework is illustrated in Figure 2.

4.1 Problem Formulation

We consider the task of visual-to-code generation as a conditional generation problem. Given a visual input x (e.g., a chart image, SVG, or webpage screenshot) and a textual instruction I, the goal is to generate executable code $y = (y_1, ..., y_T)$ that renders into an image $\mathcal{R}(y)$ visually isomorphic to x. Standard Supervised Fine-Tuning (SFT) optimizes the likelihood $P_\theta(y|x, I)$. However, SFT lacks the feedback mechanism to penalize visually minor but symbolically fatal errors. We therefore transition to Reinforcement Learning, aiming to maximize an expected reward:

$$J(\theta) = \mathbb{E}_{y \sim \pi_\theta(\cdot|x,I)}[R(y, x)] \quad (1)$$

where R is our proposed multi-faceted reward function.

4.2 Multi-Faceted Reward Modeling

To address the "Reward Alignment Paradox" (Section 3), we construct a composite reward function that decouples executability, semantic fidelity, and symbolic precision.

Visual and Execution Feedback. We employ two foundational signals:

- **Execution Constraints (R_{exec}):** A binary indicator where $R_{exec}(y) = 1$ if $\mathcal{R}(y) \neq \emptyset$ (i.e., the code compiles and renders), and 0 otherwise. This acts as a hard gatekeeper for the reward signal.
- **Semantic Similarity (R_{vis}):** For valid executions, we compute the cosine similarity between the CLIP embeddings of the rendered image $\mathcal{R}(y)$ and the ground truth x: $R_{vis} = \text{CLIP}(\mathcal{R}(y), x)$.

Symbolic Attribute Alignment. Standard visual metrics fail to capture fine-grained details (e.g., exact hex colors or axis ranges). To bridge this gap,

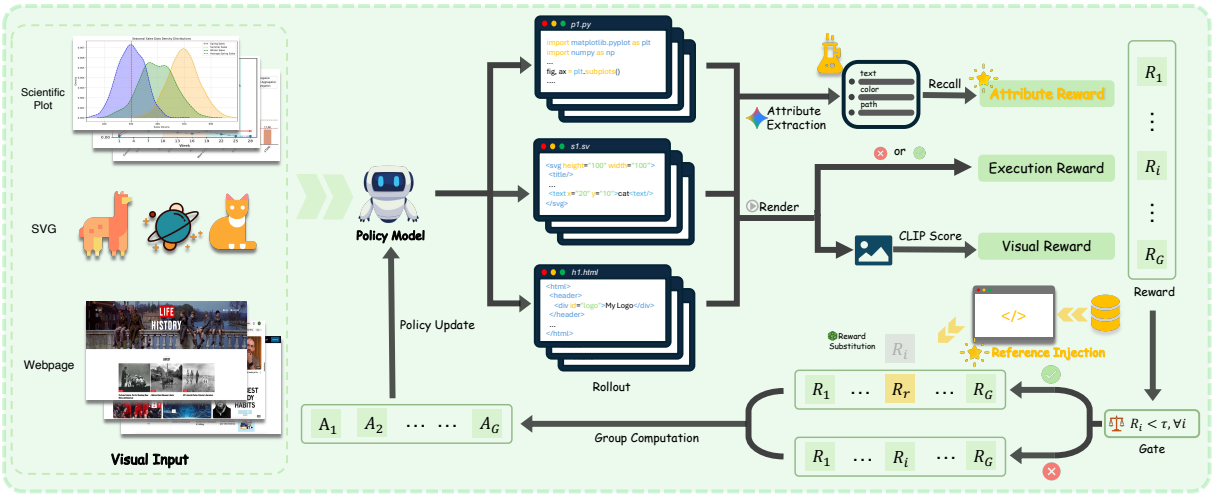

Figure 2: **Overview of the proposed framework.** The model processes diverse visual inputs (Scientific Plots, SVGs, Webpages) to generate corresponding executable code. To ensure high-quality generation, we employ a multi-faceted reward mechanism consisting of: (1) **Attribute Reward** for fine-grained symbolic alignment, (2) **Execution Reward** to verify code compilability, and (3) **Visual Reward** (e.g., CLIP score) for semantic consistency. These rewards guide the *Policy Update* via a reference-guided group computation strategy.

we introduce an **Attribute Extraction Module**, $\mathcal{E}_\phi$, parameterized by a lightweight auxiliary LLM (e.g., Qwen-2.5-3B). For a generated code sequence y, the module extracts a discrete set of symbolic attributes $A_{gen} = \mathcal{E}_\phi(y)$. This set includes domain-specific elements: $\{\text{color_codes}, \text{text_labels}, \text{axis_limits}\}$ for plots, or $\{\text{DOM_tags}, \text{styles}\}$ for webpages.

We define the Attribute Reward R_{attr} as the *Soft Recall* of these generated attributes against the ground-truth attributes $A_{gt} = \mathcal{E}_\phi(y_{gt})$:

$$R_{attr}(y, y_{gt}) = \frac{\sum_{a \in A_{gen}} \mathbb{I}(a \in A_{gt})}{|A_{gt}| + \epsilon} \tag{2}$$

Unlike exact matching, this formulation allows partial credit (e.g., getting the colors right but the labels wrong), providing a dense gradient signal even when the global structure is imperfect.

The final reward is a weighted sum: $R_{total} = w_1 R_{exec} + w_2 R_{vis} + w_3 R_{attr}$.

4.3 Reference-Guided Exploration Strategy

A standard GRPO update samples a group of outputs $G = \{y_1, ..., y_G\} \sim \pi_\theta(x)$ and computes advantages relative to the group mean. However, in the visual-to-code domain, the "cold start" problem often results in a group where *all* candidates fail ($R_{total} \approx 0$). In this scenario, the group mean is zero, the variance is zero, and the gradient becomes uninformative.

To mitigate this, we propose a Reference Injection strategy that modifies the sampling distribution. During the rollout phase, we monitor the quality of the generated group. If the maximum reward in the group falls below a threshold τ, we force-replace the lowest-performing candidate y_{worst} with the ground-truth reference code y_{ref}.

$$G' = \begin{cases} (G \setminus \{y_w\}) \cup \{y_{\text{ref}}\}, & \\ \quad \text{if } \max_{y \in G} R(y) < \tau, & \\ G, & \\ \quad \text{otherwise.} & \end{cases} \tag{3}$$

This injection transforms the learning process from pure exploration to *dynamic imitation*. The reference code y_{ref} introduces a high-reward trajectory into the group, ensuring that the group mean $\bar{R}$ is non-trivial.

4.4 Reference-Aware Policy Optimization

Our optimization objective is a modification of the GRPO loss. We utilize the injected group G' to compute the advantage estimates. For each sample $y_i \in G'$, the advantage is standardized as:

$$A_i = \frac{R(y_i) - \frac{1}{|G'|}\sum_{y_j \in G'} R(y_j)}{\text{std}(\{R(y_j)\}_{j=1}^{|G'|}) + \epsilon} \tag{4}$$

The Anchor Effect. The inclusion of y_{ref} (where $R(y_{ref}) \gg 0$) critically alters the optimization dynamics. It acts as a **performance anchor**, significantly raising the group mean. Consequently, suboptimal generated samples y_{gen} receive a strongly

negative advantage $A_{gen} \ll 0$. The objective function is then:

$$\mathcal{L}_{\text{RG-GRPO}} = -\frac{1}{|G'|}\sum_{y_i \in G'}\Bigg[\min\left(\frac{\pi_\theta(y_i)}{\pi_{\text{old}}(y_i)}A_i, \text{clip}(\dots)A_i\right) - \beta \mathbb{D}_{KL}\Bigg] \quad (5)$$

where $\rho_i = \pi_\theta(y_i)/\pi_{\text{old}}(y_i)$. By "anchoring" the advantage calculation with the reference, the optimizer is not only encouraged to increase the probability of the injected reference (imitation) but is also explicitly discouraged from repeating the failed exploration attempts (suppression of errors), leading to faster convergence and reduced variance compared to vanilla GRPO.

5 Experiments

5.1 Experimental Setup

Datasets and Benchmarks. We conduct comprehensive experiments to evaluate the unified multimodal code generation capabilities of our framework across three distinct domains: scientific Python plotting, scalable vector graphics (SVG) creation, and webpage code generation.

- **Training Data:** To foster robust cross-task generalization, we curate a unified instruction tuning dataset comprising **MSRL** (Chen et al., 2025) for Python plot generation, **SVG-Stack** (Rodriguez et al., 2025) for SVG creation, and **Web2Code** (Yun et al., 2024) for hierarchical webpage structures.
- **Evaluation Benchmarks:** We assess performance on authoritative domain-specific benchmarks: **ChartMimic** (Yang et al., 2025) for chart-to-code generation (under the direct generation setting), **UniSVG** (Li et al., 2025) for SVG translation, and **Design2Code** (Si et al., 2025) and **Screen-10K** (Jiang et al., 2025)for real-world webpage-to-HTML synthesis. For ChartMimic and UniSVG, we report the official low-level and high-level metrics used by each benchmark. For Design2Code and ScreenBench, we report block, text, position, color, and CLIP-based similarity metrics.

Implementation Details. **Model Architecture:** We adopt `Qwen3VL-8B-Instruct` as our backbone policy model, selected for its strong multimodal reasoning capabilities among open-source MLLMs. For the Reference-Guided reward calculation, we utilize the lightweight `Qwen2.5-3b-Instruct` specifically as the Attribute Extractor. This smaller model is selected to strike an optimal balance between computational efficiency and the structural understanding required for feature extraction.
Training Configuration: Our Visual Reinforcement Learning stage is built upon the Group Relative Policy Optimization (GRPO) algorithm. We set the global batch size to 128 and the number of rollouts per prompt to $G = 16$ to estimate the group baseline. During inference and rollout generation, we use a temperature of 0.7 with a maximum response length of 4096 tokens.

5.2 Main Results

We evaluate our proposed method against state-of-the-art open-source MLLMs and proprietary closed-source models, as shown in Table 1.

Superiority over Open-Source Baselines. As shown in the table, our model, Qwen3-VL-8B (Ours), achieves state-of-the-art performance among all open-source models of similar scale. Compared to the strong baseline Qwen3-VL-8B, our method delivers substantial and consistent improvements across all metrics. This validates the effectiveness of our reinforcement learning framework in enhancing the visual-to-code generation capabilities of MLLMs. Notably, our model outperforms competing other 7B/8B class models (e.g., LLaVA-v1.6, InternVL3.5) by a significant margin, establishing a new benchmark for open-source performance in this domain.

Comparison with Closed-Source Models. We further extend our evaluation to leading proprietary models, including Gemini 3 Pro, GPT 5 and Claude 4.5 Opus. Despite the substantial disparity in parameter scale and training resources, `Qwen3-VL-8B (Ours)` demonstrates remarkably competitive performance. As detailed in Table 1, our method significantly narrows the gap with state-of-the-art commercial systems on the Design2Code benchmark and even surpasses larger proprietary models on specific visual alignment tasks. This result indicates that our visual reinforcement learning framework effectively unlocks the potential of compact models, enabling them to rival large-scale systems in specialized multimodal code generation tasks without requiring massive computational overhead.

Model	ChartMimic_direct_v2			UniSVG-ISVGEN			Design2Code					ScreenBench				
	Exec.R	Low-L	High-L	Low-L	High-L	Score	Block	Text	Pos.	Color	CLIP	Block	Text	Pos.	Color	CLIP
Closed-Source Models																
Gemini-3-Pro	83.2	71.5	70.8	71.5	84.8	80.4	0.852	0.971	0.905	0.892	0.918	0.752	0.851	0.735	0.712	0.798
Claude-4.5-Opus	85.8	73.2	72.5	69.5	85.5	81.2	0.858	0.969	0.912	0.885	0.915	0.761	0.849	0.742	0.725	0.805
GPT-5	84.5	75.6	73.9	70.8	88.8	81.9	0.855	0.973	0.898	0.899	0.920	0.758	0.854	0.728	0.718	0.802
Open-Source Models																
LLaVA-v1.6-7B	60.2	20.4	20.7	43.2	74.6	62.0	0.736	0.910	0.729	0.816	0.802	0.635	0.830	0.544	0.592	0.727
InternVL3.5-8B	66.5	42.4	47.1	54.1	76.9	67.8	0.805	0.942	0.798	0.825	0.876	0.710	0.822	0.598	0.620	0.755
Deepseek-VL-7B	41.1	18.8	20.5	47.2	73.5	63.0	0.718	0.824	0.702	0.720	0.843	0.680	0.773	0.570	0.614	0.732
Qwen2.5-VL-7B	65.5	39.9	40.7	56.4	73.8	63.3	0.822	0.951	0.815	0.831	0.893	0.723	0.828	0.613	0.632	0.762
Qwen3-VL-8B	77.3	63.4	66.8	60.0	78.1	70.7	0.841	0.968	0.852	0.855	0.905	0.741	0.843	0.652	0.661	0.784
Ours	**86.4**	**76.1**	**74.4**	**72.2**	**89.1**	**82.3**	**0.865**	**0.975**	**0.925**	**0.908**	**0.922**	**0.768**	**0.857**	**0.755**	**0.734**	**0.812**

Table 1: Comparison of different models on ChartMimic, UniSVG, Design2Code, and ScreenBench benchmarks. Best results are highlighted in **bold** and second best are underlined.

Configuration	ChartMimic			UniSVG			Design2Code					ScreenBench				
	Exec.R	Low-L	High-L	Low-L	High-L	Score	Block	Text	Pos.	Color	CLIP	Block	Text	Pos.	Color	CLIP
Qwen3-VL-8B (Base)	77.3	63.4	66.8	60.0	78.1	70.7	0.841	0.968	0.852	0.855	0.905	0.741	0.843	0.652	0.661	0.784
Ablation Variants																
w/o Reference Injection	79.6	69.8	68.5	68.8	85.6	78.9	0.848	0.970	0.895	0.885	0.910	0.748	0.846	0.710	0.705	0.792
w/o Attribute Reward	83.2	70.4	71.3	67.5	81.7	76.2	0.859	0.972	0.875	0.870	0.915	0.762	0.852	0.685	0.680	0.801
Ours (Full Model)	**86.4**	**76.1**	**74.4**	**72.2**	**89.1**	**82.3**	**0.865**	**0.975**	**0.925**	**0.908**	**0.922**	**0.768**	**0.857**	**0.755**	**0.734**	**0.812**

Table 2: Ablation study on ChartMimic, UniSVG, Design2Code, and ScreenBench. The results demonstrate that the Attribute Reward is particularly critical for fine-grained metrics (Position, Color), while Reference Injection significantly stabilizes layout performance (Block).

Training Analysis. To understand the optimization dynamics of our framework, we visualize the evolution of the reward components during training in Figure 3.

- **Syntactic Correctness:** The *Execution Rate* (Figure 3a) increases rapidly in the early stages, indicating that the policy quickly learns to generate syntactically valid code that renders successfully.
- **Visual & Structural Alignment:** Both *Visual Reward* and *Attribute Reward* (Figure 3b, c) show consistent growth, demonstrating that the model progressively learns to align the generated output with the visual constraints of the reference image.
- **Convergence:** The *Overall Reward* (Figure 3d) exhibits a stable upward trend, confirming that our Reference-Guided GRPO algorithm effectively optimizes the multi-objective objective without suffering from reward collapse or instability.

5.3 Ablation Study

To verify the effectiveness of each component in our proposed framework, we conduct ablation studies on the `Qwen3-VL-8B` backbone. We dissect the contributions of our two core innovations: the *Reference Injection Strategy* and the *Element-level Attribute Reward*.

Experimental Configurations. We evaluate four distinct settings as shown in Table 2. First, the Base Model serves as the vanilla Qwen3-VL-8B baseline without our RL framework. Second, w/o Reference Injection tests our method excluding the injection mechanism (relying on standard prompting). Third, w/o Attribute Reward uses only the global CLIP score as the reward signal, removing the fine-grained attribute extractor. Finally, Ours (Full Model) represents the complete framework combining Reference Injection with the Attribute Reward.

Impact of Attribute Reward (Fine-Grained Alignment). The ablation of the Attribute Reward reveals its specific role in refining local visual details. Comparing Ours with w/o Attribute Reward, we observe that while global metrics like CLIP scores remain relatively high, fine-grained structural metrics suffer significantly without element-level supervision. On Design2Code, the Position score drops from 0.925 to 0.875, and Color accuracy falls from 0.908 to 0.870. Similarly, on ScreenBench, removing the attribute reward causes a sharp decline in Position ($0.755 \rightarrow 0.685$) and Color ($0.734 \rightarrow 0.680$). These results confirm

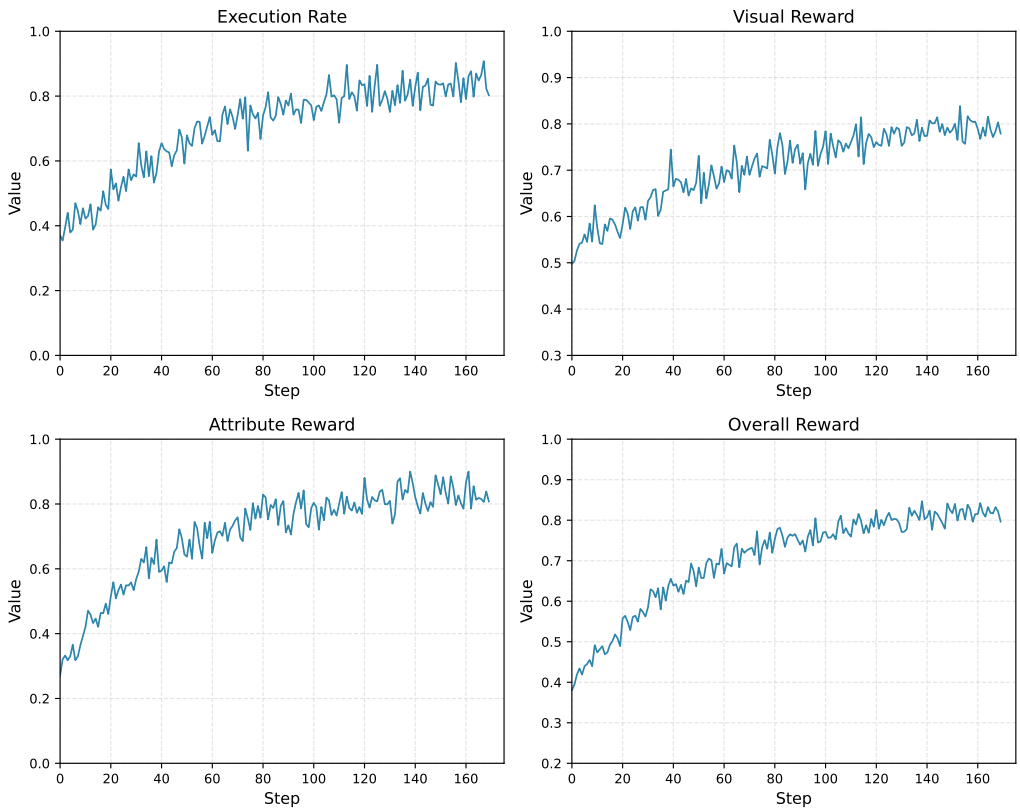

Figure 3: Training reward curves of our method. We report four metrics: (a) **Execution Rate** measures the fraction of generated code that compiles and renders successfully; (b) **Visual Reward** captures the CLIP similarity between rendered outputs and ground truth images; (c) **Attribute Reward** quantifies the element-level alignment via our proposed attribute extraction mechanism; (d) **Overall Reward** is the weighted combination of the three components ($R_{total} = w_1 R_{exec} + w_2 R_{vis} + w_3 R_{attr}$). All metrics show consistent improvement throughout training, demonstrating the effectiveness of our Reference-Guided GRPO framework.

that while a global reward guides general semantic alignment, our proposed Attribute Extractor is essential for enforcing precise spatial layout and color fidelity.

Impact of Reference Injection (Stability). The *Reference Injection* strategy proves critical for stabilizing the generation process and ensuring layout correctness. Regarding execution stability on ChartMimic, removing the injection mechanism (w/o Reference Injection) leads to a significant drop in Execution Rate from 86.4% to 79.6%, indicating that the model struggles to generate valid plotting code without the structural hints provided by the injection. Furthermore, regarding layout consistency on Design2Code, the Block metric (measuring layout bounding box alignment) decreases from 0.865 to 0.848 without injection. This suggests that the Reference Injection strategy effectively constrains the infinite action space, guiding the model toward structurally valid and visually consistent outputs.

5.4 Qualitative Result

Additional qualitative comparisons are provided in Appendix A, illustrating typical improvements in color fidelity, layout structure, and element completeness.

6 Conclusion

We presented UniCoder, a unified reinforcement learning framework for visual-to-code generation. UniCoder addresses reward coarseness and exploration stagnation through Symbolic Attribute Alignment and Reference-Guided Code Optimization, which provide dense element-level feedback and stabilize policy exploration with reference trajectories. Experiments across scientific plotting, SVG generation, and webpage synthesis show that our 8B model substantially improves over open-source baselines and narrows the gap with leading proprietary systems, indicating that fine-grained symbolic rewards and reference-guided exploration are effective for improving precise visual-to-code synthesis.

Limitations

First, our evaluation focuses on three representative visual-to-code settings: scientific plotting, SVG generation, and webpage synthesis. These settings cover diverse rendering targets and programming structures, but they are still centered on static visual reconstruction. The current study does not address interactive interfaces, animations, mobile applications, or programs whose correctness depends on complex runtime behavior beyond the rendered static output. Second, our symbolic reward design relies on task-relevant visual attributes that can be extracted and compared from code, such as colors, text labels, coordinate ranges, and layout-related styles. While these attributes are well aligned with the benchmarks studied in this paper, extending the same framework to substantially different visual domains may require adapting the attribute schema and extraction prompts to capture new domain-specific properties.

References

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A versatile vision-language model for understanding, localization, text reading, and beyond. *Preprint*, arXiv:2308.12966.

Tanvir Bhathal and Asanshay Gupta. 2025. Websight: A vision-first architecture for robust web agents. *Preprint*, arXiv:2508.16987.

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

Ajay Jain, Amber Xie, and Pieter Abbeel. 2022. Vectorfusion: Text-to-svg by abstracting pixel-based diffusion models. *Preprint*, arXiv:2211.11319.

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

Jinke Li, Jiarui Yu, Chenxing Wei, Hande Dong, Qiang Lin, Liangjing Yang, Zhicai Wang, and Yanbin Hao. 2025. Unisvg: A unified dataset for vector graphic understanding and generation with multimodal large language models. In *Proceedings of the 33rd ACM International Conference on Multimedia*, MM '25, page 13156–13163. ACM.

Fangyu Liu, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023a. Deplot: One-shot visual language reasoning by plot-to-table translation. *Preprint*, arXiv:2212.10505.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning. *Preprint*, arXiv:2304.08485.

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. *Preprint*, arXiv:2403.05525.

Juan A. Rodriguez, Abhay Puri, Shubham Agarwal, Issam H. Laradji, Pau Rodriguez, Sai Rajeswar, David Vazquez, Christopher Pal, and Marco Pedersoli. 2025. Starvector: Generating scalable vector graphics code from images and text. *Preprint*, arXiv:2312.11556.

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

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2025. Design2code: Benchmarking multimodal code generation for automated front-end engineering. *Preprint*, arXiv:2403.03163.

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. *Preprint*, arXiv:2405.07990.

Hao Wu and Wei Liu. 2025. Gcpo: When contrast fails, go gold. *arXiv preprint arXiv:2510.07790*.

Ronghuan Wu, Wanchao Su, Kede Ma, and Jing Liao. 2023. Iconshop: Text-guided vector icon synthesis with autoregressive transformers. *Preprint*, arXiv:2304.14400.

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. *Preprint*, arXiv:2406.09961.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *Preprint*, arXiv:2406.20098.

A Additional Qualitative Comparisons

Figure 4 provides a qualitative comparison that highlights the practical advantages of our method. Standard baselines and even leading proprietary MLLMs demonstrate distinct *perception* and *planning* failures when handling complex plotting tasks. The baseline model suffers from severe perception alignment issues, failing to recognize specific visual attributes and reverting to default library color palettes. In contrast, UniCoder produces a high-fidelity result that closely mirrors both the stylistic appearance and the structural integrity of the reference, with only negligible deviation.

To further demonstrate the generalization capability of our framework, this appendix provides an extended set of qualitative comparisons. The following figures showcase side-by-side comparisons between the executable code outputs generated by our method, UniCoder, and those produced by leading baseline MLLMs such as Qwen3-VL, GPT-5, and Gemini-3-Pro.

These examples (Figure 5, 6, 7, 8, 9) were selected to cover the diverse modalities addressed in this work. They serve to highlight the common failure modes of existing end-to-end MLLMs, such as *color hallucination*, *topological errors*, and *element omission* in complex layouts—and illustrate how our unified, reference-guided approach successfully overcomes these challenges. As shown in the comparisons, our method consistently produces code with significantly higher visual fidelity and structural completeness, more closely matching the precise design intent of the source visuals.

B Implementation Detail

B.1 Prompt Template

The prompt templates are shown in Figure 10, 11.

B.2 Training Details

Our training and experiments were conducted using the EasyR1 codebase on a computational node equipped with 8 NVIDIA A100 GPUs. We adopted a single-stage Reinforcement Learning pipeline. The model, initialized from Qwen3-VL-8B-Instruct, was directly optimized using our proposed algorithm based on Group Relative Policy Optimization (GRPO). The training was performed on a selected subset of 3000 from the MSRL, SVG-Stack, and Web2Code datasets. We configured the training with a global batch size of 128 and a rollout number (G) of 16. The model was optimized with a peak learning rate of 4×10^{-6} and bfloat16 precision. During inference and rollout generation, we used a temperature of 0.7 with a maximum response length of 4096 tokens.The reward function incorporated our element-level reward and the conventional CLIP score to explicitly align the generated code with visual constraints.

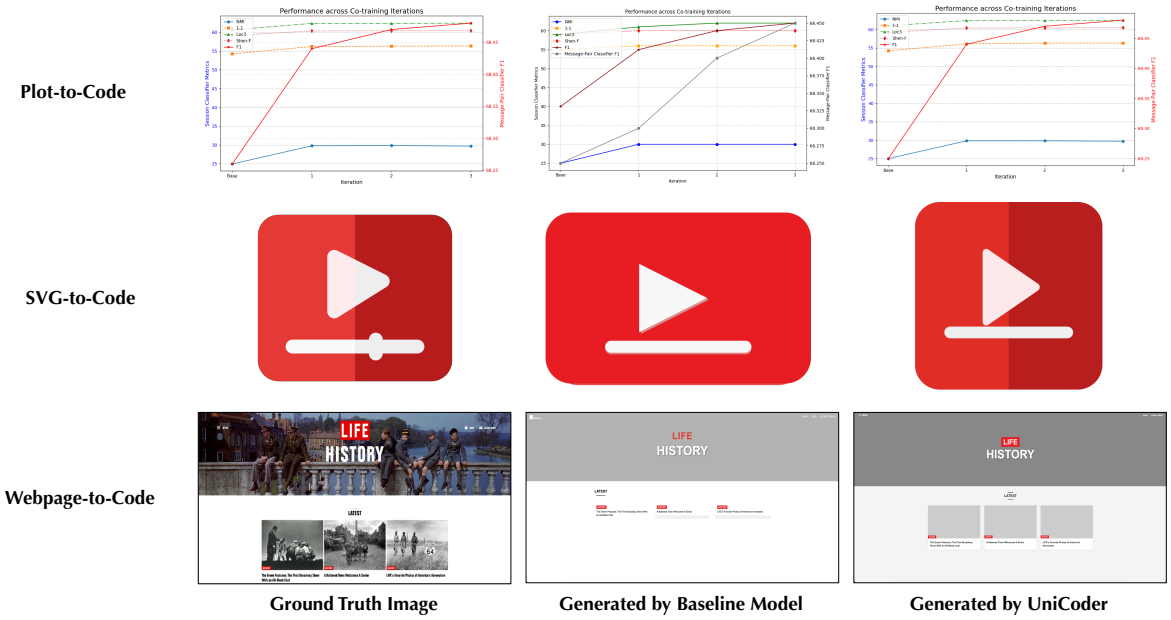

Figure 4: Qualitative comparison. As shown, the baseline model suffers from severe color hallucination and missing of elements, failing to replicate the visual palette. In contrast, our UniCoder achieves the highest fidelity, accurately recovering both the specific hex codes and the complete structure with only negligible errors.

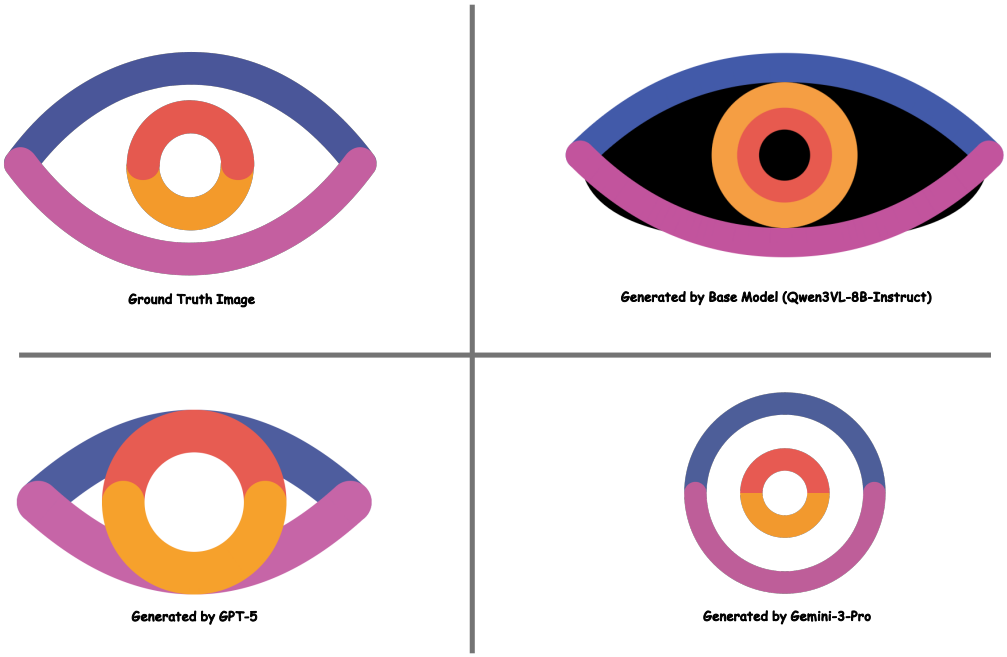

Figure 5: Qualitative comparison between our proposed method and various baselines.

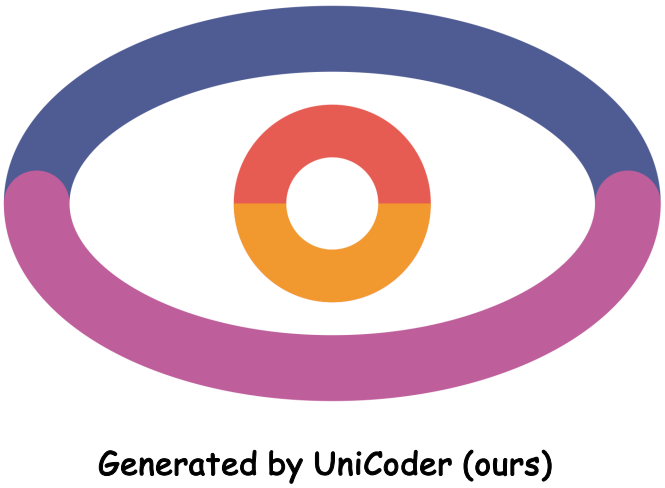

Figure 6: Qualitative comparison between our proposed method and various baselines.

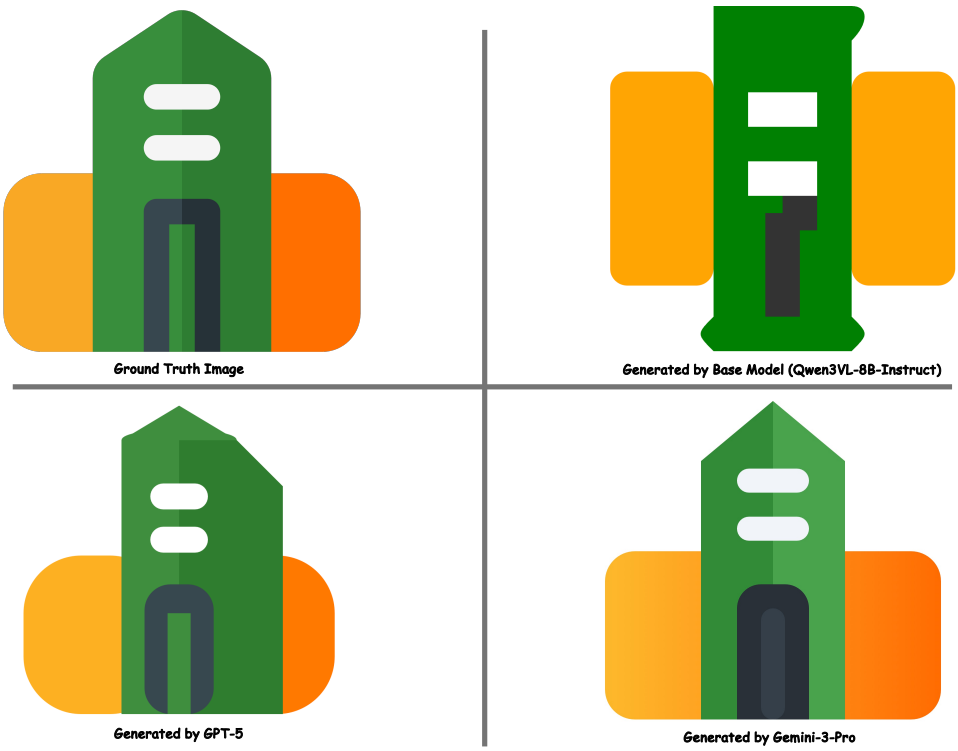

Figure 7: Qualitative comparison between our proposed method and various baselines.

Generated by UniCoder (ours)

Figure 8: Qualitative comparison between our proposed method and various baselines.

Figure 9: Qualitative comparison between our proposed method and various baselines.

Sample Prompt for V2C Generation

Python2Code Task	<image>\nYou are an expert Python developer who specializes in writing matplotlib code based on a given picture. I need your help to generate the Python code that can reproduce the picture based on the picture I provide.\nNote that it is necessary to use figsize=(10, 6) to set the image size to match the original size.\nNow, please give me the matplotlib code that reproduces the picture.
SVG2Code Task	<image>\nYou are an expert who specializes in writing SVG code based on a given picture. I need your help to generate the corresponding code that can reproduce the picture based on the picture I provide.
Webpage2Code Task	<image>\nYou are an expert webpage developer who specializes in writing HTML/CSS code based on a given picture. I need your help to generate the corresponding HTML/CSS code that can reproduce the picture based on the webpage screenshot provided. Replace the images with gray blocks.

Figure 10: Prompt Templates for Various Code Generation Tasks.

Prompt for Attribute Extractor

You are a helpful assistant for evaluating code and extract attributes from the code provided.

Analyze the following code (Python, SVG, HTML/CSS) and extract:
1. Text content: All titles, labels, legends, and text elements visible in the image.
2. Colors: All color specifications (hex codes or names). Convert named colors to HEX format (e.g., 'red' -> '#FF0000').

Code:{code}

Return a JSON object with keys "text" (list of strings) and "color" (list of hex strings).

Example format: {{"text": ["Title", "X Label"], "color": ["#FF0000", "#0000FF"]}}

Output ONLY the JSON.

Figure 11: Prompt Templates for Our Attribute Extract Module.