SKETCHFACTORY: A UNIFIED FRAMEWORK FOR LEARNING AND EVALUATING SKETCH-TO-CODE GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Generating runnable front-end code from hand-drawn sketches and textual requirements can accelerate interface development. Existing work on sketch-to-code has primarily focused on benchmark development. However, a unified framework integrating training data synthesis, task-specific evaluation, and reinforcement learning algorithm remains lacking. To address this gap, we introduce SKETCHFACTORY, a pipeline that synthesizes high-quality sketch-to-code data by deriving sketches and textual requirements from rendered webpages. We further propose an evaluation strategy that uses a vision-language model to judge rendered pages along interpretable quality dimensions, accommodating geometric imprecision in sketches. Building on these multidimensional scores, we develop difficulty-adaptive reward reweighting (DARR), a reinforcement learning algorithm that shifts reward weights toward the policy's weaker dimensions based on recent dimension-wise performance. Experiments on two benchmarks under two independent judges show that trained on SKETCHFACTORY data with DARR, our `DARR-Qwen3-8B` policy raises the overall score of its base model by 1.42–5.05 points on a 0–10 scale across the four benchmark–judge settings.

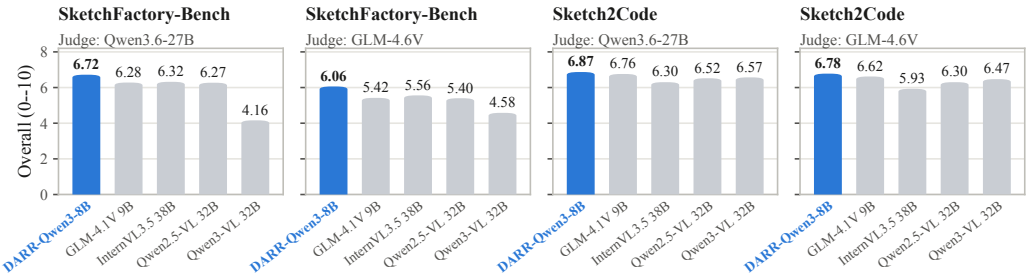


Figure 1: Overall scores on two sketch-to-code benchmarks under two independent judges.

1 INTRODUCTION

Using vision-language models to translate visual designs into runnable front-end code can substantially reduce front-end development costs (Yun et al., 2024; Si et al., 2025). In the early stages of design, complete visual specifications are often unavailable, and designers typically use sketches to convey page layouts without specifying other visual details (Huang et al., 2019; Li et al., 2025a). Generating front-end interfaces directly from such sketches and brief textual requirements could further shorten development cycles. However, sketches typically depict only the positions, sizes, and spatial arrangement of page regions, while omitting colors, typography, and textual content. To generate runnable code from a sketch, a model should reconstruct the page structure from sparse strokes and use textual requirements to supply the missing information, which requires tight coupling between visual layout understanding and textual requirement grounding.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Existing research primarily focuses on reconstructing code from high-fidelity designs, a task commonly referred to as design-to-code. Several datasets and evaluation benchmarks have been developed using webpage screenshots paired with their corresponding code (Si et al., 2025; Yun et al., 2024; Laurençon et al., 2024; Gui et al., 2025). Building on these resources, researchers have improved code generation through page decomposition before generation (Wan et al., 2025), data filtering based on automated feedback (Wu et al., 2024), and reinforcement learning with rendering feedback (Lu et al., 2025; Li et al., 2025b). These approaches take high-fidelity renderings with predefined colors, typography, and text as input, making the task closer to transcription. In comparison, sketch-to-code has received considerably less attention. Li et al. (2025a) introduced a benchmark for evaluating the ability of vision-language models to convert rough hand-drawn sketches into webpage prototypes, and reported that existing models still perform poorly on this task.

Despite this progress, three key challenges remain. (1) There is a lack of scalable pipelines for generating sketch-to-code training data. Constructing paired data requires annotators to draw sketches and engineers to implement the corresponding code, resulting in high annotation costs. (2) Existing evaluation metrics are poorly suited to sketch-to-code. Static visual metrics inherited from design-to-code, such as block matching by bounding-box overlap, rely on close spatial correspondence between the rendered pages of the generated and reference code (Si et al., 2025; Gui et al., 2025). Since hand-drawn sketches convey approximate positions, sizes, and proportions, these metrics may penalize geometric differences even when generated pages faithfully capture the intended layout. (3) Existing reinforcement learning methods for webpage code generation use fixed reward weights or separate training stages for textual and visual capabilities (Lu et al., 2025; Li et al., 2025b). These approaches do not adapt to uneven progress across reward dimensions during training.

To address these challenges, we propose SKETCHFACTORY, an integrated framework encompassing data construction, evaluation, and training, illustrated in Figure 2. For data construction, we render existing webpage code and derive sketches and textual requirements from the resulting pages. Each sketch combines edge detection on the rendering with text-line and image-region geometry read from the document object model (DOM). Each line of text is replaced with a sinusoidal curve that carries no semantic content, preserving layout information while preventing text leakage. For evaluation, we replace static pixel-level similarity metrics with a vision-language model that jointly examines the rendered reference and predicted pages and assigns separate scores along multiple interpretable dimensions. For training, we introduce *difficulty-adaptive reward reweighting* (DARR). DARR continually estimates recent performance along each evaluation dimension and assigns greater reward weights to dimensions with lower current scores, which mitigates the imbalance across dimensions during multi-dimensional learning. Our main contributions are as follows:

❶ **Data Construction and Benchmark.** We introduce SKETCHFACTORY, a pipeline for synthesizing consistent (sketch, requirement, code) training triplets from rendered webpages. We also construct and release SKETCHFACTORY-BENCH, a manually annotated benchmark comprising 414 samples with human-drawn sketches, to evaluate transfer from synthetic to real sketches.

❷ **Training Strategy.** We propose DARR, which replaces fixed reward weights with weights derived from the policy's own recent performance, shifting reward toward the dimensions on which it currently scores lowest, which alleviates the imbalance across multiple dimensions.

❸ **Experimental Evaluation.** Across two benchmarks and two independent judges, DARR consistently outperforms fixed-weight training with fewer optimization steps. As summarized in Figure 1, our DARR-trained 8B policy surpasses open-source models.

2 RELATED WORK

2.1 DATA AND BENCHMARKS FOR VISUAL CODE GENERATION

Research on visual code generation has largely focused on high-fidelity webpage screenshots. Design2Code benchmarks screenshot-to-code generation on real webpages (Si et al., 2025), while WebSight and WebCode2M provide synthetic and real-world screenshot–code pairs for training (Laurençon et al., 2024; Gui et al., 2025). Web2Code combines instruction-tuning data with an evaluation framework for webpage understanding and code generation (Yun et al., 2024). These resources support reconstruction from detailed visual specifications but do not directly provide sketch-conditioned training data.

Sketch2Code extends evaluation to hand-drawn sketches, covering both direct generation and interactive refinement through user feedback (Li et al., 2025a). Its focus is on assessing existing models rather than developing a training framework. Our work complements this benchmark with a pipeline that derives aligned sketches and textual requirements from rendered webpages, together with a manually annotated benchmark for evaluating transfer to human-drawn sketches.

2.2 Visual Evaluation and Reinforcement Learning

Evaluation of generated webpages includes both static similarity metrics and model-based judgments. Design2Code measures visual fidelity through element matching (Si et al., 2025), whereas Web2Code uses a multimodal judge to assess rendered pages across ten visual criteria (Yun et al., 2024). We build on multidimensional visual assessment for sketch-to-code, where geometric imprecision makes close spatial matching an imperfect measure of whether the intended layout is preserved.

Visual feedback also supports reinforcement learning. WebGen-Agent uses screenshot and GUI-agent feedback for step-level optimization (Lu et al., 2025), while ReLook incorporates a multimodal critic into an iterative generation and refinement process (Li et al., 2025b). Beyond webpage generation, dynamic reward weighting has been studied for exploring objective trade-offs in multi-objective alignment (Lu et al., 2026). Our focus is on uneven learning across visual reward dimensions. DARR adjusts reward weights according to recent dimension-wise performance.

3 Preliminaries

We first introduce the notation for the sketch-to-code generation setting. Consider an input instance consisting of a sketch image $\boldsymbol{s}$ and an associated requirement description $\boldsymbol{q}$, where $\boldsymbol{s} \in \mathcal{S}$ is an image over a pixel space and $\boldsymbol{q} \in \mathcal{Q}$, together with the target code $\boldsymbol{y} \in \mathcal{Y}$, are token sequences over a vocabulary $\mathcal{V}$. Specifically, $\boldsymbol{q} = (q^1, q^2, \ldots, q^m)$ and $\boldsymbol{y} = (y^1, y^2, \ldots, y^n)$, where $q^j, y^j \in \mathcal{V}$ and m, n are the lengths of the requirement and the code, respectively. The training examples are i.i.d. drawn from the distribution $p(\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{y}^*)$, where $\boldsymbol{y}^*$ denotes the reference implementation.

We therefore introduce a deterministic rendering operator $\mathcal{G} : \mathcal{Y} \to \mathcal{I}$ that maps a code sequence to an image in a rendering space $\mathcal{I}$. The objective is to learn a policy π_θ that generates code maximizing an expected reward that measures the similarity between the rendered prediction and the rendered reference webpages, with KL divergence regularization:

$$\mathcal{R}(\theta) = \mathbb{E}_{\substack{(\boldsymbol{s},\boldsymbol{q},\boldsymbol{y}^*)\sim p(\boldsymbol{s},\boldsymbol{q},\boldsymbol{y}^*),\\ \boldsymbol{y}\sim\pi_\theta(\cdot|\boldsymbol{s},\boldsymbol{q})}} \big[r\big(\mathcal{G}(\boldsymbol{y}^*), \mathcal{G}(\boldsymbol{y})\big) \big] - \lambda D_{\mathrm{KL}}(\pi_\theta(\cdot \mid \boldsymbol{s}, \boldsymbol{q}) \,\|\, \pi_{\text{ref}}(\cdot \mid \boldsymbol{s}, \boldsymbol{q})) \,, \tag{1}$$

where $r(\cdot, \cdot)$ denotes the reward function used to evaluate the visual similarity between the rendered pages, λ is the KL divergence regularization coefficient, and π_{ref} is the reference policy.

Multi-dimensional reward. The quality of generated webpages is typically evaluated across multiple dimensions, including visual structure, aesthetics, and content fidelity. Accordingly, we represent the reward as a vector in which each entry is the score of one dimension,

$$\boldsymbol{r} = \big(r_1, r_2, \ldots, r_K\big) \in [0, r_{\max}]^K, \qquad \boldsymbol{r} = \Phi\big(\mathcal{G}(\boldsymbol{y}^*), \mathcal{G}(\boldsymbol{y})\big), \tag{2}$$

where Φ scores a rendered pair along K interpretable *dimensions* of visual quality and $r_{\max}$ is the maximum attainable score per dimension. The specific settings of Φ, K, and $r_{\max}$ are described in Section 5.1. To optimize the policy, the score vector must be reduced to a scalar reward. Common practice assigns each dimension a preset weight reflecting its importance,

$$r_{\text{fixed}} = \sum_{k=1}^{K} \bar{w}_k \, r_k, \qquad \bar{w}_k \geq 0, \quad \sum_{k=1}^{K} \bar{w}_k = 1, \tag{3}$$

where the weights $\bar{\boldsymbol{w}}$ are specified before training and held constant throughout. Additionally, evaluating Eq. (2) requires successful code extraction and rendering. We write $\xi \in \{0, 1\}$ for the indicator that all steps succeed, and define the reward of an invalid rollout to be zero.

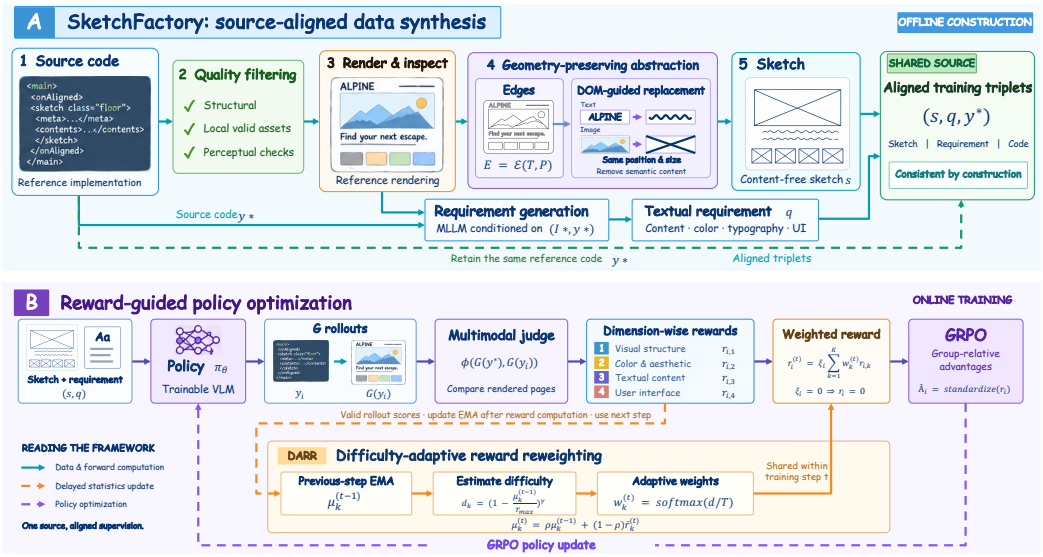


Figure 2: Overview of SKETCHFACTORY and difficulty-adaptive reward reweighting (DARR).

4 METHOD

Figure 2 summarizes the framework, which comprises data synthesis and a training strategy. Data synthesis takes the source code of an existing webpage and derives both a sketch and a textual requirement from it (Section 4.1). Because all three components originate from the same source, the resulting triplets are consistent without manual annotation, and the code serves directly as the reference target. The training strategy then optimizes the policy on these triplets with reinforcement learning (Section 4.2). A multimodal judge scores each rollout along K dimensions, and DARR tracks the recent score of every dimension to reweight them before aggregation, directing reward toward the dimensions the policy currently handles worst.

4.1 SKETCHFACTORY: AUTOMATED TRAINING-DATA SYNTHESIS

Manual construction of training triplets $(\boldsymbol{s}, \boldsymbol{q}, \boldsymbol{y}^*)$ requires both sketch annotation and code implementation, with additional effort to ensure their consistency. SKETCHFACTORY instead derives the sketch and textual requirement from the source code $\boldsymbol{y}^*$ of an existing webpage and its rendering, aligning all three components through a shared source.

Sketch synthesis begins with the reference webpage code $\boldsymbol{y}^*$. We render it to obtain $\boldsymbol{I}^* = \mathcal{G}(\boldsymbol{y}^*)$, and simultaneously query the live document object model (DOM) for the geometry of two element classes: the set of *text-line rectangles* $\mathcal{T} = \{\boldsymbol{t}_1, \ldots, \boldsymbol{t}_{|\mathcal{T}|}\}$ and the set of *image rectangles* $\mathcal{P} = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{|\mathcal{P}|}\}$, where each rectangle is a tuple (x, y, w, h) in page coordinates.

The sketch is then produced by a three-step transform. First, an edge detector $\mathcal{E}$ extracts the structural contours of the rendering,

$$\boldsymbol{E} = \mathcal{E}(\boldsymbol{I}^*) \in \{0,1\}^{H \times W}, \tag{4}$$

which preserves borders, separators, and block boundaries. Second, the regions occupied by $\mathcal{T}$ and $\mathcal{P}$ are cleared and replaced by content-free placeholders to remove textual and image content while retaining spatial information. Each text line $\boldsymbol{t} = (x, y, w, h)$ is replaced by a sinusoid whose geometry scales with the line height,

$$f_{\boldsymbol{t}}(u) = y + \frac{h}{2} + a(h)\,\sin\!\left(\frac{2\pi(u-x)}{\ell(h)}\right), \qquad u \in [x, x+w], \tag{5}$$

where the amplitude $a(h)$, wavelength $\ell(h)$, and stroke width are monotone in h. This preserves the position, extent, and visual weight of a text line while removing its semantic content. Each image

Algorithm 1 One DARR training step

Require: moving average $\boldsymbol{\mu}^{(\tau-1)}$, step index τ

$d_k \leftarrow 1 - \mu_k^{(\tau-1)}/r_{\max}$ for all k ▷ difficulty, Eq. (9)
$\boldsymbol{w}^{(\tau)} \leftarrow \textsc{Softmax}_T(\boldsymbol{d})$ ▷ Eq. (10)
sample $\{\boldsymbol{y}_i\}_{i=1}^{G} \sim \pi_\theta(\cdot \mid \boldsymbol{s}, \boldsymbol{q})$
for each rollout i **do**
    extract code from $\boldsymbol{y}_i$, render $\mathcal{G}(\boldsymbol{y}_i)$, evaluate $\boldsymbol{r}_i = \Phi(\mathcal{G}(\boldsymbol{y}^*), \mathcal{G}(\boldsymbol{y}_i))$
    $\xi_i \leftarrow 0$ if any step failed, else $\xi_i \leftarrow 1$
    $r_i \leftarrow \xi_i \cdot \langle \boldsymbol{w}^{(\tau)}, \boldsymbol{r}_i \rangle$ ▷ Eq. (11)
end for
update θ by GRPO on $\{r_i\}_{i=1}^{G}$ ▷ standard GRPO update
$\mu_k^{(\tau)} \leftarrow \rho\, \mu_k^{(\tau-1)} + (1-\rho)\, \bar{r}_k^{(\tau)}$ ▷ after rewards, Eqs. (7)–(8)

region $\boldsymbol{p}$ is replaced by a filled rectangle outlined with diagonals. Finally, the composed result is rendered as black strokes on a white canvas to yield the sketch $\boldsymbol{s}$.

Although the synthesized sketch preserves spatial structure, it omits the textual content and visual details needed to fully specify the reference implementation. To provide these missing specifications, we generate the textual requirement $\boldsymbol{q}$ using a multimodal model conditioned on both the rendering and the reference webpage code,

$$\boldsymbol{q} \sim \pi_{\text{desc}}\big(\cdot \mid \boldsymbol{I}^*, \boldsymbol{y}^*\big). \tag{6}$$

Conditioning on $\boldsymbol{y}^*$ in addition to $\boldsymbol{I}^*$ enables precise specification of properties that may be ambiguous in the rendering. In addition to aligning sketches and requirements with their source code, the pipeline filters reference webpages to ensure that they provide reproducible and visually valid reward targets. Structural and perceptual filtering are applied in sequence. Structural filtering removes webpage code samples that reference remote or missing assets, which can compromise rendering reproducibility. Perceptual filtering uses a multimodal model to exclude pages with substantial visual defects, including broken images, collapsed layouts, minimal content, and blocking interstitials.

4.2 DIFFICULTY-ADAPTIVE REWARD REWEIGHTING

Existing training approaches typically rely on manually specified reward weights or separate training stages to develop different capabilities, without adapting reward allocation to dimension-wise performance during training. DARR instead estimates a step-dependent weight vector $\boldsymbol{w}^{(\tau)}$ from recent evaluation scores, assigning greater weight to lower-scoring dimensions.

To estimate dimension-wise performance, we aggregate evaluation scores from valid rollouts at each training step. Let τ index training steps and let $\mathcal{B}^{(\tau)}$ denote the set of rollouts at step τ with $\xi_i = 1$. The mean score of dimension k over the valid rollouts of the step is

$$\bar{r}_k^{(\tau)} = \frac{1}{|\mathcal{B}^{(\tau)}|} \sum_{i \in \mathcal{B}^{(\tau)}} r_{i,k}. \tag{7}$$

Step-level performance estimates are subject to variation in input batches and policy sampling. We therefore maintain an exponential moving average per dimension,

$$\mu_k^{(\tau)} = \rho\, \mu_k^{(\tau-1)} + (1-\rho)\, \bar{r}_k^{(\tau)}, \qquad \rho \in (0,1), \tag{8}$$

initialized from the first step. Using the smoothed performance estimates, we define the difficulty of dimension k as

$$d_k^{(\tau)} = 1 - \frac{\mu_k^{(\tau)}}{r_{\max}}. \tag{9}$$

Lower smoothed scores correspond to higher estimated difficulty. The resulting difficulty estimates are mapped to reward weights through a temperature-controlled softmax,

$$w_k^{(\tau)} = \frac{\exp\big(d_k^{(\tau)}/T\big)}{\sum_{j=1}^{K} \exp\big(d_j^{(\tau)}/T\big)}, \tag{10}$$

where the temperature $T > 0$ controls weight concentration. As $T \to 0$, weights concentrate on dimensions with maximal estimated difficulty. As $T \to \infty$, the weights converge to $1/K$.

These weights determine the scalar reward of rollout i at step τ,

$$r_i^{(\tau)} = \xi_i \cdot \sum_{k=1}^{K} w_k^{(\tau)} r_{i,k}, \tag{11}$$

For a group of G rollouts generated from the same input, GRPO computes the advantage of rollout i by standardizing its reward within the group,

$$\hat{A}_i^{(\tau)} = \frac{r_i^{(\tau)} - \mathrm{mean}\big(\{r_j^{(\tau)}\}_{j=1}^{G}\big)}{\mathrm{std}\big(\{r_j^{(\tau)}\}_{j=1}^{G}\big) + \varepsilon}, \tag{12}$$

where $\varepsilon > 0$ ensures numerical stability. The same advantage is assigned to all tokens in the rollout, and the policy is optimized using the standard GRPO objective (Shao et al., 2024). To ensure that rewards within each training step are evaluated under a common aggregation function, reward computation and statistical updates follow two ordering constraints. First, $\boldsymbol{w}^{(\tau)}$ is computed from $\boldsymbol{\mu}^{(\tau-1)}$ and shared across all rollouts at step τ. Thus, current rollouts do not affect their own reward weights, and group standardization compares rewards under a common aggregation function. Second, the moving average is updated only after all rewards at step τ have been computed. The resulting statistics are used for subsequent weight updates. Algorithm 1 summarizes one step.

5 Experiments

5.1 Experimental Setup

We crawl HTML code from public web design websites and render each page to obtain its corresponding image. We then apply rule-based quality filters to remove samples with missing images, abnormal overlays, rendering failures, and other defects. After applying the synthesis and filtering pipeline described in Section 4.1, we obtain approximately $20{,}000$ training samples.

To evaluate the quality of our synthetic data, we sample an additional 400 webpages held out from training to construct SketchFactory-Bench. Twelve annotators first assess these webpages manually, retaining 348 that pass the quality review, and then create sketch annotations for the retained samples. This process yields 414 sketch–webpage pairs in total, as 66 webpages receive a second independent sketch annotation. The benchmark therefore captures variation in human sketching styles while using the original webpage code as the reference implementation. We further evaluate on Sketch2Code (Li et al., 2025a), an external benchmark of 731 sketches drawn by 21 annotators over 484 webpages sourced from the C4 validation set by Si et al. (2025). Appendix E compares the inputs of the two benchmarks side by side.

To automatically evaluate the quality of generated webpages, we use VLM-as-judge to compare each generated rendering against its reference using the criteria summarized in Appendix Table 3. The full evaluation prompts are provided in Appendix A. The judge assigns a score from 0 to 10 to each of the ten criteria. The overall score is the equally weighted mean of the four dimension scores. `Qwen3.6-27B` provides training rewards and serves as one evaluation judge. We additionally evaluate all models with `GLM-4.6V` to examine whether improvements persist under a different evaluator. Both judges use temperature 0. We train `Qwen2.5-VL-7B-Instruct`[1] and `Qwen3-VL-8B-Instruct`[2]. Detailed training configurations are provided in Table 4 in the appendix. Detailed sketch-synthesis and inference configurations are provided in Appendix B.

5.2 Main Results

Overall performance. Tables 1 and 2 compare our trained policies with open-source vision-language models under `Qwen3.6-27B` and `GLM-4.6V`, respectively. `DARR-Qwen3-8B` ranks first on both benchmarks under both judges. On SketchFactory-Bench, it obtains overall scores

[1] https://huggingface.co/Qwen/Qwen2.5-VL-7B-Instruct
[2] https://huggingface.co/Qwen/Qwen3-VL-8B-Instruct

Table 1: Per-dimension and overall scores (0–10) under the **Qwen3.6-27B** judge. r_1 visual structure, r_2 color & aesthetic, r_3 textual content, r_4 user interface; **Ov** is the equally weighted mean of the four dimension scores.

			SketchFactory-Bench					Sketch2Code				
Family	Model	Size	r_1	r_2	r_3	r_4	**Ov**	r_1	r_2	r_3	r_4	**Ov**
LLaVA-1.5	`llava-1.5-7b`	7B	0.27	0.33	0.80	0.28	0.42	0.72	1.23	1.93	0.71	1.15
	`llava-1.5-13b`	13B	0.76	1.08	1.72	0.75	1.08	0.79	1.22	2.06	0.73	1.20
LLaVA-NeXT	`v1.6-mistral-7b`	7B	1.47	2.39	2.80	1.69	2.09	1.88	2.29	3.98	2.16	2.58
	`llama3-next-8b`	8B	1.40	2.26	3.16	1.81	2.16	1.35	1.94	3.23	1.55	2.02
	`v1.6-vicuna-13b`	13B	0.00	0.00	0.01	0.00	0.01	0.05	0.21	0.16	0.05	0.12
	`v1.6-34b`	34B	0.05	0.24	0.05	0.03	0.09	0.82	1.17	1.38	0.87	1.06
LLaVA-OV	`ov-qwen2-7b`	7B	1.72	2.18	3.16	1.76	2.21	2.63	3.46	4.84	2.83	3.44
Gemma-3-it	`gemma-3-12b-it`	12B	3.04	3.85	6.31	3.32	4.13	4.09	5.18	7.12	4.36	5.19
	`gemma-3-27b-it`	27B	5.67	5.53	6.92	5.28	5.85	5.99	6.06	7.38	5.67	6.27
InternVL3.5	`InternVL3.5-8B`	8B	5.12	6.11	7.04	5.35	5.91	5.24	6.18	7.35	5.30	6.02
	`InternVL3.5-14B`	14B	5.32	5.98	7.01	5.44	5.94	5.55	6.19	7.30	5.51	6.14
	`InternVL3.5-38B`	38B	5.98	6.21	7.37	5.71	6.32	5.75	6.32	7.47	5.65	6.30
Qwen2.5-VL	`Qwen2.5-VL-7B`	7B	3.60	5.07	6.16	4.15	4.74	4.45	6.06	7.05	4.70	5.57
	`Qwen2.5-VL-32B`	32B	5.96	6.09	7.29	5.73	6.27	6.11	6.50	7.49	5.97	6.52
Qwen3-VL	`Qwen3-VL-8B`	8B	1.45	2.08	1.75	1.38	1.67	4.90	5.78	6.16	4.95	5.45
	`Qwen3-VL-32B`	32B	4.04	4.38	4.49	3.74	4.16	6.23	6.78	7.29	5.99	6.57
GLM-4.1V	`GLM-4.1V-9B-Think`	9B	6.02	6.44	6.77	5.90	6.28	**6.39**	6.97	7.46	**6.21**	6.76
Ours: GRPO on SKETCHFACTORY *data*												
Qwen2.5-VL-7B	base (no RL)	7B	3.60	5.07	6.16	4.15	4.74	4.45	6.06	7.05	4.70	5.57
	+ GRPO, fixed weight	7B	5.30	6.54	7.22	5.42	6.12	5.61	6.90	7.44	5.63	6.39
	+ GRPO, **DARR**	7B	5.76	6.92	**7.51**	5.81	6.50	6.06	7.22	**7.71**	6.05	6.76
Qwen3-VL-8B	base (no RL)	8B	1.45	2.08	1.75	1.38	1.67	4.90	5.78	6.16	4.95	5.45
	+ GRPO, fixed weight	8B	5.82	6.95	6.89	5.71	6.34	6.21	7.26	7.60	6.18	6.81
	+ GRPO, **DARR**	8B	**6.33**	**7.25**	7.23	**6.07**	**6.72**	6.31	**7.32**	7.64	6.20	**6.87**

of 6.72 and 6.06 under `Qwen3.6-27B` and `GLM-4.6V`, outperforming `InternVL3.5-38B` by 0.40 and 0.50 points. On Sketch2Code, it scores 6.87 and 6.78, surpassing `Qwen3-VL-32B` by 0.30 and 0.31 points. These results demonstrate that our trained 8B policy can outperform evaluated open-source models with substantially larger parameter counts.

Effect of SKETCHFACTORY training. GRPO training consistently improves both policy backbones over their instruction-tuned base models, regardless of the reward-weighting strategy, benchmark, or judge. Here fixed-weight GRPO refers to Eq. (3) with uniform weights $\bar{w}_k = 1/K$, the default in the absence of prior knowledge about dimension importance. For `Qwen2.5-VL-7B`, fixed-weight GRPO yields gains of 0.82–1.73 points across the four benchmark–judge combinations, while DARR yields gains of 1.19–1.87 points. The improvements are larger for `Qwen3-VL-8B`, fixed-weight GRPO gains 1.36–4.67 points, and DARR gains 1.42–5.05 points. Consistent gains on both the human-drawn SKETCHFACTORY-BENCH and the external Sketch2Code benchmark demonstrate that training on our synthetic data transfers beyond the training distribution.

Benefit of adaptive reward weighting. Holding the training data and policy backbone fixed, DARR outperforms fixed-weight GRPO in all settings. Under `Qwen3.6-27B`, DARR improves SKETCHFACTORY-BENCH and Sketch2Code by 0.38 and 0.37 points for `Qwen2.5-VL-7B`, and by 0.38 and 0.06 points for `Qwen3-VL-8B`. Under `GLM-4.6V`, the corresponding gains are 0.14 and 0.12 points for `Qwen2.5-VL-7B`, and 0.32 and 0.13 points for `Qwen3-VL-8B`. The consistent gains under `GLM-4.6V`, a judge never used to provide training rewards, indicate that DARR improves webpage quality rather than overfitting to the reward model. Averaging the two judges' per-sample scores and clustering sketches by their reference webpage, all four backbone–benchmark

Table 2: Per-dimension and overall scores (0–10) under the **GLM-4.6V** judge. r_1 visual structure, r_2 color & aesthetic, r_3 textual content, r_4 user interface; **Ov** is the equally weighted mean of the four dimension scores.

			SketchFactory-Bench					Sketch2Code				
Family	Model	Size	r_1	r_2	r_3	r_4	**Ov**	r_1	r_2	r_3	r_4	**Ov**
LLaVA-1.5	`llava-1.5-7b`	7B	0.05	0.04	0.21	0.04	0.08	0.18	0.15	0.40	0.13	0.21
	`llava-1.5-13b`	13B	0.29	0.30	0.58	0.25	0.35	0.25	0.22	0.62	0.18	0.32
LLaVA-NeXT	`v1.6-mistral-7b`	7B	0.92	1.02	1.40	0.79	1.03	0.80	0.71	1.45	0.64	0.90
	`llama3-next-8b`	8B	0.75	0.84	1.12	0.66	0.84	0.68	0.66	1.24	0.55	0.78
	`v1.6-vicuna-13b`	13B	0.00	0.00	0.00	0.00	0.00	0.02	0.03	0.04	0.02	0.03
	`v1.6-34b`	34B	0.02	0.01	0.01	0.01	0.01	0.36	0.37	0.54	0.29	0.39
LLaVA-OV	`ov-qwen2-7b`	7B	0.95	1.04	1.47	0.83	1.07	1.41	1.43	2.32	1.15	1.58
Gemma-3-it	`gemma-3-12b-it`	12B	2.82	2.64	3.67	2.47	2.90	3.85	3.76	5.36	3.56	4.13
	`gemma-3-27b-it`	27B	5.06	4.52	5.48	4.64	4.93	5.83	5.49	6.92	5.46	5.93
InternVL3.5	`InternVL3.5-8B`	8B	4.60	4.55	5.36	4.34	4.71	5.36	5.21	6.52	5.08	5.54
	`InternVL3.5-14B`	14B	4.91	4.70	5.44	4.57	4.91	5.43	5.20	6.67	5.13	5.61
	`InternVL3.5-38B`	38B	5.55	5.29	6.16	5.23	5.56	5.78	5.57	6.87	5.52	5.93
Qwen2.5-VL	`Qwen2.5-VL-7B`	7B	3.37	3.38	4.03	3.07	3.46	4.68	4.81	5.96	4.34	4.95
	`Qwen2.5-VL-32B`	32B	5.39	5.11	6.01	5.07	5.40	6.17	5.97	7.16	5.89	6.30
Qwen3-VL	`Qwen3-VL-8B`	8B	1.36	1.39	1.51	1.30	1.39	5.13	5.15	5.86	4.94	5.27
	`Qwen3-VL-32B`	32B	4.61	4.49	4.82	4.42	4.58	6.39	6.26	7.09	6.14	6.47
GLM-4.1V	`GLM-4.1V-9B-Think`	9B	5.44	5.25	5.78	5.22	5.42	6.49	6.47	7.19	6.31	6.62
Ours: GRPO on SketchFactory *data*												
Qwen2.5-VL-7B	base (no RL)	7B	3.37	3.38	4.03	3.07	3.46	4.68	4.81	5.96	4.34	4.95
	+ GRPO, fixed weight	7B	5.08	5.11	5.75	4.83	5.19	6.11	6.16	7.17	5.83	6.32
	+ GRPO, **DARR**	7B	5.17	5.33	5.80	5.00	5.33	6.27	6.31	7.22	5.98	6.44
Qwen3-VL-8B	base (no RL)	8B	1.36	1.39	1.51	1.30	1.39	5.13	5.15	5.86	4.94	5.27
	+ GRPO, fixed weight	8B	5.73	5.65	6.10	5.48	5.74	6.48	6.52	7.35	6.24	6.65
	+ GRPO, **DARR**	8B	**6.05**	**6.01**	**6.36**	**5.83**	**6.06**	**6.67**	**6.63**	**7.43**	**6.40**	**6.78**

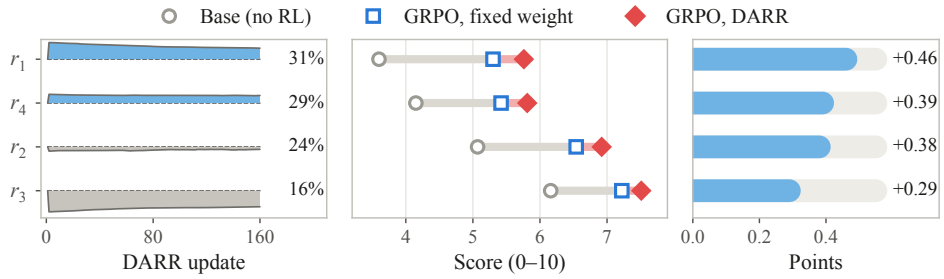


Figure 3: Dimension-wise behavior of DARR for `Qwen2.5-VL-7B` under the `Qwen3.6-27B` judge, from left to right: (a) reward weight over training, shown as the deviation from the uniform 25%, blue above and gray below; (b) per-dimension score of the base policy, fixed-weight GRPO, and DARR on SketchFactory-Bench; (c) the resulting DARR gain over fixed weighting.

improvements remain significant after Holm correction; Appendix C reports the paired tests and confidence intervals.

5.3 Dimension-wise Analysis

Under fixed weighting, the policy can raise its overall score by first improving the dimensions it finds easy while neglecting the harder ones, which leaves learning unbalanced across dimensions.

For `Qwen2.5-VL-7B`, textual content indeed stays the highest-scoring dimension while visual structure and user interface stay the lowest, and the gap between the best and the worst dimension remains between 1.50 and 1.87 points, which motivates shifting reward weight toward dimensions with lower recent scores. DARR mitigates this imbalance by adaptively shifting reward weight toward weaker dimensions. A single update after the uniform initialization already moves weight toward visual structure and UI and away from textual content (Figure 3(a)), and the allocation settles at 31.29%, 23.56%, 15.85%, and 29.30% for r_1 through r_4. In training reward, visual structure and UI gain 1.39 and 1.16 points against 0.49 for textual content, narrowing the spread across the four dimensions from 2.94 to 2.04 points. As the gap closes, the visual-structure weight falls back from its early peak of 34.51% to 31.29% while textual content rises from 12.95% to 15.85%, so the allocation keeps tracking the policy's current weaknesses.

The same emphasis carries over to held-out data, where Figure 3(b, c) compares the two weighting strategies on SKETCHFACTORY-BENCH. DARR improves every dimension over fixed weighting, with the largest gain on visual structure, the base policy's weakest dimension, and the smallest on textual content, its strongest. A lower reward weight therefore does not imply a lower score, because the four dimensions reinforce one another. A more faithful layout also removes failures that other criteria penalize, such as text hidden behind an overlapping element, which the judge scores under textual content as well as under visual structure. Emphasizing the weaker dimensions thus pulls the stronger ones along with them instead of trading them off.

5.4 TRANSFER TO SCREENSHOT-TO-CODE GENERATION

To further evaluate whether our data and method improve webpage generation beyond sketch inputs, we conduct a cross-task evaluation on the design-to-code benchmark IW-Bench V2 (Guo et al., 2025). The downloaded benchmark contains 1,702 webpage screenshots paired with reference implementations; each policy receives a screenshot and a shared HTML reconstruction instruction. We report Element Accuracy (EA), the fraction of reference DOM elements matched at a similarity threshold of 0.9, and Layout Accuracy (LA), the normalized longest common subsequence of matched elements in DOM order. Appendix D provides implementation details.

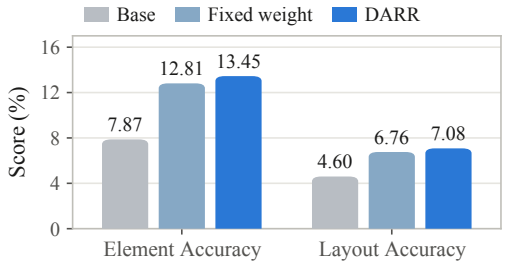


Figure 4: **IW-Bench V2 results.**

As shown in Figure 4, both fixed-weight and DARR training improve EA and LA over the base policy. For `Qwen3-VL-8B`, DARR achieves 13.45% EA and 7.08% LA, exceeding fixed-weight training by 0.64 and 0.32 percentage points. These results show that capabilities learned from our sketch-to-code data transfer to screenshot-conditioned webpage reconstruction, improving both element coverage and DOM ordering.

6 CONCLUSION

We presented SKETCHFACTORY, which couples data synthesis, evaluation, and training for sketch-to-code generation: it derives consistent training triplets from rendered webpages, releases 414 human-drawn sketches as SKETCHFACTORY-BENCH, and scores rendered pages with a vision-language judge along four interpretable dimensions instead of pixel-level similarity. These dimension-wise scores drive DARR, which shifts reward weight toward the policy's currently weakest dimensions. Across two benchmarks and two independent judges, DARR outperforms fixed-weight training in every setting. Our `DARR-Qwen3-8B` policy surpasses open-source models four to five times its size, and the gains carry over to screenshot-to-code reconstruction on IW-Bench. Notably, it improves every dimension at once rather than trading weaker ones against stronger. Since both reward and evaluation come from vision-language judges, the remaining headroom is bounded by what they can discriminate, making human studies and stronger evaluators natural next steps.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets. Specifically, the SKETCH-FACTORY pipeline uses a multimodal model to produce the textual requirement of each training triplet (Eq. (6)) and to perceptually filter reference webpages (Section 4.1), and vision-language models serve as the judges that supply training rewards and evaluation scores (Section 5.1). Additionally, we used generative AI tools to polish the writing of this paper, and we have reviewed and take responsibility for all such text.

REFERENCES

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Bohua Chen, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. Webcode2m: A real-world dataset for code generation from webpage designs. In *Proceedings of the ACM Web Conference (WWW)*, pp. 1834–1845, 2025. arXiv:2404.06369.

Hongcheng Guo, Wei Zhang, Junhao Chen, Yaonan Gu, Jian Yang, Junjia Du, Shaosheng Cao, Binyuan Hui, Tianyu Liu, Jianxin Ma, et al. Iw-bench: Evaluating large multimodal models for converting image-to-web. pp. 6449–6466, 2025.

Forrest Huang, John F. Canny, and Jeffrey Nichols. Swire: Sketch-based user interface retrieval. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI)*, 2019.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.

Ryan Li, Yanzhe Zhang, and Diyi Yang. Sketch2code: Evaluating vision-language models for interactive web design prototyping. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025a. arXiv:2410.16232.

Yuhang Li, Chenchen Zhang, Ruilin Lv, Ao Liu, Ken Deng, Yuanxing Zhang, Jiaheng Liu, Wiggin Zhou, and Bo Zhou. Relook: Vision-grounded rl with a multimodal llm critic for agentic web coding. *arXiv preprint arXiv:2510.11498*, 2025b.

Yining Lu, Zilong Wang, Shiyang Li, Xin Liu, Changlong Yu, Qingyu Yin, Zhan Shi, Zixuan Zhang, and Meng Jiang. Learning to optimize multi-objective alignment through dynamic reward weighting. *Transactions of the Association for Computational Linguistics*, 14:1051–1073, 2026.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning. *arXiv preprint arXiv:2509.22644*, 2025.

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

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *Proceedings of the 2025 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*, pp. 3956–3974, 2025. arXiv:2403.03163.

Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R. Lyu. Divide-and-conquer: Generating ui code from screenshots. *Proceedings of the ACM on Software Engineering (FSE)*, 2:2099–2122, 2025. arXiv:2406.16386.

Jason Wu, Eldon Schoop, Alan Leung, Titus Barik, Jeffrey P Bigham, and Jeffrey Nichols. UICoder: Finetuning large language models to generate user interface code through automated feedback. In *Proceedings of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 7511–7525, 2024.

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

Table 3: Evaluation dimensions and their constituent criteria.

Dimension	Evaluation criteria
r_1 visual structure	layout, alignment, proportion, harmony
r_2 color & aesthetic	color scheme, aesthetic resemblance
r_3 textual content	font, text match, numeric/special characters
r_4 user interface	UI consistency

A VLM JUDGE PROMPTS

The following prompts reproduce the text used by our shared judge implementation for both training rewards and offline visual evaluation. The evaluation rubric follows Web2Code (Yun et al., 2024). The system message contains the system prompt below. The user message contains the evaluation prompt followed by two images in order: the reference rendering and the generated rendering. The judge returns ten comma-separated scores in the order listed in the prompt. Scores 1–4 are averaged to obtain r_1, scores 5–6 to obtain r_2, and scores 7–9 to obtain r_3; score 10 gives r_4, as summarized in Table 3. The overall evaluation score is the equally weighted mean of these four dimension scores.

A.1 SYSTEM PROMPT

```
You are an advanced AI model equipped with OCR and image processing
↪  capabilities, capable of analyzing visual elements in detail.
```

A.2 EVALUATION PROMPT

```
Your task is to assess two webpage images and output a score between 0
↪  and 10 for each of the following questions.
If the answer to a question is a definite YES, output a score of 10,
↪  signifying perfect similarity.
Conversely, a definite NO should yield a score of 0, indicating no
↪  similarity.
For answers that fall in between, assign a score accordingly, where a
↪  higher number indicates a greater degree of similarity. Only provide
↪  the numerical score for each question, without any additional text.
Example contexts are provided for clarity. Examples provide the idea,
↪  but you can output any number in 0-10
range accordingly. Only output a comma separated list containing 10
↪  numbers. DO NOT give score of 10 for any category unless otherwise
↪  the two images are identical.

Layout Consistency (Score: 0-10): Does the placement of headers,
↪  footers, and sidebars match in both webpages? (e.g., A score of 10
↪  for identical layouts, 5 for similar but not exact placements, and 0
↪  for completely different layouts.)
Element Alignment (Score: 0-10): Are elements like images, buttons, and
↪  text boxes aligned similarly on both pages? (e.g., A score of 10 for
↪  perfectly aligned elements, 6 for slight misalignments, and 0 for
↪  major misalignments.)
Proportional Accuracy (Score: 0-10): Do the sizes and aspect ratios of
↪  images, buttons, and text boxes appear consistent across both pages?
↪  (e.g., A score of 10 for exact proportions, 4 for noticeable size
↪  differences, and 0 for drastic inconsistencies.)
Visual Harmony (Score: 0-10): Do both webpages exhibit a similar level
↪  of visual harmony and balance in their design? (e.g., A score of 10
↪  for harmonious designs, 5 for some dissonance, and 0 for clashing
↪  designs.)
```

```
Color Scheme and Aesthetic Match (Score: 0-10): How closely do the color
↪  schemes of the two webpages align in terms of background and text
↪  colors? Evaluate the similarity in hues, saturation, and overall
↪  color aesthetics. (e.g., A score of 10 for perfectly matching color
↪  schemes, including identical hues and saturation levels, 6 for
↪  similar color palettes with minor variations, and 0 for starkly
↪  different color schemes that create entirely different visual
↪  impacts.)
Aesthetic Resemblance (Score: 0-10): Is the overall aesthetic appeal
↪  (modern, minimalistic, traditional, etc.) similar on both pages?
↪  (e.g., A score of 10 for identical aesthetics, 4 for somewhat
↪  similar but distinguishable styles, and 0 for completely different
↪  aesthetics.)

Font Characteristics and Consistency (Score: 0-10): Assess the degree of
↪  consistency in font attributes across both webpages. This includes
↪  not only the font type and size but also the nuances of font style
↪  (italic, bold) and weight (light, regular, bold). (e.g., A score of
↪  10 for complete uniformity in font type, size, style, and weight
↪  across both pages, 5 for consistency in font type and size but
↪  variations in style or weight, and 0 for wide disparities in font
↪  type, size, style, or weight, leading to a distinctly different
↪  textual appearance.)
Textual Content Match (Score: 0-10): Do the words and sentences match
↪  between the two webpages? (e.g., A score of 10 for identical text, 5
↪  for some similar paragraphs or sections, and 0 for completely
↪  different textual content.)
Numeric and Special Character Accuracy (Score: 0-10): Are numbers,
↪  dates, and special characters (like email addresses) consistent
↪  between the two pages? (e.g., A score of 10 for exact matches, 6 for
↪  minor discrepancies, and 0 for major differences.)

User Interface Consistency (Score: 0-10): Do the user interface elements
↪  (like menus, buttons, and forms) on both pages share a similar
↪  design language and appearance? (e.g., A score of 10 for identical
↪  UI elements, 6 for slight design variations, and 0 for completely
↪  different UI designs.)
```

B Experimental Configurations

B.1 Training

Table 4: Training configurations used for both policy backbones.

Configuration	Value
Rollout group size	8
Training batch size	128
Mini-batch size	32
Learning rate	5×10^{-7}
Learning-rate schedule	Cosine decay
Warmup ratio	3%
KL coefficient	0.01
Maximum prompt length	16,384 tokens
Maximum response length	8,192 tokens
DARR EMA coefficient ρ	0.95
DARR temperature T	0.3

B.2 Sketch Synthesis and Inference

Sketch synthesis uses Canny edge detection (Canny, 1986) and headless Chromium rendering. For a text line of height h, the sinusoidal placeholder parameters adapt to the line height as specified

in Table 5. During inference, each model uses its default sampling parameters; we apply neither model-specific decoding adjustments nor retries for unsuccessful generations.

Table 5: Sketch-synthesis and inference configurations.

Configuration	Value
Canny thresholds	$(50, 150)$
Rendering viewport width	1280 pixels
Placeholder amplitude	$\max(2, 0.22h)$ pixels
Placeholder wavelength	$\max(8, 1.1h)$ pixels
Placeholder stroke width	$\min(\max(1, 0.12h), 8)$ pixels
Inference sampling parameters	Model defaults
Maximum response length	8,192 tokens
Model-specific decoding tuning	None
Retry on unsuccessful generation	No

C Statistical Significance Analysis

We assess the statistical reliability of the improvements from fixed-weight GRPO to DARR for both policy backbones using the paired per-sample overall scores underlying Tables 1 and 2. To obtain one judge-robust score per prediction, we first average the Qwen3.6-27B and GLM-4.6V overall scores for each sample with equal weight. Because multiple sketches can correspond to the same reference webpage, we use the reference webpage as the resampling cluster: SKETCHFACTORY-BENCH contains 414 sketch–webpage pairs from 348 webpages, and Sketch2Code contains 731 pairs from 484 webpages. For each benchmark, we apply a two-sided paired cluster sign-flip randomization test to the difference between the judge-averaged DARR and fixed-weight scores. We estimate 95% confidence intervals using a webpage-cluster bootstrap and apply Holm correction jointly over the four backbone–benchmark comparisons. The tests use 500,000 randomization samples and 100,000 bootstrap samples.

DARR has a positive mean advantage for both backbones on both benchmarks. After combining the two independent judges, all four improvements are statistically significant after Holm correction. For Qwen2.5-VL-7B, the judge-averaged gains are 0.26 points on SKETCHFACTORY-BENCH and 0.25 points on Sketch2Code. For Qwen3-VL-8B, the larger gain on SKETCHFACTORY-BENCH is reflected in its wider separation from zero, whereas the smaller Sketch2Code gain remains positive with a confidence interval whose lower endpoint is close to zero.

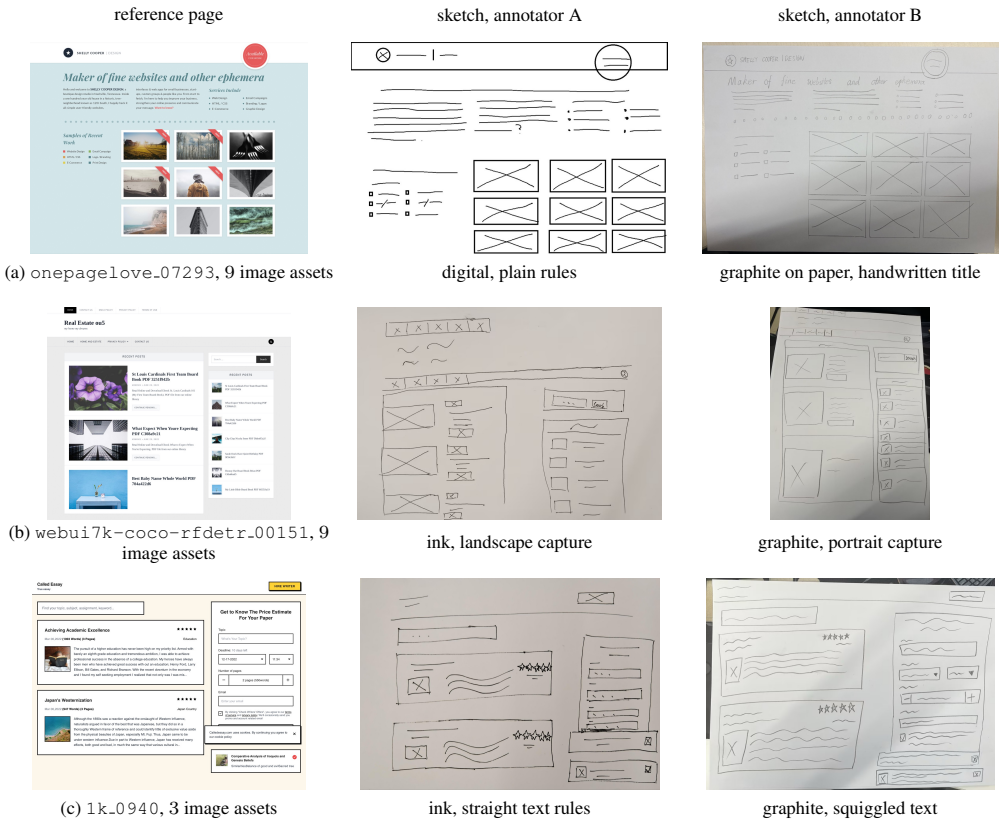
D IW-Bench Evaluation Implementation

The supplementary screenshot-to-code experiment uses the 1,702 samples in the downloaded IW-Bench V2 archive, rather than the 1,200 samples described for the original benchmark. All reported numbers are computed locally from the same sample manifest and evaluation implementation. The evaluator follows IW-Bench's element-matching and DOM-order subsequence definitions, with a matching threshold of 0.9. It compares element tags, direct text, attributes, styles, JavaScript bindings, and child tags. For scalable evaluation, reference DOM serialization is cached and wrapper removal uses the local fast simplification procedure. Serialization is capped at 5,000 elements; direct-text fields longer than 2,048 characters are represented by their first and last 1,024 characters for text similarity. The local serializer uses empty style dictionaries to reproduce the style contribution of the upstream serializer's numeric-key representation. The evaluator supports static HTML parsing as a fallback when browser serialization fails. Thus, the reported comparisons use a common local adaptation of the benchmark evaluator.

The original archived evaluation covers 27 models, each with 1,702 scored samples; we additionally evaluate the two DARR checkpoints using the same generation and metric protocol. We verify the aggregate scores by retaining the latest record for each sample ID and recomputing the arithmetic mean from the per-sample records. The main transfer table reports open-source base models in the 7B–8B range together with the Base, fixed-weight GRPO, and DARR variants of our two policy

Table 6: Paired significance analysis of DARR versus fixed-weight GRPO after averaging the two judges' per-sample scores. Δ denotes the mean paired improvement (DARR minus Fixed). Confidence intervals and tests cluster sketches by their reference webpage. p_{Holm} is corrected jointly over all four comparisons.

Backbone	Benchmark	Fixed	DARR	Δ	95% CI	p_{Holm}
Qwen2.5-VL-7B	SKETCHFACTORY-BENCH	5.66	5.91	+0.26	[0.13, 0.39]	< 0.001
	Sketch2Code	6.36	6.60	+0.25	[0.16, 0.33]	< 0.001
Qwen3-VL-8B	SKETCHFACTORY-BENCH	6.04	6.39	+0.35	[0.15, 0.55]	0.0015
	Sketch2Code	6.73	6.82	+0.09	[0.01, 0.17]	0.023



(a) `onepagelove_07293`, 9 image assets

(b) `webui7k-coco-rfdetr_00151`, 9 image assets

(c) `1k_0940`, 3 image assets

Figure 5: Three doubly annotated pages from SKETCHFACTORY-BENCH. Each row shows one reference page and the two independent sketches drawn for it. The pairs differ in medium, in capture geometry, and in how much interface detail the annotator commits: in (a) one annotator works in a drawing tool while the other writes the page title out by hand; in (b) the same layout is captured once in landscape and once in portrait; in (c) one annotator denotes text with straight rules and the other with squiggles, and only the second draws the quantity stepper and the dropdown carets. Every reference page keeps its original image assets, so a crossed box in the sketch has to be bound to one of several distinct files.

backbones. All six Qwen variants have complete predictions and metric records for all 1,702 samples. The experiment evaluates whether capabilities learned through sketch-to-code training transfer to screenshot-conditioned webpage reconstruction.

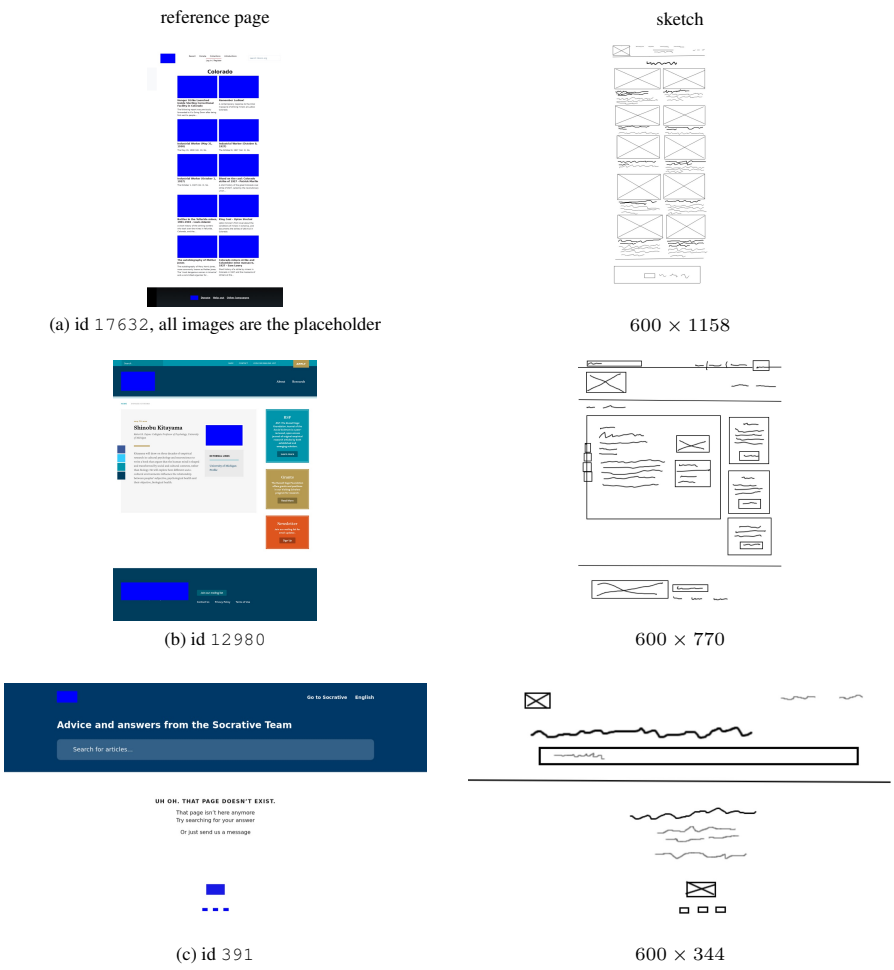


Figure 6: Three samples from Sketch2Code. Every sketch is delivered on a uniform 600 px-wide canvas with axis-aligned boxes and no capture artefacts. In the reference pages, each image is replaced by the same blue `rick.jpg` placeholder, a convention inherited from Si et al. (2025), so the target code repeats one fixed path instead of naming distinct asset files.

E Benchmark Comparison with Sketch2Code

This appendix compares the inputs of SketchFactory-Bench with those of Sketch2Code (Li et al., 2025a), the external benchmark used in Section 5.2. Figures 5 and 6 show three samples from each benchmark.

Sketch appearance. Our sketches are collected as image files rather than through a drawing interface, which leaves both the medium and the capture to the annotator. The photographs carry desk backgrounds, oblique lighting, and shadow gradients, and at least one page was photographed upside down, while the software-drawn sketches include one laid out on a dark canvas. Resolution is correspondingly unconstrained, spanning 540 to 9411 px in width over 141 distinct values, with 63% portrait and 37% landscape. Sketch2Code instead presents every one of its 731 sketches at exactly 600 px wide. The drawing vocabulary varies as well: both benchmarks mark images with crossed boxes, but our annotators denote text by straight rules, sinusoidal squiggles, or literal handwritten words, and draw interface affordances such as search fields and magnifier icons at their own discretion. Because 66 of our pages carry two independent sketches, this variation is observable within a fixed layout rather than confounded with it.

Asset references. Sketch2Code replaces every image in its reference pages with a single placeholder file, leaving 96.1% of its 484 pages with no genuine image reference; a policy therefore reproduces one constant path wherever an image belongs. In SKETCHFACTORY-BENCH, 81.0% of pages retain their original assets, averaging 4.54 per page, and the sketch gives only the position and rough aspect ratio of each one. Recovering the page therefore requires choosing among several concrete files, which is closer to the conditions under which a prototype is implemented from a drawing.