w1: VISUALLY GROUNDED CODE IMPROVES SCREENSHOT-TO-HTML GENERATION

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper introduces w1, a very competitive vision-language model (VLM) for converting webpage screenshots into HTML code. We find this task bottlenecked by the quality of supervision targets: code of real webpages contains many tokens, such as invisible elements, unused styles, redundant nesting, and media source strings, which are not visible from the webpage screenshot. It is not possible for a VLM to predict such code from the screenshot. To address this problem, we propose a data pipeline that removes the invisible code while preserving the rendering results. Specifically, our pipeline parses the HTML, matches each DOM element with its computed CSS, removes code without visible effect, canonicalizes render-equivalent structure, and replaces media sources with placeholders. The result is the WebGround-3M dataset. It has 3M screenshot-HTML pairs, where the code grounds in the screenshot. In this dataset, supervision targets are 30% shorter on average while rendering very similar to the original screenshots. On WebGround-3M, fine-tuning a VLM demonstrates an effective improving trend. Moreover, from WebGround-3M, we curate 200k synthetic preference pairs targeting at syntactic repetition and semantic perturbation; direct preference optimization (DPO) using these preference pairs adds further performance gains. Across six public benchmarks, w1 achieves highly competitive code generation performance. In particular, on benchmarks evaluated using GPT scores and the Design2Code metric, w1-9B outperforms zero-shot GPT5 and Gemini-3.5-flash, and models trained on WebSight-v1/v2, VinciCoder, MCD, WebCode2M, and Web2Code.

1 INTRODUCTION

When training vision-language (or wider computer vision) models, the underlying assumption is that the training ground truth, or target, can be reflected or inferred from the input image. In other words, if we require a model to generate an output that it does not see in or infer from an image, training gradient would not make sense, which would compromise model effectiveness.

This paper considers the task of screenshot-to-HTML generation. In this task, given a webpage screenshot as input, a vision language model (VLM) should output syntactically valid HTML and cascading style sheets (CSS) that render the same visual results as the input, in aspects like page layout, visual hierarchy, text content, colors, and component structure.

While the assumption of input-target consistency appears to be an 'easy peasy' reality in computer vision, it is non-trivial for screenshot-to-HTML generation, especially for real-world webpages. Some code produces no visible content, including document metadata, or document object model (DOM) and CSS for inactive interaction states (Appendix A). Other code, such as the media reference in `<img src="hero.png">`, refers to visible content but cannot be inferred from its appearance: the same image can be hosted at different addresses. These tokens are valid implementation details, yet are not visually grounded in the screenshot. By requesting the VLM to generate code which is not visually grounded in the input, it create a task that is infeasible for the VLM to determine.

To address these issues, we introduce a data pipeline that removes visually ungrounded tokens from HTML targets while preserving high visual similarity from their rendered appearance. Given a screenshot–HTML pair, we use heuristics and browser-computed styles and geometry to identify render-irrelevant content in the DOM and CSS. We then remove such render-irrelevant content, which does not affect rendering outcomes. For media URLs, we replace the original references with

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.

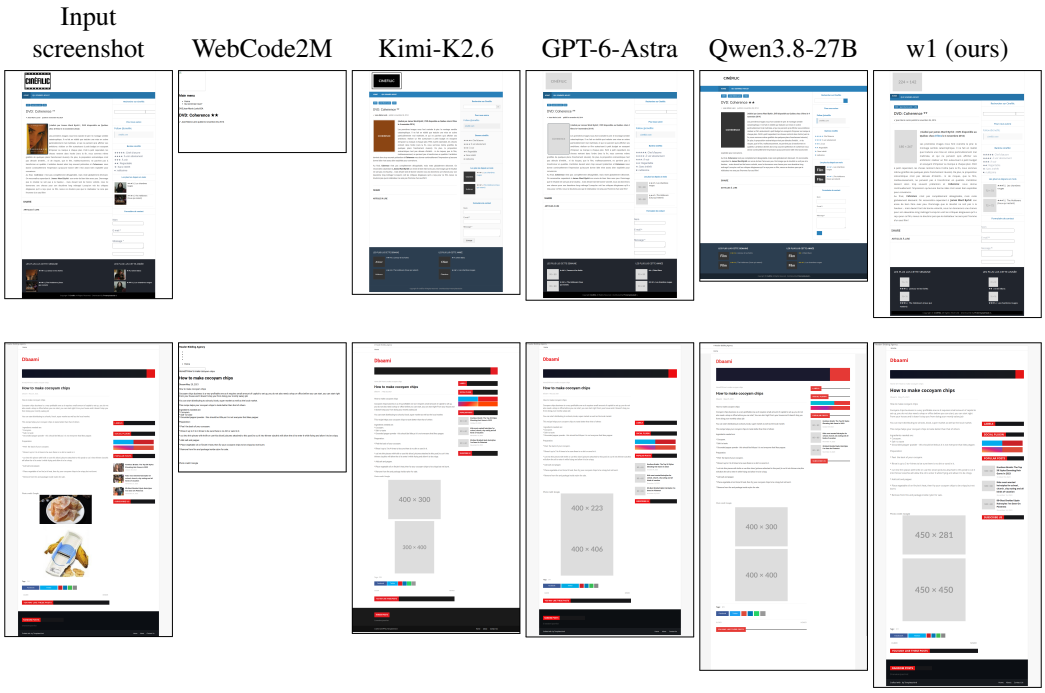


Figure 1: **Qualitative comparison.** From left to right: reference screenshots and rendered outputs from Qwen3-VL-4B-Instruct fine-tuned on WebCode2M, Kimi-K2.6, GPT-6-Astra, Qwen3.8-27B, and w1-4B, respectively. Note that the frontier models create colored images similar to the source, which would increase their similarity scores.

placeholder links that encode the displayed media size, so the VLM predicts media placement and size instead of predicting arbitrary asset addresses. To construct the corresponding input screenshots, we render the cleaned HTML with randomly sampled natural images in these media slots, while retaining placeholder links in the supervision targets. We apply this pipeline to WebCode2M (Gui et al., 2025a)[1], yielding **WebGround-3M**. Compared with WebCode2M, WebGround-3M has approximately 30% fewer target tokens on average, while screenshot similarity is kept high.

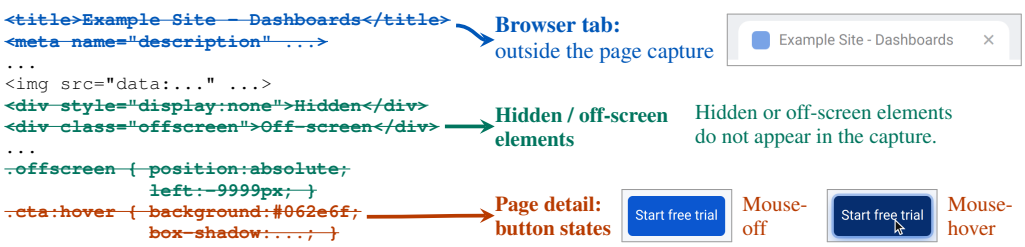
Training on WebGround-3M yields **w1**, a strong VLM for screenshot-to-HTML generation. There are two training stages. First, we perform supervised fine-tuning (SFT) of a base VLM model using screenshot–HTML pairs in WebGround-3M. Because WebGround-3M is clean, training signals are more effective; because the code targets are much shorter, training to generate such code is much easier. As a result, the performance of w1 improves consistently with increasing SFT data scale.

In the second stage, we use direct preference optimization (DPO) to align w1 with human preference. We address two common failure patterns: code repetition and visual inconsistency (*e.g.*, drift in color and font size away from the input screenshot). We randomly choose a subset of WebGround-3M as positive samples, from which we synthesize negative samples containing the two failure patterns. We show DPO adds further improvement at each data scale.

We evaluate w1 on six public benchmarks and reports very competitive code generation performance. For example, on the CC-Hard benchmark, w1 reaches 93% Design2Code score, outperforming zero-shot GPT-5.6-Sol (92.1%) and substantially exceeding models trained on WebSight, Vinci-coder, MCD, and Web2Code. We summarize the main points below.

- We identify that real webpage code contains substantial redundant content which is not visually grounded in the screenshot. This is a major bottleneck in the screenshot-to-HTML data quality.
- We propose a data pipeline that removes such noisy code, resulting in the WebGround-3M dataset, the code targets of which are cleaner and shorter, while preserving rendering equivalence.

[1]Despite its name, WebCode2M contains approximately 3.17 million screenshot–HTML pairs.



(a) Code without static visual effect: metadata, hidden/off-viewport elements, and inactive hover styling.



Original webpage with an embedded logo | Ours: Placeholder-link target $\hat{h}$ | Ours: SFT input $\hat{x} = \mathcal{R}(\hat{h})$

(b) The target code uses a size-matched placeholder link for the logo; a COCO image replaces the logo in the SFT input.

Figure 2: **Examples of visually ungrounded code.** (a) Code with no static visual effect. (b) Logo references that cannot be inferred from pixels, replaced by a size-matched placeholder in the target HTML and a COCO image in the SFT input (right).

- We perform supervised fine-tuning and show that WebGround-3M leads to better SFT results than the original WebCode2M dataset.
- We construct 200k synthetic preference pairs from WebGround-3M targeting repetition and visual inconsistency, use DPO for post-training, and demonstrate further improvement.
- The resulting w1 model has very competitive performance on a wide range of benchmarks.

2 Proposed Data Pipeline for Two Stage Training

Overview. Given an HTML document h, we render a reference screenshot x. Section 2.1 presents cases where part of h is not visible in x. In Section 2.2, we present the proposed data pipeline which removes tokens not visually grounded in x and obtains a visible HTML target $\hat{h}$ with placeholder links; the input $\hat{x}$ is rendered from a copy with embedded COCO images (Lin et al., 2014). The $\hat{x}$-$\hat{h}$ pairs are used for supervised fine-tuning (SFT). In Section 2.3, we show two failure modes after SFT and turn them into a preference dataset: we corrupt $\hat{h}$ with these failures to create negatives for direct preference optimization (DPO).

2.1 Visually Grounded and Ungrounded HTML

An HTML document combines *structure code*, the HTML markup and CSS rules that define visual structure, with *content*, *i.e.*, text and media placed inside that structure. At a fixed capture viewport, a browser renders h into the reference screenshot x:

$$x = \mathcal{R}(h), \tag{1}$$

where $\mathcal{R}$ is the renderer. Most HTML code and content can be derived from the rendering results, such as font size, line spacing, font color, etc. Some cannot be grounded visually in real-world webpages, and we identify two main types of such code.

Code with no static visual effect. Fig. 2a illustrates a `:hover` rule that takes effect only under user interaction, a state that static captures miss (Mesbah et al., 2012); removing it leaves the static screenshot unchanged. Appendix A provides a complete cleaning taxonomy and examples.

Code that links to or embeds media content. Webpages can reference media through URLs, such as `<img src="hero.png">`, or embed it directly through data URLs

Table 1: **Comparison of training sets.** On a per-page basis, we compare number of code tokens (Qwen3 tokenizer), number of code characters excluding whitespace, and number of DOM elements and their depth. *AI generated* indicates whether HTML is generated by AI (Table 4). WebGround-3M is cleaned version of WebCode2M and has much fewer tokens, characters, DOM elements and decreased DOM depth. Percentages indicate reductions relative to WebCode2M.

Dataset			Code		DOM		Screenshot
Corpus	AI-generated	Pages	Tokens	Chars	Elements	Depth	$W \times H$ (px)
MCD (Jiang et al., 2026)	Yes	200,061	582	1,356	21.5	5.0	$1,280 \times 817$
VinciCode (Zhao et al., 2025a)	Partly	263,338	1,373	3,627	54.6	7.0	$1,284 \times 943$
Web2Code (Yun et al., 2024)	Partly	806,604	422	1,058	19.9	5.5	$1,321 \times 881$
WebSight-v1 (Laurençon et al., 2024)	Yes	822,987	410	1,024	19.3	5.5	$1,283 \times 869$
WebSight-v2 (Laurençon et al., 2024)	Yes	1,922,671	447	1,257	19.5	6.3	$2,560 \times 1,608$
WebCode2M (Gui et al., 2025a)	No	3,171,024	4,363	11,476	188.2	14.9	$1,295 \times 1,617$
WebGround-3M	No	3,171,024	3,063 -29.8%	6,252 -45.5%	132.3 -29.7%	12.1 -18.8%	$1,295 \times 1,597$

containing base64-encoded payloads, such as `<img src=" iVBORw0KGgoAAAANSUhEUgAA...">`. These references and payloads supply media that affects the rendered appearance, but the screenshot does not uniquely determine the original asset URL or encoded bytes. For example, the logo in Fig. 2b (left) reveals its appearance, position, and size, but not the source reference or payload used to embed it. Thus, the screenshot provides evidence for generating the media layout and styling, but not for recovering its original source representation.

2.2 Constructing WebGround-3M with Visually Grounded Code for SFT

This section introduces the data cleaning pipeline, which removes visually ungrounded HTML code from real-world screenshot-HTML pairs. We start from a noisy and real-world dataset WebCode2M (Gui et al., 2025a), whose released version contains 3.17 million webpages and an average of 188.2 DOM elements per page.

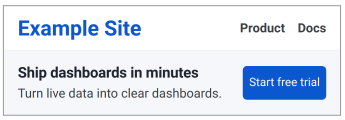
Step 1: Structural code cleaning. We apply heuristics that remove code without visual effect while preserving the rendered layout, style, and text. First, we remove scripts and non-layout metadata, which never affect the static rendering. Second, we prune invisible elements using browser-computed visibility and geometry at a 1280 px width, but keep invisible nodes that still shape the layout, such as spacers and flex/grid items. Finally, we simplify the remaining code by removing unused CSS and identifiers, normalizing tags, and collapsing redundant wrappers.

Step 2: Media code unification. We replace media reference code with a placeholder (refer Fig. 2b-middle). This placeholder matches the size of the media being replaced. For example, the placeholder code is `https://placehold.co/240x60` for an 240×60 px image.

SFT pair construction. After these two steps, we create a rendering copy $\tilde{h}$ of the cleaned HTML target $\hat{h}$ by replacing placeholder links with randomly sampled COCO images embedded as base64 data URLs, preserving media dimensions (Fig. 2b). Rendering this copy yields $\hat{x} = \mathcal{R}(\tilde{h})$, which we pair with $\hat{h}$ to form the SFT pair $(\hat{x}, \hat{h})$. Step 1 highly preserves webpage structure: the CLIP similarity between rendered webpage and original webpage is 98.81%. After Step 2, CLIP similarity becomes 84.48%. The lower score is because of random image replacement, but the layout remains highly similar (same as Step 1).

Corpus statistics. Across 3.17M pages, cleaning reduces average target token length by 29.8% from 4,363 to 3,063 tokens and DOM size by 29.7% to 132.3 elements. Note that webpages in WebCode2M and WebGround-3M are significantly more complex than MCD and Websight in terms of number of code tokens and DOM elements. Average page height changes little (1,617 to 1,597 px).

Training stage 1: Supervised fine-tuning (SFT) on WebGround-3M. We fine-tune a base VLM on WebGround-3M using token-level cross-entropy (Ouyang et al., 2022) to predict $\hat{h}$ from $\hat{x}$ (Section 4).



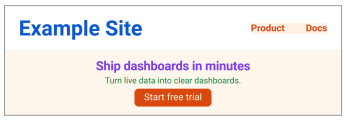

```
<div class="site-nav">
 <span>Product</span><span>Docs</span>
</div>
.site-nav { color:#3c4043;
  font-size:24px; gap:24px;
  background:transparent; }
...
```

(a) Chosen: cleaned HTML h^+




```
<div class="site-nav">
 <span>Product</span><span>Docs</span>
</div>
<div class="site-nav">
 <span>Product</span><span>Docs</span>
</div>
...
```

(b) Rejected: code repetition




```
<div class="site-nav">
 <span>Product</span><span>Docs</span>
</div>
.site-nav { color:#d9480f;
  font-size:18px; gap:40px;
  background:#fff1e6; }
...
```

(c) Rejected: visual inconsistency

Figure 3: **From SFT failure modes to preference pairs.** (a) Cleaned HTML forms the chosen sample h^+. Rejected samples h^- introduce (b) repeated navigation that omits subsequent content or (c) visual inconsistencies in colors, font sizes, and layout. Injected code edits are highlighted in red.

2.3 Constructing Preference Data from SFT Failure Modes for DPO

After SFT on WebGround-3M, we find there are two recurring failure modes: *code repetition* and *visual inconsistency* (for examples see Fig. 3). To mitigate this, we construct preference pairs that contrast the cleaned targets in WebGround-3M with random corruptions reflecting these errors.

Failure mode 1: Code repetition. The model repeatedly generates duplicate HTML elements or style blocks, exhausting the token budget before completing the page. This degenerate behavior is consistent with previous reports in autoregressive generation (Holtzman et al., 2019). We mimic this failure by duplicating tags, style blocks, or code fragments in the cleaned target.

Failure mode 2: Visual inconsistency. After SFT, generated HTML often renders incorrect colors, sizes, layout, or spacing. We construct negative samples by perturbing the corresponding CSS properties while keeping the DOM structure unchanged.

Preference pair construction. For each screenshot $\hat{x}$ sampled from WebGround-3M, we use its cleaned HTML $\hat{h}$ (also from WebGround-3M) as the preferred sample h^+. We apply one or both corruptions to obtain the rejected output h^-. This yields 200k preference pairs $(\hat{x}, h^+, h^-)$. Fig. 3 shows examples of clean code and its perturbations to mimic the two failure modes.

Training stage 2: Direct preference optimization (DPO). We further post-train the SFT model on these pairs using DPO (Rafailov et al., 2023). The objective increases the probability of generating clean samples over corrupted samples, mitigating code repetition and inconsistency.

3 Discussion

Ungrounded code viewed as input-conditional and systematic label noise. Whether a code token is noise depends on the screenshot it pairs with: grounded tokens are good signals, while ungrounded tokens are noise, which is input-conditional (Frénay & Verleysen, 2014). On the other hand, ungrounded code is also systematic instead of random. Random label noise corrupts examples independently. Differently, ungrounded code has systematic causes: it is a byproduct of building a working webpage and generally falls into two types described in Section 2.1. The systematic nature means that nearly every webpage contains ungrounded code. It also makes cleaning possible: deterministic rules can be used to detect and remove them (Section 2.2).

Why corrupted negatives rather than online-sampled ones? There are four main reasons. First, we have a clear picture of two types of failure modes, both of which can be easily mimicked by code perturbation. Online policy update can be slow to capture these cases. Second, online rollout requires generating thousands of tokens and rendering in the browser, which is much more expensive than off-line sample preparation. Third, it is non-trivial for online rollout to produce perfect positive samples because of the difficulty of the task. In comparison, off-line preparation of positive samples is efficient. Last, reward may not be reliable, because visual similarity measurement is not perfect. For example, some models generate SVGs or colorful placeholders that roughly approximate embedded images (Fig. 1). These increase visual similarity scores in screenshot-to-code benchmarks but do not provide faithful replacements for the original image assets in real-world web development.

Table 2: **Method comparisons.** We report Web2Code GPT-judge score ('GPT', 0–10) and Design2Code score ('D2C', 0–100) for each benchmark. Higher is better. 'Avg' averages the six benchmarks. In Block 1, we benchmark close-source frontier models. In Block 2, we benchmark open-source large models. In Block 3, we compare various existing training sets by fine-tuning Qwen3-VL-4B-Instruct on them. In Block 4, we compare three versions of w1 with their base models. Green indicates gains and red declines. Among open-source models (including corpus-trained models and w1), the best score per column is **bold** and the second-best is underlined.

Model	CC-Hard		D2C-All		D2C-Hard		WC-S		WC-M		WC-L		Avg	
	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C
Proprietary models														
GPT-6-Astra	8.59	94.2	9.49	95.3	8.64	94.7	8.76	92.2	8.56	93.6	8.23	93.1	8.71	93.8
GPT-5	6.88	88.2	7.84	91.3	7.09	83.7	6.97	88.3	6.71	87.3	6.57	88.1	7.01	87.8
Gemini-3.7-flash	7.54	88.4	8.27	92.2	7.77	90.5	6.99	83.4	6.49	79.7	6.56	80.6	7.27	85.8
Gemini-3.5-flash	7.36	86.6	8.24	91.3	7.79	89.9	6.60	80.7	6.21	78.1	6.33	80.5	7.09	84.5
Open-source large models														
Kimi-K2.6	7.64	91.9	**8.46**	**93.0**	7.36	89.5	**7.73**	90.2	7.49	91.3	**7.41**	91.3	**7.68**	**91.2**
MiMo-V2.5	7.25	90.0	7.96	91.6	7.34	87.3	7.23	89.0	7.03	90.1	6.63	88.0	7.24	89.3
MiniMax-M3	7.48	91.2	8.20	93.0	7.04	86.7	7.37	90.7	7.13	90.1	7.09	90.2	7.38	90.3
Qwen3.5-27B	6.56	85.0	7.63	91.2	6.78	87.0	7.18	87.6	6.47	85.9	6.30	86.0	6.82	87.1
Qwen3.6-27B	7.03	88.7	7.89	91.5	6.64	84.1	6.59	82.5	6.10	83.0	6.61	86.7	6.81	86.1
Qwen3.8-27B	7.66	89.9	8.38	92.6	**7.61**	**89.7**	7.61	88.5	7.25	88.7	7.18	88.9	7.62	89.7
Training-data comparison · Qwen3-VL-4B-Instruct														
Qwen3-VL-4B (base)	0.77	22.7	4.20	59.2	2.75	43.7	3.57	53.1	2.68	45.7	1.71	33.3	2.61	42.9
MCD	5.45	83.4	6.27	88.1	5.57	83.4	6.06	84.2	5.71	84.9	5.25	82.9	5.72	84.5
VinciCode	2.61	64.6	3.27	70.9	2.20	58.4	3.88	75.1	4.11	78.9	3.03	70.0	3.18	69.7
Web2Code	1.23	32.9	4.58	66.6	2.99	54.9	4.41	68.9	3.39	59.2	2.60	51.2	3.20	55.6
WebSight-v1	1.66	35.6	4.54	65.5	3.42	57.2	4.31	66.8	3.04	53.2	2.45	49.5	3.24	54.6
WebSight-v2	4.42	72.7	5.30	78.1	4.32	75.9	4.67	76.8	4.52	79.4	4.21	76.5	4.57	76.6
WebCode2M	5.56	83.4	4.73	79.3	2.95	65.0	6.05	86.0	6.90	92.4	5.88	87.2	5.34	82.2
WebGround-3M · w1-SFT	6.88 +1.32	90.3 +6.9	5.62 +0.89	83.8 +4.5	4.72 +1.77	77.2 +12.2	6.82 +0.77	87.4 +1.4	6.73 −0.17	91.0 −1.4	6.37 +0.49	88.9 +1.7	6.19 +0.85	86.4 +4.2
Matched model sizes: base → w1 (ours)														
Qwen3.5-0.8B (base)	2.51	51.0	3.22	55.1	2.65	48.8	2.47	46.7	2.82	52.1	3.13	53.8	2.80	51.3
w1-0.8B-DPO	6.71 +4.20	88.8 +37.8	6.11 +2.89	88.2 +33.1	5.33 +2.68	86.3 +37.5	6.65 +4.18	88.9 +42.2	6.94 +4.12	91.5 +39.4	6.40 +3.27	90.0 +36.2	6.36 +3.56	88.9 +37.6
Qwen3.5-4B (base)	4.63	65.6	6.27	80.4	4.63	63.9	4.97	68.8	4.67	68.2	5.09	71.8	5.04	69.8
w1-4B-DPO	7.48 +2.85	92.4 +26.8	6.93 +0.66	90.4 +10.0	6.12 +1.49	86.2 +22.3	7.38 +2.41	91.1 +22.3	7.41 +2.74	92.9 +24.7	7.03 +1.94	**93.0** +21.2	7.06 +2.02	91.0 +21.2
Qwen3.5-9B (base)	6.54	85.4	7.28	88.8	6.66	84.3	6.59	84.8	6.14	82.9	6.12	85.0	6.55	85.2
w1-9B-DPO	**7.69** +1.15	**93.0** +7.6	7.06 −0.22	90.2 +1.4	6.02 −0.64	86.1 +1.8	7.52 +0.93	**91.6** +6.8	**7.49** +1.35	**93.0** +10.1	7.02 +0.90	92.9 +7.9	7.13 +0.58	91.1 +5.9

Light blue cells indicate scores matching or exceeding the best other listed open-source model in that column.

Real-world webpages versus synthetic webpages. Real-world pages are abundant in scale, can be obtained cheaply, and offer real-world diversity. We aim to improve model capacity using such native data; in comparison, using synthetic webpages is a way of distilling frontier models.

Limitation. Our dataset is still limited in scale. Our future work is to further scale the data. Moreover, our data cleaning method is useful for now but may not be optimal, because data cleaning does not have ground truth. Further improvement will be useful.

4 Experiments

4.1 Datasets and Evaluation Protocols

Training data. For SFT, we use WebGround-3M pairs (Section 2.2) ranging from 100k pairs to the full set of 3.17M pairs to study the effect of data scaling. For DPO, we use the 200k synthetic preference pairs (Section 2.3). We also fine-tune the same base model separately on other existing

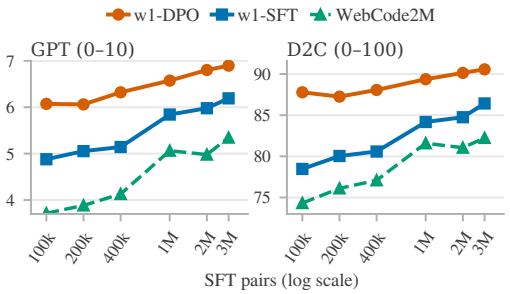


Figure 4: **Effect of WebGround-3M data scale.** We draw mean GPT and D2C scores across six benchmarks, with data volume on a logarithmic axis.

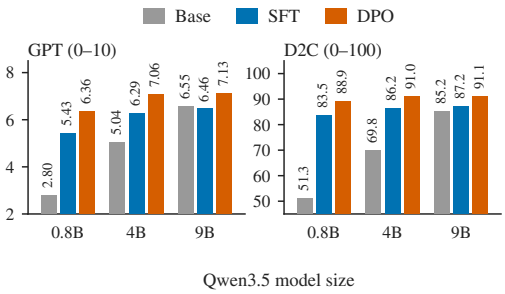


Figure 5: **Effect of model size.** Mean GPT and D2C scores for Qwen3.5 (0.8B, 4B, and 9B). For each mode size, we report performance of the base model, SFT model, and DPO model.

datasets: MCD (Jiang et al., 2026), VinciCode (Zhao et al., 2025a), Web2Code (Yun et al., 2024), WebSight-v1/v2 (Laurençon et al., 2024), and WebCode2M Gui et al. (2025a).

Benchmarks. We evaluate six public benchmarks of screenshot-to-html tasks. CC-Hard (Gui et al., 2025b) contains 128 pages sampled from Common Crawl, which has complex layouts and deeply nested DOM structure. Design2Code (Si et al., 2025) provides 2 test sets: the original set of 484 real-world webpages and an additional hard set of 80 webpages. WebCode2M (Gui et al., 2025a) provides three test sets: Short, Mid, and Long, which are grouped by target code length. Detailed statistics for all six benchmarks are provided in Appendix C.

Evaluation protocols. We render predicted HTML in headless Chromium at a 1280-pixel viewport width and evaluate full-page captures against input screenshots. (1) Web2Code GPT (Yun et al., 2024) uses GPT-5.4-nano to assess ten criteria across four categories: structure and alignment, color and aesthetics, text and content consistency, and interface and interactivity. The overall score averages the four category scores, each ranging from 0 to 10. (2) Design2Code (Si et al., 2025) averages five sub-scores: block matching, text similarity, position and text color agreement between matched visual blocks, and CLIP-based high-level visual similarity.

4.2 Implementation Details

Our proposed w1 series uses Qwen3.5 backbones (0.8B, 4B, and 9B), trained with full-parameter, cross-entropy SFT (Ouyang et al., 2022) on WebGround-3M, then vanilla DPO on 200k synthetic preference pairs. SFT uses learning rates of 2×10^{-6} for the vision encoder and 2×10^{-5} for the aligner and language model, 5% warmup, cosine decay to 1% of the peak rate, and weight decay 0.1. DPO ($\beta = 0.1$) freezes the vision encoder, uses the frozen SFT model as reference, and updates the aligner and language model at 5×10^{-8} with 10% warmup and cosine decay. Both stages run for one epoch with a global batch size of 32, 10,000-token contexts (packed for SFT), and at most 1,280 visual tokens per screenshot, using bfloat16, FlashAttention, and ZeRO-2 on four NVIDIA H100 GPUs. We train with ms-swift (Zhao et al., 2025b) and use vLLM (Kwon et al., 2023) for inference and evaluation, with greedy decoding, at most 8,192 new tokens, and repetition penalty 1.0.

4.3 Main Evaluation

Comparison with existing training sets. Table 2 reports higher mean GPT scores for WebGround-3M than for MCD, WebCode2M, or VinciCode. Specifically, w1-SFT achieves 6.19, compared with 5.72, 5.34, and 3.18, respectively.

Data cleaning improves generation quality. At every matched SFT scale, models trained on WebGround-3M outperform those trained on WebCode2M on mean D2C and GPT scores (Fig. 4), supporting the effectiveness of the cleaning pipeline.

Generation performance improves with more cleaned SFT data. Fig. 4 shows that SFT performance improves consistently as the training set grows from 100k to 3M examples. Across six

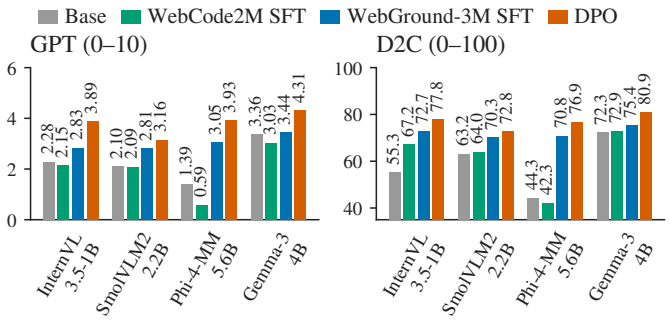


Figure 6: **Effectiveness of WebGround-3M for various backbones.** We report mean GPT and D2C scores across six benchmarks for each base model, after SFT on 200k WebCode2M or WebGround-3M pairs, and after DPO on 100k preference pairs following WebGround-3M SFT. w1 with or without DPO is consistently superior under each backbone.

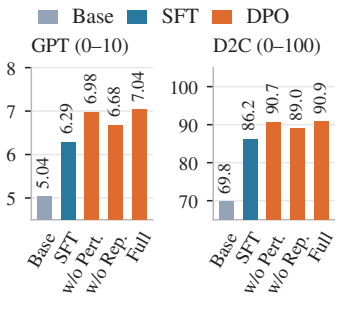


Figure 7: **Ablation of preference pair construction.** Mean GPT and D2C scores for Qwen3.5-4B: the base model, SFT, and DPO on 100k pairs using repetition, visual perturbation, or an equal mixture.

benchmarks, the mean GPT score rises from 4.88 to 6.19, and the mean D2C score from 78.5 to 86.4, with no clear sign of saturation.

DPO benefits at every SFT scale. DPO with the 200k preference pairs improves all SFT checkpoints from 100k to 3M training pairs (Fig. 4). Specifically, it adds 0.70–1.19 average GPT score and 4.1–9.3 average D2C score.

Comparison with frontier models. Results are summarized in Table 2. w1-4B-DPO achieves 91.0 average D2C across six benchmarks, close to Kimi-K2.6 (91.2) and above MiniMax-M3 (90.3) and Qwen3.8-27B (89.7). On CC-Hard, it reaches 92.4 D2C, outperforming all listed large open-source baselines; on WC-L, it is within 0.1 D2C of GPT-6-Astra (93.0 vs. 93.1). Its average GPT score also improves substantially over its base (5.04 to 7.06), compared with GPT-5 (7.01), although remaining below Kimi-K2.6 (7.68) and GPT-6-Astra (8.71). w1-9B-DPO is slightly better than the 4B version. These results indicate that w1 is competitive.

Qualitative comparison. Fig. 1 compares full-page renderings from w1-4B-DPO with four comparison systems (Qwen3-VL-4B-Instruct fine-tuned on WebCode2M, Kimi-K2.6, GPT-6-Astra, and Qwen3.8-27B) on two CC-Hard pages, alongside the reference screenshots. More results are provided in Fig. 8 in Section D.

4.4 Further Analysis

Improvement with model size. We post-train Qwen3.5 at 0.8B, 4B, and 9B with WebGround-3M pairs for SFT and 200k preference pairs for DPO. Results are presented in Fig. 5. We report that the introduced training data effectively improves the performance of different base model sizes. The only slight drop happens at SFT on the 9B model, but DPO then improves it. We also find that our fine-tuning narrows the performance gap between different model sizes. For example, the average D2C score spans 51.3–85.2 before training but 88.9–91.1 after DPO.

Generalization across backbones. To evaluate whether the gains extend beyond Qwen models, we apply the same SFT–DPO recipe to InternVL3.5-1B, SmolVLM2-2.2B, Phi-4-multimodal (5.6B), and Gemma-3-4B, using 200k WebGround-3M pairs for SFT and 100k preference pairs for DPO. As shown in Fig. 6, SFT on WebGround-3M improves both mean GPT and D2C scores over all four base models, and DPO yields further gains on both metrics. At the same SFT data volume, WebGround-3M also outperforms WebCode2M on both metrics across all four backbones. SFT on WebCode2M lowers the mean GPT score relative to each base model, showing that additional training data alone does not guarantee improvement. Together, these results support the effectiveness of our data and training recipe across the evaluated model families.

Ablations of preference-pair corruptions. We ablate DPO data on 100k pairs, with repetition-only, perturbation-only, or mixed preference pairs (Fig. 7). Both single-corruption variants improve over

SFT; repetition gives larger gains than perturbation. The mixture has the highest mean scores, and we observe that removing either component leads to a drop in performance.

Computational cost. Data preparation runs on HPC systems equipped with 4th Gen Intel Xeon Scalable processors. We use 32 parallel workers for HTML cleaning pipeline and 40 workers for full-page screenshot capture in headless Chromium. Cleaning and screenshot capture take approximately 36 and 71 hours for 3.14M pages, respectively, totaling 107 hours. Post-training Qwen3.5-4B on four NVIDIA H100 GPUs takes 128 hours for SFT on 3M pairs and 29 hours for DPO on 200k preference pairs, with one epoch per stage.

5 Related Work

Screenshot-to-code generation. Screenshot parsing has since served as a pretraining objective for visual language understanding (Lee et al., 2023). Modern approaches fine-tune VLMs (Liu et al., 2023; Bai et al., 2025) on screenshot–HTML pairs: WebSight synthesizes pages with LLMs so that pairs are clean by construction (Laurençon et al., 2024). Web2Code converts crawled and synthetic pages into instruction data (Yun et al., 2024). Design2Code benchmarks frontier models such as GPT-4o (Hurst et al., 2024) on real pages (Si et al., 2025). Other works improve inference by decomposing the page into regions before generation (Wan et al., 2025), or improves training with automated compiler and vision feedback (Wu et al., 2024). This work uses real webpages and ground their code to improve supervision target.

Supervision quality and label noise. Learning with corrupted class labels is well studied (Frénay & Verleysen, 2014; Zhang et al., 2017; Song et al., 2022), and data curation for large models filters or prunes low-quality examples (Sorscher et al., 2022; Gunasekar et al., 2023). Ungrounded HTML fits neither template: the tokens are correct as code yet unpredictable from the input image, and they sit inside otherwise valuable examples, so example-level filtering would discard the page along with the noise. Our pipeline instead repairs the target, removing only the tokens the screenshot cannot determine and verifying that the rendering is preserved (Section 2.2). A related asymmetry appears in minified and obfuscated JavaScript (Collberg et al., 1997), where identifier names are unrecoverable from the shipped artifact and must be guessed from context (Bavishi et al., 2018; Tran et al., 2019). Our setting poses the same unrecoverability from pixels, but instead of guessing the hidden content, we remove it from supervision.

Preference optimization for generation failures. Maximum likelihood neither exposes a model to its own erroneous prefixes (Bengio et al., 2015) nor penalizes degenerate repetition (Holtzman et al., 2019), motivating objectives with explicit negative signals such as unlikelihood training (Welleck et al., 2020). RLHF (Ouyang et al., 2022) and DPO (Rafailov et al., 2023) obtain this signal from output comparisons, typically over sampled generations ranked by human or AI judges (Bai et al., 2022). We instead synthesize negative examples by introducing code repetition and CSS perturbations into cleaned HTML targets (Section 2.3).

6 Conclusion

This paper introduces **w1**, a vision-language model which has strong capability of converting webpage screenshots into HTML. The secret of w1 lies in the training data. Our key finding is that a considerable amount of real-world HTML code does not render anything visible in the static webpage. Such code, together with the media HTML code, *e.g.*, link for a video, cannot be inferred from the screenshot visually and compromises model training if used. Our data pipeline identifies such invisible code and removes them from the HTML target while keeping the rendering results mostly unchanged. The resulting dataset, **WebGround-3M**, allows for very effective VLM supervised fine-tuning and gives a nice improving curve. Moreover, from WebGround-3M we curate positive and negative preference pairs which target at our prior knowledge of limitations in HTML code generation. DPO post-training from these pairs further improves w1 generation performance, which is very competitive compared with baseline retrained with existing HTML datasets and various open-weight foundation models. In future work, we will further scale up our dataset to cover wider webpage distributions and investigate reinforcement learning for better post-training.

Ethics Statement

Our training data is derived from WebCode2M (Gui et al., 2025a), a publicly released corpus of screenshot–HTML pairs crawled from real-world webpages, which we use under the terms of its release. We do not crawl any new pages. Our cleaning pipeline (Section 2.2) removes visually ungrounded code such as scripts, tracking snippets, and embedded media links, and replaces media sources with placeholder links, which reduces the amount of third-party content and identifiers carried into the targets. Visible page text is retained, so the corpus may still contain names or other personal information that the original pages displayed publicly. The images inserted into media slots when rendering training inputs are sampled from COCO (Lin et al., 2014). A screenshot-to-HTML model could be misused to clone the appearance of existing websites, for example for phishing; this risk is shared with prior screenshot-to-code systems, and our models do not reproduce site-specific back-end logic or credentials. No human subjects were involved in this work.

Reproducibility Statement

Section 2.2 describes each step of the data cleaning pipeline that produces WebGround-3M, and Section A lists the DOM processing rules it applies. Section 2.3 describes how the preference pairs are constructed. Section 4.2 reports the backbones, optimization hyperparameters, training framework, hardware, and decoding settings for both SFT and DPO. The benchmarks and evaluation protocols are described in Section 4.1 and Section C, and Section D lists every value plotted in the figures. All backbones, benchmarks, and the source corpus are publicly available.

AI Use Statement

Large language models were used to polish the writing of this paper and to assist with code for data processing and plotting. All text and code were reviewed and verified by the authors, who take full responsibility for the content. As part of the evaluation methodology, an LLM (GPT-5.4-nano) serves as the judge for the GPT-judge scores, as described in Section 4.1. LLMs were not used for research ideation.

References

Shuai Bai, Keqin Chen, Xuejing Liu, et al. Qwen2.5-VL technical report. *arXiv preprint arXiv:2502.13923*, 2025.

Yuntao Bai, Saurav Kadavath, Sandipan Kundu, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

Rohan Bavishi, Michael Pradel, and Koushik Sen. Context2name: A deep learning-based approach to infer natural variable names from usage contexts. *arXiv preprint arXiv:1809.05193*, 2018.

Samy Bengio, Oriol Vinyals, Navdeep Jaitly, and Noam Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems*, volume 28, 2015. URL https://proceedings.neurips.cc/paper/2015/hash/e995f98d56967d946471af29d7bf99f1-Abstract.html.

Christian Collberg, Clark Thomborson, and Douglas Low. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, 1997.

Benoît Frénay and Michel Verleysen. Classification in the presence of label noise: A survey. *IEEE Transactions on Neural Networks and Learning Systems*, 25(5):845–869, 2014. doi: 10.1109/TNNLS.2013.2292894.

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)*, 2025a.

Yi Gui, Zhen Li, Zhongyi Zhang, Guohao Wang, Tianpeng Lv, Gaoyang Jiang, Yi Liu, Dongping Chen, Yao Wan, Hongyu Zhang, et al. LaTCoder: Converting webpage design to code with layout-as-thought. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2025b.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, et al. Textbooks are all you need. *arXiv preprint arXiv:2306.11644*, 2023.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*, 2019.

Aaron Hurst, Adam Lerer, Adam P. Goucher, et al. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

Lingjie Jiang, Shaohan Huang, Yixia Li, Guanhua Chen, Dongdong Zhang, Furu Wei, et al. Viscodex: Unified multimodal code generation via merging vision and coding models. In *International Conference on Learning Representations*, volume 2026, pp. 41476–41513, 2026.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023.

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.

Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning*, 2023.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conference on Computer Vision*, pp. 740–755. Springer, 2014.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. *Advances in neural information processing systems*, 36:34892–34916, 2023.

Ali Mesbah, Arie van Deursen, and Stefan Lenselink. Crawling ajax-based web applications through dynamic analysis of user interface state changes. *ACM Transactions on the Web*, 6(1):1–30, 2012. doi: 10.1145/2109205.2109208.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model. *Advances in neural information processing systems*, 36:53728–53741, 2023.

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 Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 3956–3974, 2025.

Hwanjun Song, Minseok Kim, Dongmin Park, Yooju Shin, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *IEEE transactions on neural networks and learning systems*, 34(11):8135–8153, 2022.

Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and Ari S. Morcos. Beyond neural scaling laws: Beating power law scaling via data pruning. *Advances in Neural Information Processing Systems*, 35, 2022.

Hieu Tran, Ngoc Tran, Son Nguyen, Hoan Nguyen, and Tien N Nguyen. Recovering variable names for minified code with usage contexts. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pp. 1165–1175. IEEE, 2019.

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

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *International Conference on Learning Representations*, 2020.

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 Conference of the North American Chapter of the Association for Computational Linguistics*, 2024.

Sukmin Yun, Haokun Lin, Rusiru Thushara, et al. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal LLMs. *Advances in Neural Information Processing Systems*, 37, 2024.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *International Conference on Learning Representations*, 2017.

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

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, et al. Swift: a scalable lightweight infrastructure for fine-tuning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 29733–29735, 2025b.

A HTML Cleaning Rules and Limitations

This appendix details the transformations used by the processing pipeline. In the implementation, `benchmarkProcessor.ts` reads each benchmark `index.html`, calls `processHtmlToOptimized` with browser-based dimension capture enabled, writes `screenshot.html`, optionally writes the self-contained `screenshot_embedded.html`, captures comparison screenshots, and records the transformation statistics in `processing_result.json`. The taxonomy below describes the source patterns that this processor removes, normalizes, or replaces before the resulting HTML is used as supervision. The before–after code blocks are schematic excerpts illustrating individual operations, not complete processor outputs; unrelated styles and document content are omitted. The rules aim to retain visible layout, style, and text, but do not guarantee pixel-equivalent rendering. In particular, removing scripts, embedded applications, or overlays can change visible content, and media normalization intentionally replaces asset appearance. These operations should be distinguished from pruning nodes that browser measurements identify as invisible in the captured state.

Execution and metadata code. Document metadata and runtime logic are not uniquely determined by a static screenshot. The processor removes these nodes before visual measurement. This pass also removes unsupported embedded content such as frames and canvas elements. Such elements can be visible; their removal is a heuristic simplification rather than a rendering-preserving transformation. In the schematic example below, the removed embedded elements are explicitly hidden.

Raw webpage source

```
<head>
  <title>Checkout - experiment B</title>
  <meta name="description" content="landing page copy">
  <meta name="viewport" content="width=device-width, initial-scale=1">
```

Processed type	Implementation signal	Cleaned supervision target
Execution and metadata	Non-layout head tags; body scripts, frames, canvas, legacy applets, `noscript`, body-level resource links, and AMP runtime elements.	Remove execution-related and unsupported content, which may include visible media or widgets; retain inline styles and layout-relevant metadata such as viewport settings.
Undefined CSS variables	`var(--name)` references whose custom property is never declared in the document.	Substitute the fallback value, or remove the invalid declaration when the value becomes empty or unparsable.
Off-screen or overlay DOM	Fixed/absolute boxes translated or offset outside the 1280 px capture width, or full-viewport fixed overlays.	Remove the subtree before browser measurement.
Empty invisible structure	Empty elements with no visible background, border, outline, shadow, pseudo-content, or layout-reserving dimension.	Remove the node unless it acts as a clearfix, flex/grid item, spacer, or parent of visible content.
Browser-measured invisible DOM	Chromium geometry, `checkVisibility()`, paint-order evidence, opacity, clipping, text rendering, and horizontal viewport bounds.	Prune invisible leaves bottom-up while retaining ancestors and layout-critical nodes.
Hidden layout boxes	Invisible nodes with meaningful width and height, e.g., `visibility:hidden` or `opacity:0` boxes that still reserve space.	Clear the hidden contents and write explicit measured dimensions to preserve layout without supervising unseen text or children.
Unused CSS and identifiers	PurgeCSS reachability, selector matching after DOM pruning, inline-style overrides, unused `id`/`class` references, and `@import` rules.	Keep only CSS and attributes that can still affect the cleaned DOM.
Semantic/custom wrappers	Semantic block wrappers, custom hyphenated elements, and single-child wrappers with neutral attributes or inheritable styles.	Normalize to `div`, rewrite selectors when needed, and collapse wrappers that add no visual constraint.
External visual assets	`img`, `video`, `audio`, `source`, `track`, and inline CSS background URLs whose contents come from external files.	Replace dataset-specific references with neutral placeholders sized from captured geometry; synthetic placeholder references do not preserve the source URL, path, filename, hash, or query.

Table 3: Implementation-aligned taxonomy of source patterns processed by the benchmark cleaning pipeline. Each category corresponds to an operation reflected in the per-page statistics emitted by `processing_result.json`.

```
  <link rel="icon" href="/favicon.ico">
  <link rel="canonical" href="https://example.com/a/b">
  <style>.hero { padding: 48px; }</style>
  <script src="https://www.googletagmanager.com/gtag/js"></script>
</head>
<body>
  <noscript><img src="/pixel.gif?campaign=42" width="1" height="1"></noscript>
  <iframe hidden src="https://maps.example.com/embed"></iframe>
  <canvas hidden id="runtime-chart"></canvas>
  <amp-sidebar hidden id="menu">...</amp-sidebar>
  <main class="hero">Visible page content</main>
</body>
```

Cleaned supervision target

```
<head>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>.hero { padding: 48px; }</style>
</head>
<body>
  <main class="hero">Visible page content</main>
</body>
```

Undefined CSS custom properties. Real pages often contain CSS variables created by build systems or theme switchers. When a variable is never declared in the page, the implementation resolves the fallback if one exists and drops declarations that become invalid.

Raw webpage source

```
<style>
  .card {
    color: var(--text-color, #222);
    border: 1px solid var(--missing-border);
    transform: translateX(var(--closed-offset, -120%));
  }
</style>
<div class="card">Account settings</div>
```

Cleaned supervision target

```
<style>
  .card {
    color: #222;
    transform: translateX(-120%);
  }
</style>
<div class="card">Account settings</div>
```

Off-screen and full-viewport overlay DOM. The static pass resolves CSS positions and transforms, then removes fixed or absolute elements that are entirely outside the capture width or are full-screen runtime overlays. Off-screen menus and accessibility jumps can be absent from the captured state. A full-screen overlay, however, may obscure visible page content; removing it changes the rendering. The example uses a hidden loading mask to isolate the case without a visible overlay.

Raw webpage source

```
<style>
  .loading-mask { display: none; position: fixed; inset: 0; width: 100vw; height: 100vh; }
  .drawer { position: fixed; left: -100vw; width: 320px; }
  .skip { position: absolute; left: -9999px; top: 0; }
</style>
<div class="loading-mask"></div>
<nav class="drawer">Mobile navigation</nav>
<a class="skip" href="#content">Skip to content</a>
<main id="content">Visible article</main>
```

Cleaned supervision target

```
<main id="content">Visible article</main>
```

Empty invisible structure. Many source documents include spacer wrappers, empty portals, and generated containers. The implementation removes empty nodes with no visible CSS contribution, while preserving special cases that affect layout, such as clearfixes, flex/grid items, pseudo-content, or percentage-height spacers next to absolutely positioned content.

Raw webpage source

```
<style>
  .empty { background: transparent; border: 0; box-shadow: none; }
  .modal-root { display: block; }
  .clear { clear: both; }
</style>
<div class="empty"></div>
<div id="modal-root" class="modal-root"></div>
<div class="outer"><div class="inner"><h1>Visible title</h1></div></div>
<div class="clear"></div>
```

Cleaned supervision target

```
<h1>Visible title</h1>
<div class="clear"></div>
```

Browser-measured invisible DOM. After the source-only passes, the processor stamps stable element identifiers, renders the page in Chromium, and captures geometry and visibility features for each body element. Elements hidden by active media queries, clipping, opacity, zero-area boxes, horizontal overflow, or missing paint evidence are pruned bottom-up unless they are required to preserve visible descendants or layout.

Raw webpage source

```
<style>
  .desktop-only { display: none; }
  .ghost { opacity: 0; }
  .clipped { clip-path: inset(100%); }
  .zero { width: 0; height: 0; overflow: hidden; }
  .off-x { transform: translateX(1400px); }
</style>
<section>
  <p class="desktop-only">Inactive responsive copy</p>
  <p class="ghost">Transparent tracking label</p>
  <p class="clipped">Clipped menu label</p>
  <div class="zero">Collapsed panel body</div>
  <aside class="off-x">Off-canvas state</aside>
  <p class="visible">Visible copy in the screenshot</p>
</section>
```

Cleaned supervision target

```
<section>
  <p class="visible">Visible copy in the screenshot</p>
</section>
```

Hidden layout boxes with content replacement. Some invisible nodes still reserve visible layout space through their measured width, height, margin, padding, float, or flex/grid behavior. Deleting such a node would shift neighboring content, so the processor keeps the box but clears unseen descendants and writes explicit dimensions from browser measurement.

Raw webpage source

```
<style>
  .ad-slot {
    width: 320px;
    height: 120px;
    margin: 24px 0;
    visibility: hidden;
  }
</style>
<article>
  <h2>Story heading</h2>
  <div class="ad-slot">
    <span>Unseen ad network text</span>
    <img src="https://ads.example.com/banner.png">
  </div>
  <p>Body paragraph after the reserved slot.</p>
</article>
```

Cleaned supervision target

```
<article>
  <h2>Story heading</h2>
  <div class="ad-slot" style="width: 320px; height: 120px;
       margin: 24px 0; visibility: hidden"></div>
  <p>Body paragraph after the reserved slot.</p>
</article>
```

CSS, selector, and attribute cleanup. Once DOM pruning has removed invisible subtrees, the implementation optimizes embedded CSS against the remaining document. It removes `@import` rules, unmatched selectors, hidden elements reached through computed `display:none`, redundant declarations, and `id`/`class` attributes no remaining CSS rule references.

Raw webpage source

```
<style>
  /* Assume no additional rules are needed from external stylesheets. */
  #runtime-hook { display: none; }
  .hero, .unused-card, .legacy-banner { color: #111; }
  .btn { display: block; padding: 12px 20px; }
</style>
<div id="runtime-hook" class="js-mounted"></div>
<section id="server-id-91" class="hero css-1a2b3c">
  <a class="btn analytics-click" href="#">Buy now</a>
</section>
```

Cleaned supervision target

```
<style>
  .hero { color: #111; }
  .btn { display: block; padding: 12px 20px; }
</style>
<section class="hero">
  <a class="btn" href="#">Buy now</a>
</section>
```

Semantic tags, custom elements, and neutral wrappers. Block-level semantic tags such as `section`, `article`, `header`, `footer`, `main`, `nav`, `aside`, and `hgroup` are normalized to `div` because they render like ordinary block containers in the static screenshot. If a tag name appears in CSS, the processor rewrites the selector to a compact class. Hyphenated custom elements are handled similarly, with an inline-display fix when needed. Neutral single-child wrappers are then collapsed.

Raw webpage source

```
<style>
  header { color: white; background: #111; }
  x-price { font-weight: 700; }
</style>
<div id="__next">
  <header>
    <div class="wrap"><h1>Store</h1></div>
  </header>
  <x-price>$19</x-price>
</div>
```

Cleaned supervision target

```
<style>
  .__hd { color: white; background: #111; }
  .__ce0 { font-weight: 700; }
</style>
<div class="__hd"><h1>Store</h1></div>
<div class="__ce0" style="display:inline">$19</div>
```

External asset normalization. The final target cannot require the model to predict dataset-specific file names, hashes, or the bytes of referenced image, video, or audio files. The processor therefore replaces external image and media URLs with neutral placeholders. The resulting placeholder reference is a synthetic normalization artifact: for images it is a `placehold.co` size URL such as `https://placehold.co/640x360`, which encodes the captured asset slot rather than the source domain, path, filename, hash, or query. Image and background placeholders are sized from captured browser geometry when available; very small images are removed as tracking pixels, and failed relative images keep a harmless # source.

Raw webpage source

```
<style>
  .hero { width: 640px; height: 360px;
          background-image: url("https://cdn.example.com/hero-9f3a2c.png"); }
</style>
<img class="photo" src="https://cdn.example.com/photo-a81d.jpg" width="640" height="360">
<img class="pixel" src="https://tracker.example.com/p.gif" width="1" height="1">
<img class="local" src="images/product.png">
<video src="https://cdn.example.com/demo.mp4"></video>
<track src="https://cdn.example.com/captions.vtt">
<div class="hero"></div>
```

Cleaned supervision target

```
<style>
  .hero { width: 640px; height: 360px;
          background-image: url("https://placehold.co/640x360"); }
</style>
<img class="photo" src="https://placehold.co/640x360" width="640" height="360">
<img class="pixel" width="1" height="1">
<img class="local" src="#">
<video src="data:video/mp4; base64,..."></video>
<track src="data:text/vtt; base64,...">
<div class="hero"></div>
```

For visualization, the benchmark processor additionally runs `embedPlaceholderImages` on `screenshot.html`. This creates `screenshot_embedded.html`, where each synthetic

`placehold.co` image URL is replaced by a resized data-URI image, allowing the processed page to be rendered and inspected without network access.

B Origin of the Training Corpora

Table 4 records, for every screenshot–HTML corpus compared in Table 2, whether its pages were produced by an AI synthesis pipeline or crawled from the live web. This expands the *AI-generated* column of Table 1; the basis for each entry is the data-construction section of the corresponding paper.

Table 4: Origin of the SFT corpora compared in Table 2. *AI-generated HTML* marks pages produced by an AI synthesis pipeline (Yes), a mix of synthesized and crawled pages (Partly), or real crawled pages (No).

Corpus	AI-generated HTML	Basis
MCD (Jiang et al., 2026)	Yes	GPT-4o designs 200k entirely new pages, using 560k Web2Code screenshots only as stylistic seeds; the pages are rendered and filtered automatically.
VinciCode (Zhao et al., 2025a)	Partly	Web-to-HTML data combines MCD with a filtered 60k subset of real WebCode2M pages and with Web2Code; the refinement split pairs each target with code drafted by a trained VLM.
Web2Code (Yun et al., 2024)	Partly	Crawled pages together with synthesized instruction data.
WebSight-v1 / v2 (Laurençon et al., 2024)	Yes	Pages synthesized by LLMs, so that screenshot–code pairs are clean by construction.
WebCode2M	No	Real crawled webpages, used with their original code.
WebGround-3M (**w1**)	No	Crawled HTML cleaned programmatically without teacher-model distillation; screenshots are re-rendered with COCO images in media slots (Section 2.2).

C Benchmark Statistics

Table 5 reports descriptive statistics for the six existing evaluation benchmarks introduced in Section 4.1.

Table 5: Benchmark statistics (per-page averages over the reference HTML). Code tokens are counted with the Qwen3 tokenizer; DOM elements and depth are parsed from the reference HTML; visible text is the number of rendered text characters; width and height are the dimensions of the full-page screenshot, captured at a 1280 px viewport (a page whose content overflows that viewport renders wider).

Benchmark	Pages	Code tokens	DOM elem.	DOM depth	Text chars	Width (px)	Height (px)
CC-Hard	128	7,520	274	16.0	2,226	1,284	1,954
Design2Code	484	25,243	158	12.5	1,414	1,280	1,482
Design2Code-Hard	80	30,945	251	10.3	3,930	1,280	3,249
WebCode2M-Short	256	1,721	81	9.9	1,915	1,284	1,213
WebCode2M-Mid	256	3,164	144	12.4	2,814	1,296	1,486
WebCode2M-Long	256	6,815	222	15.6	2,450	1,292	1,645

D Supplementary Experimental Results

This appendix reports per-benchmark results supporting the main experiments, followed by visual examples of reconstructed pages.

D.1 Computational resources

Data preparation runs on HPC systems equipped with 4th Gen Intel Xeon Scalable processors. We use 32 parallel workers for HTML cleaning pipeline and 40 workers for full-page screenshot capture in headless Chromium. Cleaning and screenshot capture take approximately 36 and 71 hours for 3.14M pages, respectively, totaling 107 hours. At the same throughput, these two stages would take approximately 3.4 hours for 100k pages and 102.2 hours for 3M pages. Post-training Qwen3.5-4B on four NVIDIA H100 GPUs takes 128 hours for SFT on 3M pairs and 29 hours for DPO on 200k preference pairs, with one epoch per stage.

D.2 Additional model comparisons

Table 6 extends the main comparison with systems not listed in Table 2. All entries use the same metrics as the main results. The additional systems show that GPT and D2C rankings need not agree: GPT-5.6-sol has the highest mean GPT score in this table (8.59), whereas GPT-5.5 has the highest mean D2C score (93.3). These comparisons supplement the main table without repeating its model rows.

D.3 Effect of training-data scale

Table 7 lists every value plotted in Figure 4: the per-benchmark GPT-judge and D2C scores, and their six-benchmark means, for the **w1**-SFT, **w1**-DPO, and WebCode2M checkpoints at each SFT scale from 100k to 3M pairs, in the layout of Table 2.

WebGround-3M SFT has higher reported means than WebCode2M SFT at every matched scale. At 3M pairs, the means are 6.19 versus 5.34 GPT and 86.4 versus 82.2 D2C; the advantage is not uniform across benchmarks, as WebCode2M retains higher scores on WebCode2M-Mid at this scale. DPO raises both means at every SFT scale, although the DPO curves are not strictly monotonic between 100k and 200k SFT pairs.

Table 6: **Additional proprietary and open-source systems.** Systems absent from Table 2, evaluated under the same protocol. GPT (0–10) and D2C (0–100) are higher-is-better scores; Avg is the unweighted mean across all six benchmarks. GPT scores use GPT-5.4-nano as judge. Best among the systems in this table is bold.

Model	CC-Hard		D2C-All		D2C-Hard		WC-S		WC-M		WC-L		**Avg**	
	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C
Proprietary models														
GPT-5.6-sol	**8.46**	93.6	**9.26**	94.7	**8.42**	93.6	**8.66**	**91.8**	**8.45**	**92.9**	**8.20**	92.1	**8.57**	93.1
GPT-5.5	8.33	**94.0**	9.20	**95.0**	8.38	**93.7**	8.48	91.8	8.28	92.9	7.91	**92.6**	8.43	**93.3**
GPT-4o	5.78	84.5	6.66	88.9	5.61	81.6	6.06	85.9	5.58	83.8	5.56	83.8	5.87	84.8
Gemini-3.1-pro	7.40	87.8	8.30	90.9	7.70	89.0	6.37	79.4	5.99	76.7	6.32	80.6	7.01	84.1
Open-source models														
Qwen3.5-122B-A10B	6.83	88.3	7.50	90.4	6.78	86.5	6.74	87.4	6.19	84.8	6.46	87.4	6.75	87.5
Qwen2.5-VL-72B	5.64	42.5	6.82	54.3	6.07	60.0	6.33	72.0	6.09	52.8	5.88	41.7	6.14	53.9
Qwen3-VL-32B	5.47	71.3	7.58	88.1	5.95	73.9	6.38	79.4	5.79	75.7	5.75	77.0	6.15	77.6
Qwen3-VL-8B	2.17	37.6	5.80	73.9	3.72	55.0	4.91	65.0	4.01	58.8	2.87	46.7	3.91	56.2

Table 7: **Values plotted in Figure 4.** Qwen3-VL-4B-Instruct after SFT on 100k–3M pairs, for the three curves of the main-text scaling figure: **w1**-SFT (cleaned WebGround-3M pairs), **w1**-DPO (the same checkpoints after DPO on the same 200k synthetic preference pairs, Section 2.3), and WebCode2M, the uncleaned corpus at matched scale. Each benchmark reports the Web2Code GPT-judge score (**GPT**, 0–10, judge `gpt-5.4-nano`) and the Design2Code score (**D2C**, 0–100); higher is better, and **Avg** is the unweighted mean over the six benchmarks (CC-Hard, D2C-All, D2C-Hard, and the WebCode2M Short/Mid/Long splits), i.e. the quantity on the vertical axes of Figure 4. Decoding with vLLM 0.19 (CUDA 13.0, repetition penalty 1.0). GPT scores use GPT-5.4-nano as judge. Best per column in **bold**.

Model	SFT pairs	CC-Hard		D2C-All		D2C-Hard		WC-S		WC-M		WC-L		**Avg**	
		GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C
w1-SFT	100k	5.34	81.3	4.72	78.7	3.64	66.9	5.33	80.2	5.40	83.0	4.86	80.7	4.88	78.5
w1-SFT	200k	5.84	85.6	4.81	78.0	3.35	65.0	5.51	82.7	5.64	85.9	5.18	83.0	5.05	80.0
w1-SFT	400k	5.60	85.2	4.94	79.3	3.28	64.2	5.93	84.4	5.78	86.4	5.32	84.0	5.14	80.6
w1-SFT	1M	6.45	88.0	5.37	81.6	4.15	71.6	6.42	86.9	6.56	88.8	6.11	88.1	5.84	84.2
w1-SFT	2M	6.73	89.3	5.53	82.5	4.45	73.0	6.41	86.3	6.60	89.3	6.16	88.2	5.98	84.7
w1-SFT	3M	6.88	90.3	5.62	83.8	4.72	77.2	6.82	87.4	6.73	91.0	6.37	88.9	6.19	86.4
w1-DPO	100k	6.45	90.4	5.98	87.3	5.10	82.2	6.48	88.1	6.46	89.7	5.97	89.1	6.07	87.8
w1-DPO	200k	6.45	90.0	5.98	86.6	4.99	80.2	6.48	88.6	6.57	89.2	5.91	88.8	6.06	87.3
w1-DPO	400k	6.93	89.2	6.16	88.3	5.15	82.0	6.78	89.3	6.68	90.2	6.22	89.4	6.32	88.1
w1-DPO	1M	6.95	91.4	6.26	88.4	5.50	83.3	7.02	90.2	7.11	91.9	6.60	91.0	6.57	89.4
w1-DPO	2M	7.33	**92.3**	6.60	89.9	5.64	84.5	**7.22**	90.6	7.20	92.0	6.82	91.5	6.80	90.1
w1-DPO	3M	**7.36**	92.0	**6.73**	**90.0**	**6.02**	**86.1**	7.17	**90.6**	**7.23**	**92.5**	**6.86**	**92.2**	**6.90**	**90.6**
WebCode2M	100k	3.18	71.6	3.32	71.7	3.21	69.3	4.27	76.3	4.66	81.5	3.64	75.6	3.71	74.3
WebCode2M	200k	3.76	75.5	3.67	74.8	2.35	64.8	4.69	79.4	4.78	83.0	4.05	79.1	3.88	76.1
WebCode2M	400k	4.10	76.2	3.86	76.7	2.47	62.9	5.10	82.2	5.05	85.1	4.20	79.5	4.13	77.1
WebCode2M	1M	4.90	81.2	4.24	77.3	3.18	67.3	5.86	86.2	6.57	92.0	5.63	85.5	5.06	81.6
WebCode2M	2M	4.87	80.8	4.34	76.9	2.84	66.6	6.06	88.1	6.49	90.7	5.28	83.1	4.98	81.0
WebCode2M	3M	5.56	83.4	4.73	79.3	2.95	65.0	6.05	86.0	6.90	92.4	5.88	87.2	5.34	82.2

D.4 Effect of preference-pair corruption

Table 8 lists every value plotted in Figure 7: the per-benchmark GPT-judge and D2C scores, and their six-benchmark means, of the untrained Qwen3.5-4B base model, the shared 3M-pair SFT checkpoint,

Table 8: **Values plotted in Figure 7.** The untrained Qwen3.5-4B base model, the same model after SFT on the full 3M WebGround-3M pairs (**w1**-SFT), and the SFT model after DPO on 100k preference pairs whose negatives are corrupted by code repetition only, by visual perturbation only, or by a 50/50 mix of both (Section 2.3). Each benchmark reports the Web2Code GPT-judge score (**GPT**, 0–10, judge `gpt-5.4-nano`) and the Design2Code score (**D2C**, 0–100); higher is better, and **Avg** is the unweighted mean over the six benchmarks (CC-Hard, D2C-All, D2C-Hard, and the WebCode2M Short/Mid/Long splits), i.e. the bar heights of Figure 7. Decoding with vLLM 0.19 (CUDA 13.0, repetition penalty 1.0). Best per column in **bold**.

Model	DPO negatives	CC-Hard		D2C-All		D2C-Hard		WC-S		WC-M		WC-L		**Avg**	
		GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C
Qwen3.5-4B	–	4.63	65.6	6.27	80.4	4.63	63.9	4.97	68.8	4.67	68.2	5.09	71.8	5.04	69.8
w1-SFT	–	6.67	88.3	6.00	84.7	4.48	74.1	6.82	88.1	7.13	91.6	6.61	90.6	6.29	86.2
w1-DPO	Repetition only (100k)	7.41	**92.2**	6.87	89.9	5.87	85.6	7.39	90.9	**7.39**	**93.3**	6.96	92.3	6.98	90.7
w1-DPO	Perturbation only (100k)	7.13	91.3	6.48	87.8	5.23	82.4	7.00	89.0	7.28	92.2	6.98	91.6	6.68	89.0
w1-DPO	Both, mixed (100k)	**7.43**	92.2	**6.88**	**90.2**	**6.12**	**86.2**	**7.41**	**91.0**	7.34	93.1	**7.05**	**92.9**	**7.04**	**90.9**

and the DPO checkpoints trained from it on 100k repetition-only, perturbation-only, and mixed preference pairs, in the layout of Table 2.

Both single-corruption variants improve the mean GPT and D2C scores over SFT. Repetition-only DPO scores 6.98 GPT and 90.7 D2C, compared with 6.68 and 89.0 for perturbation-only DPO. The mixture gives the highest means (7.04, 90.9), with a GPT margin of 0.06 over repetition alone; it does not lead on every individual benchmark metric.

D.5 Effect of model size

Table 9 lists every value plotted in Figure 5: the per-benchmark GPT-judge and D2C scores, and their six-benchmark means, of Qwen3.5-0.8B, Qwen3.5-4B, and Qwen3.5-9B before training, after SFT on the full 3M WebGround-3M pairs, and after the subsequent DPO on the 200k preference pairs, in the layout of Table 2.

DPO improves both mean metrics over SFT at all three model sizes. The 9B model is an exception to monotonic improvement from the base model through SFT: its mean GPT score falls from 6.55 to 6.46, before rising to 7.13 after DPO. DPO also narrows the D2C gap between 0.8B and 9B, from 33.9 points before training to 2.2 points afterward.

Table 9: **Values plotted in Figure 5.** For each Qwen3.5 size of the main-text model-size figure: the untrained model (Base), the model after SFT on the full 3M WebGround-3M pairs (**w1**-SFT), and the same model after DPO on the 200k synthetic preference pairs (**w1**-DPO, Section 2.3). Each benchmark reports the Web2Code GPT-judge score (**GPT**, 0–10, judge `gpt-5.4-nano`) and the Design2Code score (**D2C**, 0–100); higher is better, and **Avg** is the unweighted mean over the six benchmarks (CC-Hard, D2C-All, D2C-Hard, and the WebCode2M Short/Mid/Long splits), i.e. the bar heights of Figure 5. Decoding with vLLM 0.19 (CUDA 13.0, repetition penalty 1.0); GPT scores use GPT-5.4-nano as judge. Best per column within each model size in **bold**.

Model	Stage	CC-Hard		D2C-All		D2C-Hard		WC-S		WC-M		WC-L		**Avg**	
		GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C
Qwen3.5-0.8B	Base	2.51	51.0	3.22	55.1	2.65	48.8	2.47	46.7	2.82	52.1	3.13	53.8	2.80	51.3
	w1-SFT	5.95	86.5	5.00	81.9	3.58	73.1	5.87	84.8	6.27	88.0	5.92	86.8	5.43	83.5
	w1-DPO	**6.71**	**88.8**	**6.11**	**88.2**	**5.33**	**86.3**	**6.65**	**88.9**	**6.94**	**91.5**	**6.40**	**90.0**	**6.36**	**88.9**
Qwen3.5-4B	Base	4.63	65.6	6.27	80.4	4.63	63.9	4.97	68.8	4.67	68.2	5.09	71.8	5.04	69.8
	w1-SFT	6.67	88.3	6.00	84.7	4.48	74.1	6.82	88.1	7.13	91.6	6.61	90.6	6.29	86.2
	w1-DPO	**7.48**	**92.4**	**6.93**	**90.4**	**6.12**	**86.2**	**7.38**	**91.1**	**7.41**	**92.9**	**7.03**	**93.0**	**7.06**	**91.0**
Qwen3.5-9B	Base	6.54	85.4	**7.28**	88.8	**6.66**	84.3	6.59	84.8	6.14	82.9	6.12	85.0	6.55	85.2
	w1-SFT	7.03	89.0	6.03	84.2	4.75	78.0	6.97	89.9	7.19	90.9	6.78	91.1	6.46	87.2
	w1-DPO	**7.69**	**93.0**	7.06	**90.2**	6.02	**86.1**	**7.52**	**91.6**	**7.49**	**93.0**	**7.02**	**92.9**	**7.13**	**91.1**

D.6 Generalization across backbones

Table 10 gives the per-benchmark results behind Figure 6 for InternVL3.5-1B, SmolVLM2-2.2B, Phi-4-multimodal (5.6B), and Gemma-3-4B. Each backbone is evaluated before training, after SFT on 200k WebCode2M pairs, after SFT on 200k WebGround-3M pairs, and after a further DPO stage on 100k preference pairs initialized from the WebGround-3M SFT checkpoint. All reported conditions now have complete GPT and D2C evaluations across the six benchmarks. At matched SFT volume, WebGround-3M has higher reported means than WebCode2M for every backbone. DPO further adds 0.35–1.06 GPT and 2.5–6.1 D2C points over WebGround-3M SFT. WebCode2M SFT, however, does not improve every base model: its mean GPT score decreases for all four backbones, and Phi-4 also loses D2C performance.

Table 10: **Values plotted in Figure 6.** For each backbone of the main-text generalization figure: the untrained model (Base), models after SFT on 200k WebCode2M or WebGround-3M pairs, and the WebGround-3M SFT model after DPO on 100k synthetic preference pairs (DPO, Section 2.3). Each benchmark reports the Web2Code GPT-judge score (**GPT**, 0–10, judge `gpt-5.4-nano`) and the Design2Code score (**D2C**, 0–100); higher is better, and **Avg** is the unweighted mean over the six benchmarks (CC-Hard, D2C-All, D2C-Hard, and the WebCode2M Short/Mid/Long splits), i.e. the bar heights of Figure 6. Decoding with vLLM 0.19 (CUDA 13.0, repetition penalty 1.0); GPT scores use GPT-5.4-nano as judge. Best per column within each backbone in **bold**.

Backbone	Stage	CC-Hard		D2C-All		D2C-Hard		WC-S		WC-M		WC-L		**Avg**	
		GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C	GPT	D2C
InternVL3.5-1B	Base	1.72	46.1	2.59	58.8	2.20	58.0	2.70	61.2	2.61	58.1	1.88	49.6	2.28	55.3
	WebCode2M SFT	1.97	63.7	2.20	68.8	1.33	60.1	2.32	69.7	2.78	72.2	2.32	68.6	2.15	67.2
	WebGround-3M SFT	2.55	71.7	3.13	76.3	2.23	66.7	2.96	72.5	3.18	74.5	2.93	74.6	2.83	72.7
	DPO	**3.78**	**78.6**	**3.87**	**80.8**	**3.50**	**69.3**	**3.95**	**78.5**	**4.21**	**80.5**	**4.02**	**79.2**	**3.89**	**77.8**
SmolVLM2-2.2B	Base	1.82	62.1	2.71	70.8	1.85	58.7	2.11	61.1	2.04	61.9	2.04	64.6	2.10	63.2
	WebCode2M SFT	1.70	60.8	2.17	66.4	1.29	52.1	2.27	68.1	2.68	69.4	2.42	67.0	2.09	64.0
	WebGround-3M SFT	2.67	67.9	2.80	71.5	1.74	59.4	3.12	73.8	3.45	75.6	3.06	73.4	2.81	70.3
	DPO	**3.00**	**70.8**	**3.35**	**76.2**	**1.91**	**61.2**	**3.35**	**75.4**	**3.85**	**77.9**	**3.48**	**75.2**	**3.16**	**72.8**
Phi-4-multimodal (5.6B)	Base	1.26	44.6	1.83	55.0	1.15	35.0	1.33	42.3	1.39	39.8	1.39	49.4	1.39	44.3
	WebCode2M SFT	0.35	36.2	0.87	48.2	0.45	38.7	0.80	47.7	0.63	44.0	0.44	38.8	0.59	42.3
	WebGround-3M SFT	2.79	67.9	3.25	74.9	1.94	61.0	3.44	74.1	3.59	74.8	3.29	72.1	3.05	70.8
	DPO	**3.96**	**74.7**	**3.91**	**80.1**	**3.13**	**71.4**	**4.26**	**79.2**	**4.41**	**79.2**	**3.88**	**76.9**	**3.93**	**76.9**
Gemma-3-4B	Base	2.85	68.2	**4.11**	79.1	3.11	70.2	3.67	73.8	3.27	71.5	3.12	71.1	3.36	72.3
	WebCode2M SFT	3.34	72.6	2.75	73.9	1.93	61.7	3.11	74.9	3.62	77.9	3.44	76.4	3.03	72.9
	WebGround-3M SFT	3.97	77.8	3.30	75.8	2.29	64.1	3.51	77.2	3.85	79.3	3.71	78.3	3.44	75.4
	DPO	**5.21**	**83.8**	3.96	**82.5**	**3.28**	**72.9**	**4.27**	**81.8**	**4.63**	**82.2**	**4.49**	**82.3**	**4.31**	**80.9**

D.7 Qualitative results and remaining errors

Figure 8 presents five randomly sampled CC-Hard examples. From top to bottom, their benchmark identifiers are 088, 044, 040, 080, and 067. These examples illustrate individual behaviors; the aggregate metrics summarize performance over the full benchmark.

In example 088, w1 preserves the green navigation bar and right sidebar, but expands the header and page height. In example 044, it retains the article-and-sidebar arrangement, while the lower content blocks differ in layout and density from the reference. Example 040 retains the black-and-gold header, right sidebar, and footer, but replaces the central image with a placeholder and changes vertical spacing. In example 080, the narrow article and sidebar remain recognizable, but the two reference photographs are not reproduced as separate image regions. Example 067 preserves the main article and sidebar, yet the cover-image placeholder is wider and vertical spacing differs. These examples show that plausible overall structure can coexist with local layout errors and missing media content. Figure 9 enlarges two regions to make these limitations visible without relying on the reduced full-page panels.

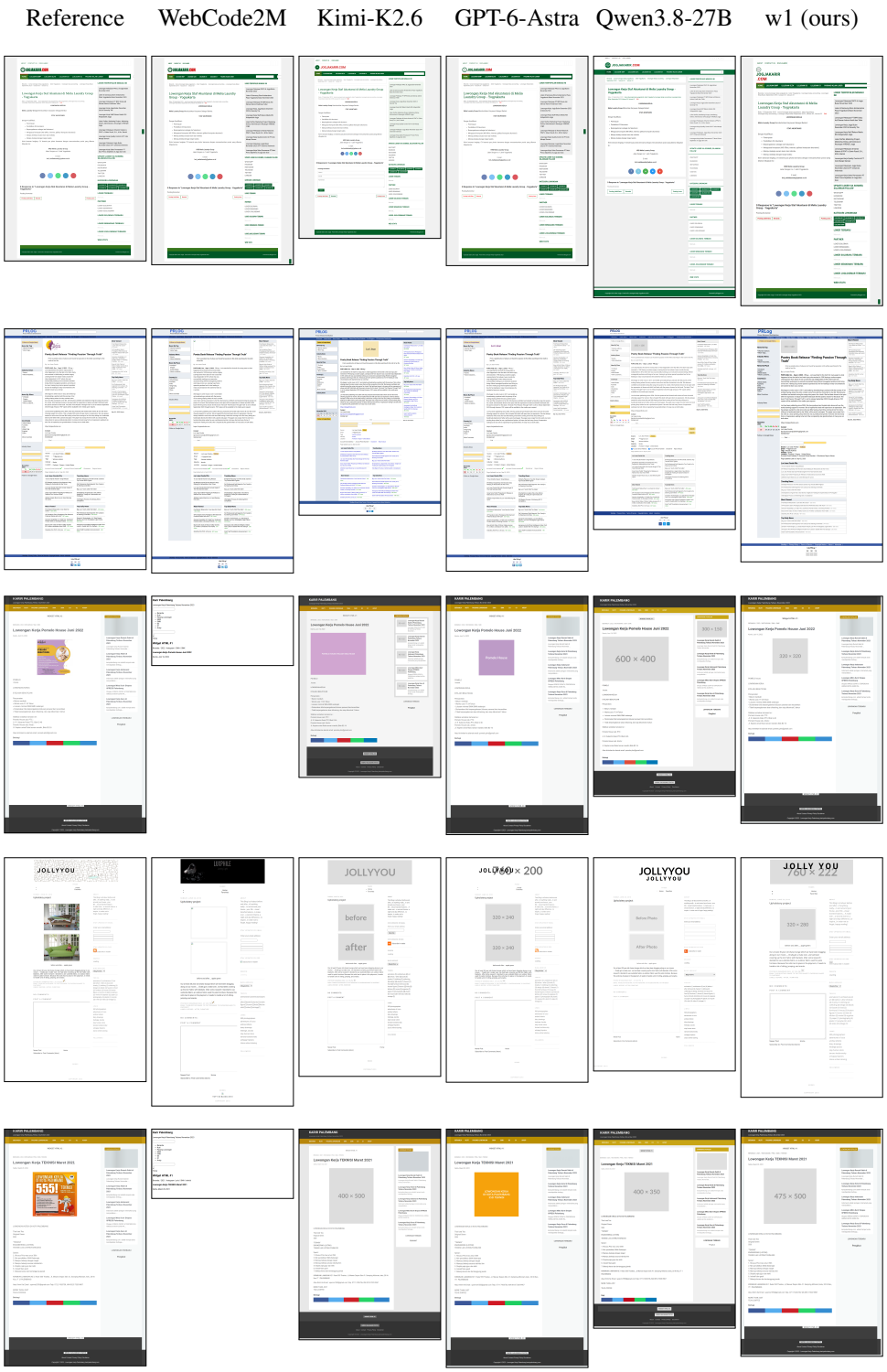
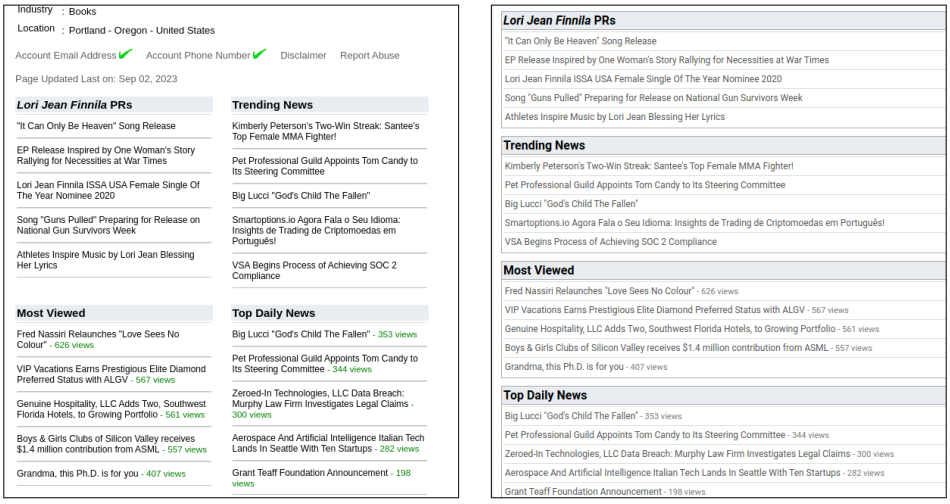


Figure 8: Qualitative comparison on five randomly sampled CC-Hard pages (088, 044, 040, 080, and 067, from top to bottom). Each row shows the reference screenshot followed by outputs from Qwen3-VL-4B-Instruct fine-tuned on WebCode2M, Kimi-K2.6, GPT-6-Astra, Qwen3.8-27B, and w1. All panels are the unmodified screenshots produced by the evaluation pipeline, top-aligned and shown at a common width. The comparison highlights differences across systems in page scale, column preservation, media-slot geometry, and text layout. Individual successes and remaining errors are discussed in Section D.7; enlarged details appear in Figure 9.

Reference w1-4B-DPO

CC-Hard 044: lower article blocks



CC-Hard 080: article images and sidebar

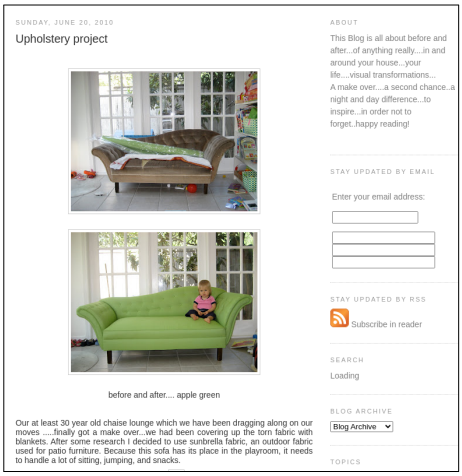




Figure 9: **Enlarged examples of remaining layout errors.** Details from the randomly sampled pages in Figure 8, selected to illustrate two visible failures. In page 044, the reference arranges the lower article blocks in two columns, whereas w1 stacks them vertically. In page 080, two separate reference photographs are replaced by one placeholder region, shifting the following text upward. Within each pair, the crop uses the same source-pixel coordinates and magnification; these are enlarged excerpts of the original screenshots, without content edits.