ReLook: Vision-Grounded RL with a Multimodal LLM Critic for Agentic Web Coding

Yuhang Li[1,†], **Chenchen Zhang**[1,†,*], **Ruilin Lv**[2], **Ao Liu**[1], **Ken Deng**[2], **Yuanxing Zhang**[3], **Jiaheng Liu**[4], **Bo Zhou**[1,*]
[1]Hunyuan Team, Tencent, [2]Independent Researcher,
[3]Peking University, [4]Nanjing University
596983629@qq.com, {adamwzhang,chaysezhou}@tencent.com

Abstract

While Large Language Models (LLMs) excel at algorithmic code generation, they struggle with front-end development, where correctness is judged on rendered pixels and interaction. We present **ReLook**, an *agentic*, vision-grounded reinforcement learning framework that empowers an *agent* to close a robust generate–diagnose–refine loop by invoking a multimodal LLM (MLLM) as a tool. During training, the agent employs an MLLM-in-the-loop to serve as a visual critic, evaluating code via screenshots and providing actionable feedback. Crucially, we enforce a strict zero-reward policy for invalid renders to guarantee renderability and mitigate reward hacking. To prevent behavioral collapse, we introduce *Forced Optimization*, a strict acceptance rule that admits only improving revisions, yielding monotonically better trajectories. At inference, we decouple the critic and run a lightweight, critic-free self-edit cycle, keeping latency *comparable to base decoding* while retaining most of the gains. Across three widely used benchmarks, ReLook consistently outperforms strong baselines in vision-grounded front-end code generation, highlighting the benefits of agentic perception, visual rewards, and training–inference decoupling.

1 Introduction

Large Language Models (LLMs) excel on closed-form benchmarks—programming contests (Li et al., 2022), SQL synthesis (Liu et al., 2024), and mathematical reasoning (Yang et al., 2024)—yet still underperform on front-end code generation, where visual fidelity and interaction are first-class. Unlike binary unit tests in algorithmic tasks, front-end quality lies on a continuum: a single misaligned pixel can signify failure.

This perceptual barrier explains current shortcomings: text-only models are blind to pixel-level consequences, yielding (i) layout drift, (ii) interaction breakage, and (iii) aesthetic inconsistency. To address this, a model must (1) see rendered HTML/CSS/JS/SVG, (2) diagnose misalignments and broken interactions, and (3) iteratively refine in situ. Existing methods miss this loop: one-shot vision-to-code systems (pix2code (Wüst et al., 2024), Design2Code (Si et al., 2024), UICoder (Wu et al., 2024)) generate but do not refine; self-refinement frameworks (CodeRL (Le et al., 2022), Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), CRITIC (Gou et al., 2023; Peng et al., 2025; Zhang et al., 2025b)) iterate but cannot see, relying on pixel-blind unit tests or linters.

To bridge this gap, we introduce **ReLook**, a vision-grounded agentic reinforcement learning framework that completes the generate–diagnose–refine loop. The agent actively invokes an MLLM as a tool to "see" rendered outputs and obtain rich textual suggestions during inference, enabling true iterative refinement. Training uses a comprehensive reward system: a powerful MLLM (e.g., Qwen2.5-VL (Wang et al., 2024)) supplies the perceptual signal text-only methods lack, and a rendering-integrity rule assigns zero reward when required screenshots are invalid to deter reward hacking.

However, we identify a critical challenge: behavioral collapse, where despite high-quality feedback, a subsequent revision can be worse. We adopt a Forced Optimization strategy that accepts only strictly improving steps, ensuring high-quality, monotonically improving trajectories. For low-latency inference, the external critic can be discarded; the model performs a lightweight self-edit cycle—render, self-edit, and converge quickly to a human-aligned result.

Evaluator validity and choice. Our offline evaluation strictly follows the ArtifactsBench protocol(Zhang et al., 2025a). ArtifactsBench

† The first two authors contributed equally to this work.
* Corresponding authors.

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 25471–25485
July 2-7, 2026

establishes evaluator validity through: (i) controlled human studies demonstrating over 90% agreement between MLLM judges (Gemini-2.5-Pro, Qwen2.5-VL-72B) and human experts, and (ii) strong ranking correlation with WebDev Arena(LMSYS Org, 2024), a large-scale crowdsourced platform. Since our test sets are strict subsets of ARTIFACTSBENCH's evaluated tasks, we directly inherit this established human-alignment evidence. To further mitigate on-policy judge overfitting, we decouple the training-time critic (Qwen2.5-VL-72B-Instruct) from the offline evaluator (Gemini-2.5-Pro). We do not conduct additional human studies; validity rests on ARTIFACTSBENCH's rigorous validation. See Appendix for detailed protocol adherence and cross-judge analysis.

Our contributions are as follows:

- **Robust Reward System.** We employ an MLLM as the reward model to provide the rich, pixel-level training signal that text-only methods cannot capture. This is critically supplemented by a zero-reward rule for answers without screenshots, which is designed to prevent reward hacking by forcing the agent to produce renderable code.

- **Agent reinforcement learning Framework.** We empower the agent to perform a generate–diagnose–refine loop by actively invoking an MLLM as a diagnostic tool. The agent can "see" its rendered output and receive rich, actionable feedback for iterative improvement. To ensure this powerful loop is productive and stable, we introduce a Forced Optimization strategy that addresses the challenge of behavioral collapse by guaranteeing the construction of high-quality, monotonically improving rollout trajectories.

- **Broad Applicability.** We perform extensive experiments on three widely-used benchmark datasets, and demonstrate that ReLook significantly outperforms the baselines. Moreover, we show the compatibility of ReLook by integrating it with different LLMs.

2 Method

2.1 Problem Formulation

Given a natural-language query q, the model outputs text t and front-end code c (HTML/CSS/JS/SVG). Correctness depends on rendered appearance and behavior rather than syntax. We therefore use an MLLM judge to provide perceptual reward and design an agentic RL framework that allows vision-grounded feedback to refine c.

2.2 Overall Framework

ReLook runs a generate–diagnose–refine loop: render to temporal screenshots, score with an MLLM, incorporate feedback, and continue under a *Forced Optimization* rule that accepts only strictly improving steps. Reflection is internalized so the external critic can be dropped at test time.

2.3 Iterative Reflection Mechanism

For each query q, the policy emits t and c. Upon <get_feedback>, we execute c in a sandbox, capture screenshots, and query the MLLM for feedback m (wrapped in <mllm_feedback>). We feed $\{q, t, c, m\}$ into the next round and stop when feedback is not requested or a round cap is reached.

The final output is represented as:

$$o = [t_1 \oplus c_1 \oplus m_1 \oplus \cdots \oplus t_R \oplus c_R] \quad (1)$$

where t_r, c_r, m_r denote the r-th round's text, code and feedback blocks, and R is the total number of reflection rounds. The prompt template is provided in the appendix.

2.4 Reinforcement Learning Framework

To optimize this framework, we employ Group Relative Policy Optimization (GRPO) as our training algorithm, which is based on a token-level policy gradient loss and is related to PPO (Schulman et al., 2017) while differing from preference-based objectives such as DPO (Rafailov et al., 2023). The objective is defined as follows:

$$\begin{aligned}
&\mathcal{J}_{GRPO}(\theta) \\
&= \mathbb{E}\left[q \sim P(Q), \{o_i\}_{i=1}^{G} \sim \pi_{\theta_{\text{combined}}}(O|q)\right] \\
&= \frac{1}{\sum_{i=1}^{G} |o_i|} \sum_{i=1}^{G} \sum_{t=1}^{|o_i|} \Bigg\{ \min\left[\frac{\pi_\theta(o_{i,t}|q_{i,t})}{\pi_{\theta_{old}}(o_{i,t}|q_{i,t})} \hat{A}_{i,t}, \right. \\
&\quad \left. \text{clip}\left(\frac{\pi_\theta(o_{i,t}|q_{i,t})}{\pi_{\theta_{old}}(o_{i,t}|q_{i,t})}, 1-\varepsilon, 1+\varepsilon \right) \hat{A}_{i,t} \right] \\
&\quad - \beta \, \mathrm{D}_{KL}\left[\pi_\theta || \pi_{ref}\right] \Bigg\}
\end{aligned} \quad (2)$$

Only tokens in t, c contribute non-zero advantages; critic tokens m are masked ($\hat{A}_{i,t}$=0 on m).

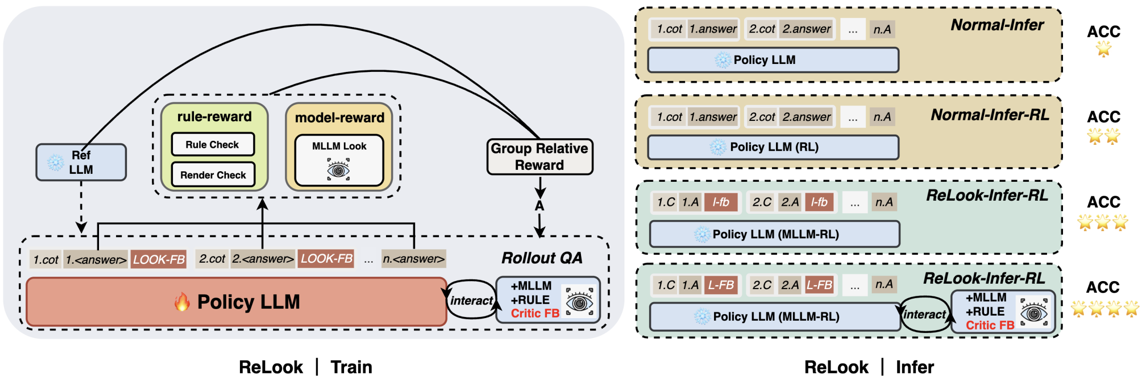

Figure 1: Overview of **ReLook**. Left: training closes a generate–diagnose–refine cycle: policy LLM generates code, pages rendered to temporal screenshots, and a vision-aware critic (MLLM) provides scores and feedback. Rewards combine visual scoring and format constraints; the policy is optimized with GRPO. Right: at inference the model runs a lightweight Re-Look cycle — external critic may be omitted for latency or used for higher accuracy.

Advantage estimation and credit assignment. We sample G trajectories per query and compute returns from $R_{\text{ReLook}}(o_i)$ (Eq. 5). Using the group mean b as baseline, the advantage is $\tilde{A}_i = R_{\text{ReLook}}(o_i) - b$, broadcast to policy tokens (text t, code c) while masking critic tokens m ($\hat{A}_{i,t}$=0 on m). Advantages are standardized and clipped to $[-2, 2]$. We regularize toward π_{ref} with KL weight β. The combined policy $\pi_{\theta_{\text{combined}}}$ is defined as: $[\pi_{\theta_{\text{old}}}$ for $t_r, c_r] \oplus [\pi_{\text{MLLM}}$ for $m_r]$, where π_{MLLM} is a frozen MLLM critic (Qwen2.5-VL-72B-Instruct). Trajectories mix policy tokens (t_n, c_n) and critic tokens (m_n). During training, only π_θ is updated. The hyperparameters ε and β control clipping range and KL regularization strength.

Token masking and lightweight feedback distillation. We optimize GRPO over policy tokens (t, c) and mask critic tokens m by zeroing advantages. To enable critic-free inference, we optionally distill m-tokens with lightweight loss $\mathcal{L}^m_{\text{distill}}$ toward frozen MLLM outputs (see Appendix §F for details). The total objective is $\mathcal{L} = -\mathcal{J}^{t,c}_{\text{GRPO}} + \gamma\, \mathcal{L}^m_{\text{distill}}$ (default γ=0.1, sweep $\gamma \in [0.05, 0.3]$). This lets RL improve visual quality while distillation transfers feedback style for test-time self-reflection.

2.5 Reward Design

Conventional RL signals for code, such as unit tests or linters, operate purely in the text domain and are blind to visual defects. To address this, our reward is derived from a powerful Multimodal LLM (Qwen2.5-VL-72B-Instruct) that scores rendered pages based on the user prompt, the generated code, and a series of temporal screenshots. Capturing screenshots at multiple time points allows the critic to assess dynamic behavior. Within each reflection round we capture three time points $\{S_1, S_2, S_3\}$ (e.g., post-load, +1s, +2s) and jointly evaluate them in a single scoring call to obtain one round-level score. To properly credit incremental progress across rounds, we then average the round-level scores within a trajectory.

A critical component of this design is a safeguard against reward hacking, where malformed but syntactically plausible code might be over-rewarded. We enforce a strict renderability constraint: if any required screenshot is invalid (e.g., due to a render failure or timeout), the reward is zero. While this eliminates degenerate reward channels, it can lead to sparse rewards early in training. To mitigate this, we engineer all prompts to include a visual-output constraint that explicitly instructs the agent to write executable HTML/CSS/JavaScript/SVG code suitable for browser rendering (see Appendix for full prompt template). This simple but effective technique increases the initial Valid Render Rate (from ~40% at the start of training to ~80% upon convergence) and better aligns the generation task with our vision-grounded reward. This reward function is formally defined as:

$$R_{\text{MLLM}}(o) = \begin{cases} \text{VisualScore}(o) & \text{if screenshot valid} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

To discourage repetition, we apply a linear length penalty from L_{start} to L_{end} (12k and 14k tokens):

$$R_{\text{len}}(o) = \begin{cases} 1 & \text{if } \text{len}(o) < L_{\text{start}} \\ \frac{L_{\text{end}} - \text{len}(o)}{L_{\text{end}} - L_{\text{start}}} & \text{if } L_{\text{start}} \leq \text{len}(o) \leq L_{\text{end}} \\ 0 & \text{if } \text{len}(o) > L_{\text{end}} \end{cases} \tag{4}$$

The final training reward is:

$$R_{\text{ReLook}}(o) = R_{\text{MLLM}}(o) \cdot R_{\text{len}}(o) \tag{5}$$

2.6 Forced Optimization

While the MLLM critic provides rich, detailed feedback, we observe a critical instability we term behavioral collapse: despite high-quality suggestions, a subsequent revision may score lower than the previous one, degrading trajectories.

We therefore adopt a **Forced Optimization** mechanism. We initially explored a *negative-reward penalty* for regressions (rejecting worse-than-previous outcomes by penalizing returns), but found it incentivized the agent to *reduce* reflection frequency to avoid penalties, i.e., reward hacking. Hence, we discard the penalty design and enforce a *strict acceptance rule*: a refinement step (new t_{r+1}, c_{r+1}) is accepted *only if* its reward strictly exceeds the best-so-far in the trajectory (no ε margin or re-scoring). Non-improving steps are rejected and a new attempt is sampled, with a maximum of 10 resampling attempts per reflection round. If the limit is reached without improvement, we terminate further reflection for that trajectory and use the best-so-far result. This guarantees monotonically improving accepted trajectories and stabilizes learning without suppressing useful reflections.

2.7 Efficient Inference

During training we retain the MLLM feedback loop for self-correction. At inference, we drop the external MLLM and run a lightweight, critic-free self-edit (at most three rounds), with screenshots and MLLM calls disabled. This preserves most gains while substantially reducing latency, following a train-slow, run-fast paradigm. See Appendix for pseudocode.

3 Experiment

3.1 Experimental Settings

Training Data Curation. We curate a 3,000-task corpus of front-end-only tasks, normalize descriptions, and remove near-duplicates via lexical/DOM/code similarity. Prompts are audited to remove hints and sanitized; the final data are split into train/val stratified by UI archetypes.

Train–Test De-duplication Protocol. To prevent leakage to ArtifactsBench, FullStack-Bench-Html and Web-Bench, we run instance-level, multi-view de-duplication before training:

- **Lexical**: TF–IDF over char 3-grams; cosine > 0.85.
- **DOM**: tag-bigram Jaccard > 0.90; fallback tree-edit distance for edge cases.
- **Code**: token-set Jaccard > 0.90 after stripping comments/whitespace and minifying.

If any criterion triggers, the instance is removed. Borderline cases (e.g., lexical in $[0.80, 0.85]$ plus structural overlap) are manually reviewed. The same procedure purges intra-train near-duplicates.

Dataset. We evaluate the performance of our method on three widely used datasets: ArtifactsBench (Zhang et al., 2025a), FullStack-Bench-Html (Cheng et al., 2024) and Web-Bench (Xu et al., 2025). ArtifactsBench contains 1,825 tasks focused on generating dynamic and interactive visual outputs, such as SVG visualizations and mini-games. Its evaluation protocol uses a Multimodal LLM to score the visual fidelity and interactive integrity of the rendered code. ArtifactsBench utilizes Gemini-2.5-Pro as an expert judge to evaluate model outputs across its 1,825 tasks, demonstrating over 90% agreement with human evaluators. To manage evaluation costs, we conduct our experiments on six of its sub-datasets. We use the shorthand A-* for ArtifactsBench subsets: A-Lite (300 randomly sampled cases), A-Easy (305 simple frontend cases), A-Game (all 413 game-related cases), A-SVG (all 123 SVG-focused cases), A-Web (all 447 web-specific cases), and A-Si (all 75 simulation-oriented cases). FullStack-Bench-Html provides a collection of front-end programming tasks where functional correctness is programmatically validated by passing a suite of predefined unit tests. Web-Bench simulates realistic web development workflows through 50 complex projects of 20 sequential tasks each. Following its official protocol, performance is measured by passing end-to-end test cases that validate the final project's functionality, and we report the pass@2 rate.

Sandboxed Rendering Environment. We execute model-produced code within a headless Chromium-based renderer that is both deterministic and secure. This sandboxed environment operates at the OS level, with filesystem and network access disabled. Safety is further enhanced by blocking dangerous APIs (e.g., `window.open`, `alert/confirm/prompt`, `eval/Function`, clipboard access, non-local `fetch/XMLHttpRequest/WebSocket`), and enforcing a per-sample wall-clock timeout. Requests to non-whitelisted origins are intercepted and failed closed. To ensure determinism, all external resources like fonts and images are replaced with local fixtures; timers and animations use deterministic seeds; and we enforce a strict Content Security Policy that disallows inline scripts and remote scripts. Within this controlled environment, our visual capture mechanism is dynamic: we take full-page screenshots that automatically adjust to the content's full dimensions, guaranteeing no elements are missed. To capture dynamic behavior, we record these screenshots at $T=3$ distinct time points (e.g., post-load, +1s, +2s), which are then passed to the MLLM critic to compute the reward signal.

Evaluator validity and cross-judge robustness. We adopt the evaluation protocol from ARTIFACTSBENCH(Zhang et al., 2025a), which establishes validity through two key mechanisms: (i) a controlled human study showing over 90% agreement between MLLM evaluators (Gemini-2.5-Pro and Qwen2.5-VL-72B) and human experts across diverse front-end tasks, and (ii) strong ranking correlation (Spearman $\rho > 0.85$) with WebDev Arena(LMSYS Org, 2024), a large-scale crowdsourced evaluation platform. Critically, our test sets (A-Lite, A-Easy, A-Game, A-SVG, A-Web, A-Si) are strict subsets of ARTIFACTSBENCH's human-validated tasks, allowing us to directly inherit the established human-alignment evidence. Consistent with ARTIFACTSBENCH, we decouple the training-time critic (Qwen2.5-VL-72B-Instruct, open-source) from the offline evaluator (Gemini-2.5-Pro, proprietary). Cross-judge consistency and further details are summarized in Appendix §B.

3.2 Evaluation Protocol

We follow a pixel-grounded evaluation protocol aligned with ARTIFACTSBENCH. For vision-based front-end tasks, models produce code that is executed in our sandboxed browser to capture temporal screenshots at three time points (post-load, +1s, +2s). An independent evaluator (Gemini-2.5-Pro) assigns a VisualScore on the $[0, 100]$ scale considering: (i) adherence to the textual specification, (ii) layout alignment and spatial fidelity, (iii) typography and color coherence, and (iv) interactive integrity when actions are specified. If no valid screenshot is produced, the score is set to zero. We report means over three runs (three random seeds) with identical decoding parameters.

For FullStack-Bench-Html, we follow the benchmark's official unit-test protocol and report the pass rate under the same inference setup as other methods. For Web-Bench, we follow its official end-to-end evaluation and report pass@2 across projects. Unless otherwise stated, all reported metrics—including Web-Bench pass@2—are averaged over three runs. Decoding hyperparameters (temperature and top-p) are fixed across systems unless otherwise stated; local fixtures are cached to avoid network variance.

Baselines and variants. Our experiments are conducted on two strong instruction-tuned base models: Qwen2.5-7B-Instruct and Llama-3.1-8B-Instruct. On each of these backbones, we compare three distinct approaches: (i) *Base Model* (the frozen instruction-tuned model, serving as a direct baseline), (ii) *Web-RL* (vision-grounded RL using the MLLM reward but without the agentic reflection mechanism), and (iii) *ReLook* (our full framework with agentic MLLM-in-the-loop reflection). All methods use identical inference parameters, prompts, and rendering infrastructure for fair comparison. We also include results for GPT-4o (via OpenAI API) and Qwen2.5-32B-Instruct (local deployment) as reference points representing stronger base models; these are evaluated under the same protocol. We do not compare with prior specialized visual code generation methods (e.g., Design2Code(Si et al., 2024), UICoder(Wu et al., 2024)) because: (i) they were evaluated on different datasets without established cross-benchmark protocols, and (ii) official implementations are not publicly available for controlled comparison on our benchmarks. Our Web-RL baseline serves as a strong vision-aware RL reference that isolates the contribution of agentic reflection.

Manual validation (GSB). To complement automated evaluation, we conduct a double-blind human study on 100 randomly sampled tasks comparing Qwen2.5-7B-ReLook against Qwen2.5-7B-Instruct. Five independent annotators directly run both systems' code in our sandboxed renderer and select one of three labels: G (ReLook better), S (same), B (ReLook worse). Majority voting aggregates per-task labels. Results are summarized as G:S:B $= 50 : 30 : 20$, indicating a clear preference for ReLook under human judgment.

Implementation Details We implement ReLook with grouped rollouts under GRPO. The model is trained for a maximum of 40 steps with a training batch size of 256. The learning rate is set to 1e-6, and we employ a linear warmup (first 5%) and cosine decay schedule. For the GRPO loss function, we set the group size G for rollouts per query to 8 and the clipping parameter ε to 0.2. We apply a KL penalty toward a reference policy with a non-zero weight β and sweep $\beta \in [0.01, 0.05]$ (step 0.01). We also sweep the advantage clipping bound in $\{1, 2, 3\}$ for robustness. The group size and the number of sampled trajectories per query are chosen to fit accelerator memory while maintaining adequate exploration; gradient accumulation is used to emulate larger effective batch sizes. Mixed precision (bfloat16/fp16) and activation checkpointing reduce memory footprint. For critic feedback tokens, GRPO is masked out; we optionally add a lightweight distillation loss with weight γ on the feedback tokens to imitate the frozen MLLM's feedback style. Unless otherwise noted, we use γ=0.1 and sweep $\gamma \in [0.05, 0.3]$ in sensitivity checks. For the length penalty, we set the bounds to $L_{\text{start}} = 12k$ and $L_{\text{end}} = 14k$.

We use 64 GPUs (32 policy, 32 MLLM), 80 minutes per step, and a curated corpus. Decoding is identical across systems (temp 1.0, top-p 0.7); results average three seeds. Prompts and sandbox are shared. We select checkpoints by mean VisualScore. We focus on 7B/8B backbones; larger-scale RL is future work.

3.3 Main Results

Table 1 and Figure 2 present the main results. ReLook achieves substantial improvements over both base models and Web-RL (vision-grounded RL without agentic reflection). Notably, ReLook-w/o-MLLM—which relies solely on internalized reflection without external critic calls—still outperforms Web-RL, demonstrating successful internalization of the refinement mechanism with minimal inference overhead.

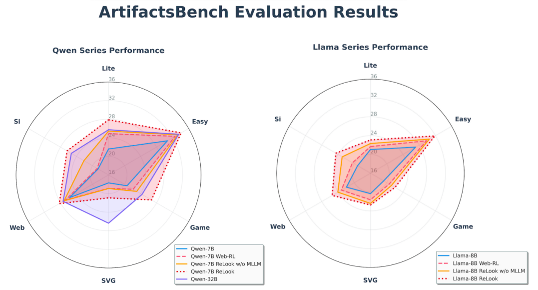

Figure 2: Radar plot showing ReLook's consistent improvements across all ArtifactsBench subsets for both Qwen2.5-7B and Llama-3.1-8B backbones (averaged over 3 seeds).

Beyond absolute scores, we consistently observe the strictly monotone ordering predicted by our design: ReLook > Web-RL > Base Model. The gap between Web-RL and ReLook underscores the importance of a *vision-aware* training signal coupled with the generate–diagnose–refine loop. Qualitative examples are shown in Appendix Figure 6.

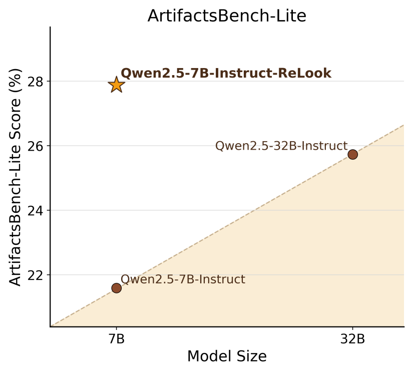

Figure 3: Performance on ArtifactsBench-Lite showing consistent ordering: ReLook > Web-RL > Base Model. Results averaged over 3 seeds.

Figure 3 summarizes performance on the ArtifactsBench-Lite subset. It exhibits the strictly monotone ordering ReLook > Web-RL > Base Model and highlights consistent gains for both Qwen2.5-7B and Llama-3.1-8B backbones, echoing the trends in Table 1. This compact subset mirrors the broader benchmark and provides an intuitive visualization of the average improvements delivered by ReLook.

Figure 4 evidences behavioral collapse in the base model after 2–3 rounds despite feedback, while ReLook improves monotonically. We cap reflections at three rounds for efficiency and keep this at inference.

Figure 5 shows that reflection frequency increases and stabilizes during training, aligning with

Model	A-Lite	A-Easy	A-Game	A-SVG	A-Web	A-Si	FullStack-Bench-Html	Web-Bench
Qwen2.5-32B-Instruct	25.73	33.52	24.36	26.36	27.34	25.30	70.00	10.90
GPT-4o	33.25	34.23	33.04	33.74	34.31	31.44	71.25	23.80
Claude 3.5 Sonnet	39.64	49.37	38.46	41.94	41.26	39.43	81.88	32.39
Llama-3.1-8B-Instruct	21.04	27.11	19.25	20.33	21.91	19.18	57.50	2.50
Llama-3.1-8B-Instruct-Web-RL	21.67	29.80	20.61	21.64	23.15	20.44	61.75	2.75
Llama-3.1-8B-Instruct-ReLook-w/o-MLLM	22.32	30.52	21.18	22.41	24.03	22.97	63.75	2.90
Llama-3.1-8B-Instruct-ReLook	**23.08**	**31.86**	**22.04**	**22.74**	**25.42**	**24.52**	**63.75**	**2.90**
Qwen2.5-7B-Instruct	21.59	30.70	20.62	17.70	25.69	18.61	65.00	3.00
Qwen2.5-7B-Instruct-Web-RL	24.89	32.64	22.14	18.87	26.73	18.82	63.25	3.48
Qwen2.5-7B-Instruct-ReLook-w/o-MLLM	25.44	33.29	23.05	18.92	27.11	22.15	67.50	4.20
Qwen2.5-7B-Instruct-ReLook	**27.88**	**34.12**	**26.72**	**20.92**	**28.31**	**26.36**	**67.50**	**4.20**

Table 1: Main results on ArtifactsBench subsets (A-Lite/Easy/Game/SVG/Web/Si), FullStack-Bench-Html, and Web-Bench (pass@2). VisualScores follow ARTIFACTSBENCH $[0, 100]$ scale. ReLook uses up to 3 reflection rounds; ReLook-w/o-MLLM relies on internalized self-reflection without external critic. For unit-test benchmarks (FullStack, Web-Bench), both variants use identical critic-free inference, yielding same scores. Bold: best per backbone. Means over 3 seeds (temp=1.0, top-p=0.7).

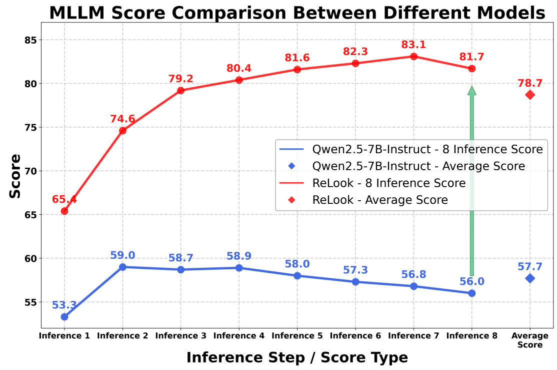

Figure 4: Behavioral collapse mitigation. Base model (Qwen2.5-7B-Instruct) degrades after initial attempts despite MLLM feedback, while ReLook exhibits monotonic improvement across eight forced reflection rounds on ArtifactsBench-Lite. Scores from training-time judge (Qwen2.5-VL-72B).

Forced Optimization's incentive structure. The average converging to 2 rounds suggests most tasks reach capacity after two refinements.

Due to space limitations, additional analyses regarding the scalability of the critic model and performance in out-of-distribution scenarios are provided in Appendix A.

3.4 Ablation Study

We ablate three components: (i) vision-based MLLM reward, (ii) format constraint invalidating non-renderable outputs, and (iii) Forced Optimization. Table 2 shows each component is critical. Vision reward provides essential perceptual signal (+3.3 points). Format constraint prevents reward hacking (+1.0). Forced Optimization has the largest impact (+2.0), directly mitigating behavioral collapse. "Single-sample" denotes limiting ReLook to sample only one instance per reflection round, ensuring that the gains do not stem from an increased generation budget. All ablations use identical seeds and the external evaluator to avoid bias.

MLLM Reward	Format Constraint	Forced Optimization	ArtifactsBench-Lite ↑
			21.59
✓			24.89
✓	✓		25.84
✓	✓	Single-sample	26.98
✓	✓	✓	27.88

Table 2: Ablation Study on ArtifactsBench-Lite. Results are averaged over 3 random seeds.

3.5 Inference speed improvement after removing MLLM

We use VLLM to deploy the Qwen2.5-7B-Instruct model on four H20 GPUs and the Qwen2.5-VL-72B-Instruct model on eight H20 GPUs. We set the number of parallel threads to 1 and limit the maximum number of reflection steps to 3. We conduct inference on 100 queries and measure the average inference time per query. ReLook takes an average of 123.04 seconds per query, whereas ReLook-w/o-MLLM takes only 18.03 seconds. The results indicate that removing the screenshot and MLLM invocation mechanism substantially improves inference efficiency.

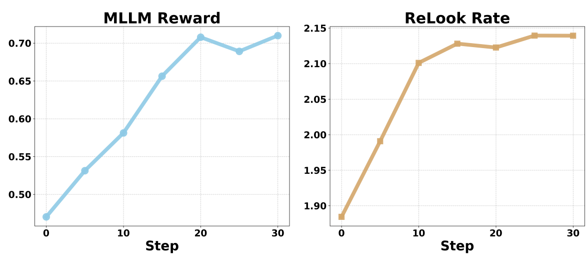

Figure 5: Intermediate Results of RL Training

3.6 Qualitative Analysis

Appendix H presents a qualitative analysis of ReLook.

4 Related Work

4.1 Visual Code Generation

Early vision-to-code systems translate static screenshots to HTML/CSS (Wüst et al., 2024) in one shot. Recent methods add structure—Design2Code (Si et al., 2024), Web2Code (Yun et al., 2024), UICoder (Wu et al., 2024), DesignCoder (Chen et al., 2025)—but mainly optimize static similarity. While these methods improve structural representation, they fundamentally lack a closed-loop mechanism to iteratively see, diagnose, and refine their own rendered outputs. ReLook addresses this by placing a renderer directly in the training loop, using vision-grounded rewards from temporal screenshots, and internalizing the refinement process via RL.

4.2 Feedback-Driven Code Reinforcement Learning

RL for program synthesis leverages unit-test rewards and large candidate sets (Le et al., 2022; Li et al., 2022); reflective/tool-driven critiques improve correction (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2023; Peng et al., 2025). However, for front-end tasks, these approaches are pixel-blind. Even with structured visual instructions, their reliance on proxy signals like unit tests or linters cannot address defects in visual layout or interactive behavior. We couple the policy with a direct perceptual reward from temporal screenshots and explicitly stabilize this noisy learning process via Forced Optimization and a zero-reward rule for invalid renders.

4.3 Acceptance criteria, best-of-N, and verifier-assisted search.

A broad line of work improves generation via external selection/search: Codex/AlphaCode sample-and-test (Chen et al., 2021; Li et al., 2022); self-consistency/tree deliberation aggregate candidates (Wang et al., 2022; Yao et al., 2023); agentic self-refinement uses iterative critique (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2023). Our *Forced Optimization* differs in both *criterion* and *rule*: a vision-grounded score from temporal screenshots (not unit tests), and acceptance *strictly* requiring monotonic improvement within one trajectory. Unlike best-of-N or offline re-ranking, our rule is *online, in-trajectory*, preventing regressions and reward hacking, yielding stable, visually aligned improvements for front-end code.

4.4 Multimodal UI Perception and Evaluation

Recent MLLMs ground web elements and layouts (OpenAI, 2023; Wang et al., 2024); web-agent/GUI benchmarks show the value of vision-conditioned reasoning (Zhou et al., 2023; Koh et al., 2024; Li et al., 2025). For evaluation, MLLM-as-Judge is common (Ge et al., 2023; Zheng et al., 2023); front-end benchmarks emphasize visual and interactive quality (Xu et al., 2025; Zhang et al., 2025a). Accordingly, we place visual scoring in training to internalize layout/interaction principles, and drop the critic at inference for latency.

ReLook unifies these threads via MLLM-based visual rewards within RL, offering a practical path to visually aware, self-improving front-end generation.

5 Conclusion

We introduce **ReLook**, a vision-grounded RL framework that establishes a robust generate–diagnose–refine loop for front-end code generation. Specifically, ReLook integrates a multimodal LLM critic with two strategic mechanisms: zero-reward penalties for invalid renders and Forced Optimization. As a result, it consistently outperforms strong baselines (ReLook $>$ Web-RL $>$ Base) while enabling efficient, critic-free inference. We anticipate that embedding perception-aligned evaluators in the learning loop will generalize beyond web UIs to broader perceptual programming domains.

6 Limitations

Addressing complex multi-file project tasks, such as those in Web-Bench, remains a significant challenge. This indicates that traversing beyond the scope of single-artifact refinement to achieve robust long-horizon reasoning will necessitate further architectural innovations. Furthermore, our experiments are currently limited to 7B/8B-scale models and front-end development. Therefore, evaluating scalability to larger models and broader software stacks requires additional investigation. Finally, to facilitate reproducibility and support future research, we will make our code publicly available upon publication.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, and 1 others. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Yunnong Chen, Shixian Ding, YingYing Zhang, Wenkai Chen, Jinzhou Du, Lingyun Sun, and Liuqing Chen. 2025. Designcoder: Hierarchy-aware and self-correcting ui code generation with large language models. *arXiv preprint arXiv:2506.13663*.

Bytedance-Seed-Foundation-Code-Team Yao Cheng, Jianfeng Chen, Jie Chen, Li Chen, Liyu Chen, Wentao Chen, Zhengyu Chen, Shijie Geng, Aoyan Li, Bowen Li, Bowen Li, Linyi Li, Boyi Liu, Jerry Liu, Kaibo Liu, Qi Liu, Shukai Liu, Si-Han Liu, Tianyi Liu, and 35 others. 2024. Fullstack bench: Evaluating llms as full stack coders. *ArXiv*, abs/2412.00535.

Wentao Ge, Shunian Chen, Guiming Hardy Chen, Junying Chen, Zhihong Chen, Nuo Chen, Wenya Xie, Shuo Yan, Chenghao Zhu, Ziyue Lin, and 1 others. 2023. Mllm-bench: evaluating multimodal llms with per-sample criteria. *arXiv preprint arXiv:2311.13951*.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. 2023. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Chong Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Ruslan Salakhutdinov, and Daniel Fried. 2024. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. *arXiv preprint arXiv:2401.13649*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Kaixin Li, Ziyang Meng, Hongzhan Lin, Ziyang Luo, Yuchen Tian, Jing Ma, Zhiyong Huang, and Tat-Seng Chua. 2025. Screenspot-pro: Gui grounding for professional high-resolution computer use. *arXiv preprint arXiv:2504.07981*.

Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, and 1 others. 2022. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A survey of nl2sql with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

LMSYS Org. 2024. Chatbot Arena Leaderboard. https://web.lmarena.ai/leaderboard. Accessed: 2024-05-23.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, and 1 others. 2023. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 36:46534–46594.

OpenAI. 2023. Gpt-4v(ision) system card. https://cdn.openai.com/papers/GPTV_System_Card.pdf.

Zhongyuan Peng, Yifan Yao, Kaijing Ma, Shuyue Guo, Yizhe Li, Yichi Zhang, Chenchen Zhang, Yifan Zhang, Zhouliang Yu, Luming Li, and 1 others. 2025. Criticlean: Critic-guided reinforcement learning for mathematical formalization. *arXiv preprint arXiv:2507.06181*.

Raphael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Chelsea Finn, and Christopher D. Manning. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. In *Proceedings of the 34th International Conference on Machine Learning Workshop*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36:8634–8652.

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

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, and 1 others. 2024. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wu, Eldon Schoop, Alan Leung, Titus Barik, Jeffrey P Bigham, and Jeffrey Nichols. 2024. Uicoder: Finetuning large language models to generate user interface code through automated feedback. *arXiv preprint arXiv:2406.07739*.

Antonia Wüst, Wolfgang Stammer, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting. 2024. Pix2code: Learning to compose neural visual concepts as programs. *arXiv preprint arXiv:2402.08280*.

Kai Xu, YiWei Mao, XinYi Guan, and ZiLong Feng. 2025. Web-bench: A llm code benchmark based on web standards and frameworks. *arXiv preprint arXiv:2505.07473*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, and 1 others. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, and 1 others. 2024. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *arXiv preprint arXiv:2406.20098*.

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

Chenchen Zhang, Jinxiang Xia, Jiaheng Liu, Wei Zhang, Yejie Wang, Jian Yang, Ge Zhang, Tianyu Liu, Zhongyuan Peng, Yingshui Tan, Yuanxing Zhang, Zhexu Wang, Weixun Wang, Yancheng He, Ken Deng, Wangchunshu Zhou, Wenhao Huang, and Zhaoxiang Zhang. 2025b. Codecriticbench: A holistic code critique benchmark for large language models. *Preprint*, arXiv:2502.16614.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, and 1 others. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, and 1 others. 2023. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*.

A Additional experimental results

Impact of the Critic Model. With the rapid evolution of multimodal technology, more advanced models such as Qwen3-VL demonstrate superior visual and code comprehension capabilities compared to Qwen2.5-VL. To investigate the impact of the critic's quality on the training outcome, we substitute the original Qwen2.5-VL-72B-Instruct critic with Qwen3-VL-8B-Instruct and Qwen3-VL-32B-Instruct, while employing Qwen2.5-7B-Instruct-ReLook as the base policy model. The comparative results are reported in Table 3. It can be observed that as the capability of the Critic Model strengthens, the performance of the final model also gradually improves.

Evaluation on OOD Scenarios. To further assess the robustness of the model in out-of-distribution (OOD) environments, we evaluate ReLook on LiveCodeBench v6 (2025-02 $\sim$ 2025-05). Qwen2.5-7B achieves a score of 0.213, compared to 0.198 for ReLook. This demonstrates that ReLook maintains competitive performance in OOD scenarios.

Validation on Different Base Models. We additionally validate our framework by instantiating the ReLook Agent with Qwen3-VL-8B-Instruct. Leveraging the model's superior code understanding and visual capabilities, the agent achieves a competitive score of 35.83 on A-Lite. This experiment underscores the framework's inherent scalability and its ability to generalize effectively to diverse baseline models.

Analysis of Selection Bias and Compute Dependence in Forced Optimization. To examine potential selection bias and compute dependence in Forced Optimization (FO), we perform a controlled single-sample ablation that restricts FO to exactly one sample per reflection round. This setting removes any advantage from repeated resampling or selective filtering, thereby isolating the contribution of the optimization mechanism itself under a fixed sampling budget. As shown in Table 4, even under this strict constraint, FO achieves substantial improvements over both the base model and Web-RL on ArtifactsBench-Lite. This indicates that the observed gains are not solely attributable to increased sampling or selection effects. These results suggest that Forced Optimization provides a principled algorithmic advantage beyond increased compute.

B External validity and cross-judge analysis

We align our evaluator setup with ARTIFACTS-BENCH(Zhang et al., 2025a), which establishes validity through rigorous empirical validation:

Human-MLLM Agreement. ARTIFACTS-BENCH conducted a controlled human study with expert web developers evaluating a stratified sample of 200 tasks. Results showed over 90% agreement (Cohen's $\kappa > 0.85$) between human judgments and MLLM evaluators (Gemini-2.5-Pro and Qwen2.5-VL-72B) on visual fidelity, layout correctness, and interactive integrity. The study used a double-blind protocol with three independent human raters per task.

Crowdsourced Validation. Beyond controlled studies, ARTIFACTSBENCH validated MLLM judges against WebDev Arena(LMSYS Org, 2024), a large-scale platform with over 10,000 pairwise comparisons from web developers. The MLLM rankings showed strong correlation (Spearman $\rho = 0.87$ for Gemini-2.5-Pro, $\rho = 0.83$ for Qwen2.5-VL-72B) with crowd preferences.

Our Protocol Adherence. Since our test sets (A-Lite, A-Easy, A-Game, A-SVG, A-Web, A-Si) are strict subsets of ARTIFACTSBENCH's 1,825 human-validated tasks, we directly inherit this established validity. We use the official evaluation scripts, judge prompts, and scoring rubric without modification. To mitigate on-policy overfitting, we decouple training-time critic (Qwen2.5-VL-72B-Instruct) from offline evaluator (Gemini-2.5-Pro). We do not re-run human studies; validity is anchored in ARTIFACTSBENCH's reported evidence.

Score Interpretation. The absolute VisualScore range (20-30 for 7B/8B models) reflects benchmark difficulty: ARTIFACTSBENCH tasks span complex interactions, dynamic animations, and pixel-perfect layouts. Reference models (GPT-4o: 33, Qwen2.5-32B: 26) establish that even frontier systems find these tasks challenging. The $[0, 100]$ scale provides fine-grained discrimination; we report relative improvements over baselines.

C Pseudocode for ReLook

D Evaluator rubric and scoring range

We use a fixed evaluation rubric for VISUALSCORE. During training, the critic's visual score used for

Critic Model	A-Lite	A-Easy	A-Game	A-SVG	A-Web	A-Si	FS-Html	Web-Bench
Qwen2.5-VL-72B-Instruct	27.88	34.12	26.72	20.92	28.31	26.36	67.50	4.20
Qwen3-VL-8B-Instruct	28.96	34.02	28.43	21.77	28.84	27.94	68.75	5.48
Qwen3-VL-32B-Instruct	**32.95**	**35.47**	**31.95**	**29.75**	**34.02**	**30.83**	**72.50**	**8.35**

Table 3: Ablation study on the Critic Model. We compare the performance of the base model (Qwen2.5-7B-Instruct-ReLook) when trained with guidance from different critic models. The results demonstrate that stronger critics lead to better performance.

Configuration	A-Lite Score
Base Model (Qwen2.5-7B-Instruct)	21.59
Web-RL (vision RL, no reflection)	24.89
ReLook (Single-sample FO)	26.98
ReLook (Full FO, up to 10 samples)	27.88

Table 4: ArtifactsBench-Lite results under controlled sampling budgets.

RL is normalized to $[0, 1]$ for stability. For offline reporting and tables, we follow ARTIFACTSBENCH and report evaluator outputs on a $[0, 100]$ scale; all numbers in the main results tables and figures are on this $[0, 100]$ scale unless otherwise specified. The rubric jointly considers: (i) specification adherence; (ii) layout alignment and spatial fidelity; (iii) typography and color coherence; and (iv) interactive integrity for tasks specifying actions. For dynamic behavior, each reflection round is evaluated on three temporal screenshots jointly in a single scoring call to obtain one round-level score; trajectory-level reward averages round-level scores. We do not perform multi-judge averaging or smoothing.

E The loss function for Forced Optimization

Faced with the "behavior collapse" problem, we first attempted a negative-reward penalty comparing post-feedback scores to pre-feedback scores. If an optimized answer was worse than its predecessor, a negative reward was applied. We observed that this encouraged the model to reduce reflection frequency to avoid penalties (reward hacking), degrading ReLook into regular RL and harming performance. Therefore, we removed this mechanism and adopted the strict acceptance rule described in Method: only strictly improving steps are accepted into trajectories.

F Lightweight distillation loss on feedback tokens

Let M denote the index set of critic feedback tokens m within a trajectory, and let the frozen MLLM critic define a teacher distribution $q_t(\cdot)$ at each position $t \in M$. The student policy distribution is $p_\theta(\cdot \mid x_{\le t})$. The lightweight distillation loss used to transfer the feedback style for test-time self-reflection is

$$\mathcal{L}^{m}_{\text{distill}} := \frac{1}{|M|} \sum_{t \in M} \mathrm{KL}\big(q_t(\cdot) \,\|\, p_\theta(\cdot \mid x_{\le t})\big) \quad (6)$$

equivalently, as a teacher-forced cross-entropy

$$\mathcal{L}^{m}_{\text{distill}} := -\frac{1}{|M|} \sum_{t \in M} \sum_{v} q_t(v)\, \log p_\theta(v \mid x_{\le t}) \quad (7)$$

Only policy-controlled tokens (t, c) contribute non-zero advantages in GRPO; feedback tokens m are masked in the RL objective (advantages set to zero). The total training objective is

$$\mathcal{L} = -\mathcal{J}^{t,c}_{\text{GRPO}} + \gamma\, \mathcal{L}^{m}_{\text{distill}}, \quad \text{with default } \gamma{=}0.1,\ \gamma \in [0.05, 0.3]. \quad (8)$$

G Reproducibility notes: GRPO and implementation details

Reference policy and KL. We regularize toward a fixed reference policy π_{ref} (the frozen instruction-tuned backbone before RL). KL is computed token-wise over policy-controlled tokens with weight β (see ranges in Experiment); we do not anneal β within a step.

Trajectory composition and masking. Trajectories mix policy tokens (t, c) and critic tokens (m). During optimization, advantages on m are masked to zero; optional lightweight distillation on m uses a small KL/CE with weight γ to the frozen MLLM outputs.

Algorithm 1 ReLook Training Framework

Require: Task specification q, base policy π_θ, vision critic MLLM (Qwen2.5-VL-72B), max steps S, max reflections R, max resampling K=10

Ensure: Optimized policy π^*_θ with internalized visual cognition

Initialize: $s \leftarrow 0$, $history \leftarrow [q]$
while $s < S$ **do**
  Initialize: $r \leftarrow 0$, $o \leftarrow$ "", $s_{\text{prev}} \leftarrow -1$
  while $r < R$ and room for improvement **do**
    $k \leftarrow 0$, $accepted \leftarrow$ False
    while $k < K$ and not $accepted$ **do**
      {Forced Optimization: resample up to K times} **Generate:** $t_r, c_r \leftarrow \pi_\theta$(next token$|history$) Extract code c_r from `<answer>` block Attempt rendering $c_r \rightarrow$ Capture screenshots $\{S_1, S_2, S_3\}$ Generate feedback $m_r \leftarrow$ MLLM$(q, c_r, \{S_i\})$ Compute visual score $s_r \leftarrow$ VisualScore$(q, c_r, \{S_i\})$ **if** $s_r > s_{\text{prev}}$ **then** {Accept only strictly improving steps}
      Append t_r, c_r to $history$ and o
      Append m_r to $history$ and o within `<mllm_feedback>`
      $s_{\text{prev}} \leftarrow s_r$, $accepted \leftarrow$ True
    end if
    $k \leftarrow k + 1$
  end while
  if not $accepted$ **then**
    {No improvement after K attempts} Break {Terminate further reflections, use best-so-far}
  end if
  $r \leftarrow r + 1$
end while
Calculate reward components for the trajectory o:
  $R_{\text{MLLM}}(o) = \begin{cases} \text{VisualScore}(o) & \text{if screenshot valid} \\ 0 & \text{otherwise} \end{cases}$.
  $R_{\text{len}}(o)$ as in Method (linear penalty: L_{start}=12,000, L_{end}=14,000).
  $R_{\text{ReLook}}(o) = R_{\text{MLLM}}(o) \cdot R_{\text{len}}(o)$.
Perform policy update using GRPO to optimize parameters θ based on R_{ReLook}.
If convergence criterion is met (no significant improvement), exit loop.
$s \leftarrow s + 1$
end while
return π^*_θ for deployment {Optionally discard MLLM during inference.}

Advantage baseline and clipping. We use a group-relative baseline (mean return over G trajectories per query); advantages are standardized within-batch and clipped to $[-2, 2]$.

Sampling limits and rejection handling. For Forced Optimization, we cap resampling attempts at 10 per reflection round to avoid infinite loops; non-improving proposals are rejected without rescoring. When the 10-attempt limit is reached without improvement, we terminate further reflection rounds for that trajectory and use the best-so-far result. This limit balances exploration (allowing sufficient attempts to find improving revisions) with computational efficiency.

Length penalty parameters. We set $L_{\text{start}} =$ 12,000 and $L_{\text{end}} =$ 14,000 tokens based on empirical analysis of typical front-end code lengths in our training corpus, ensuring reasonable generation length while discouraging degenerate repetition.

Valid Render Rate dynamics. During training, the Valid Render Rate increases from approximately 40% at initialization to approximately 80% upon convergence, demonstrating that the model learns to produce syntactically valid and renderable code through the combination of the visual-output constraint in prompts and the zero-reward penalty for invalid renders.

Decoding and seeds. Unless otherwise noted, decoding uses identical hyperparameters across systems (temperature=1.0, top-p=0.7) with three random seeds; fixtures are cached to avoid network variance. These notes complement Section 2 and Section 3 to facilitate faithful reimplementation.

H Qualitative Analysis

Figure 6 presents a qualitative comparison between the baseline model and ReLook across several representative front-end generation tasks.

Prompt column lists natural-language instructions of varying complexity, including layout composition, template library rendering, login/registration forms, and a chessboard.

Base Model column shows the outputs of an instruction-tuned baseline. While it can produce code that renders without error, the resulting webpages often suffer from issues such as layout drift, incomplete functionality, and lack of visual coherence (e.g., missing interactivity in the login page, overly simplistic rendering of the chessboard).

ReLook column demonstrates the effect of our vision-grounded reinforcement learning framework. By incorporating visual feedback into training, ReLook produces outputs that are not only executable but also visually faithful and functionally aligned with the prompts. For instance, the template library page is correctly populated with clickable cards, the login/registration form has a clean layout and interactive elements, and the chessboard renders with precise alignment.

Overall, the comparison highlights how ReLook systematically reduces layout drift, strengthens interaction correctness, and achieves higher visual fidelity compared to the baseline.

I Template prompt for ReLook rollout

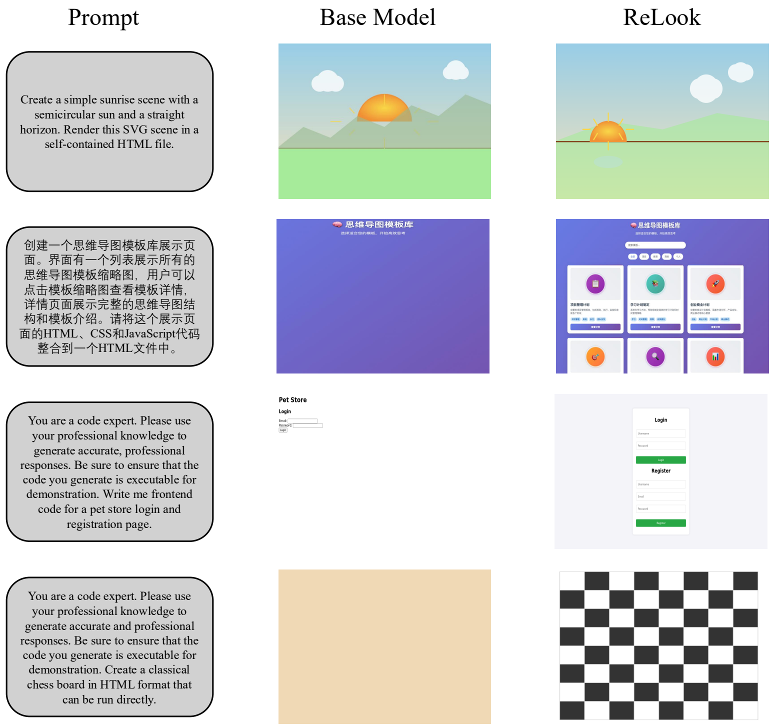

Figure 6: Visual Comparison of Frontend Websites Generated by Baseline and ReLook.

Solve the following problem step by step.
You now have the ability to selectively write executable HTML, CSS, JavaScript, or SVG code to receive feedback from the multimodal large model on the code.
The code you provided will be executed, and the feedback (wrapped in '<mllm_feedback> output_str <mllm_feedback>') can be returned to aid your reasoning and help you arrive at the final answer.
Unless you believe the current answer is flawless, please output <get_feedback> after providing the complete answer to receive feedback from the multimodal large model and improve the code based on the feedback.
user question:
{$query}

Figure 7: Template prompt for ReLook rollout.