STEPWEB: TRAJECTORY-CONSERVING STEP-LEVEL CREDIT FOR MULTI-TURN WEB REFINEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

Multi-turn webpage refinement requires agents to improve their own implementations while operating on states shaped by earlier edits. Terminal rewards can identify successful trajectories, but provide limited information about which intermediate decisions caused regression, recovered lost quality, or established genuine progress. We introduce STEPWEB, a trajectory-aware reinforcement learning framework that separates terminal trajectory comparison from within-trajectory credit assignment. Its trajectory-conserving step-level credit (TC-SLC) assigns history-aware, magnitude-sensitive signals to Regression, Repair, and New Gain events, and redistributes them across refinement turns through token-length-weighted centering while preserving the trajectory's mean advantage before policy clipping. Experiments on Vision2Web Level 1, Design2Code, Design2Code-HARD, and Interaction2Code show consistent improvements over the Qwen3.5-9B backbone, yielding a 16.4% macro-averaged relative gain across the four benchmarks. In controlled multi-turn refinement on Vision2Web Level 1, TC-SLC further reduces the edit regression rate by 29.5% relative to terminal-only RL while improving final reconstruction quality. These results demonstrate the effectiveness of trajectory-local credit assignment for reliable iterative visual coding.

1 INTRODUCTION

Multimodal models have made substantial progress in translating screenshots into executable web interfaces, driven by large-scale screenshot–code data and increasingly capable coding models (Laurençon et al., 2024; Yun et al., 2024; Gui et al., 2025a; Wan et al., 2025; Gui et al., 2025b; Yue et al., 2025). Yet high-fidelity reconstruction is rarely achieved in a single pass (Madaan et al., 2023; Chen et al., 2023; Zhong et al., 2024). In practice, webpage generation is naturally iterative: an agent produces an implementation, inspects the rendered result, identifies discrepancies, and revises the code accordingly. Unlike one-shot generation, however, each revision changes the state inherited by all subsequent turns. The challenge is therefore not only to generate a good webpage, but also to determine which intermediate edits caused regressions, recovered lost quality, or established genuine progress.

Recent work has increasingly incorporated rendered feedback into UI generation (Sansford et al., 2026; Duan et al., 2025; Guo et al., 2026). UICoder learns from automatically evaluated visual feedback (Wu et al., 2024), VisRefiner trains models to correct rendered discrepancies (Deng et al., 2026), UI2Code^{N} performs interactive visual refinement (Yang et al., 2026), and WebGen-Agent combines screenshot feedback, GUI interaction, and reinforcement learning for iterative website generation (Lu et al., 2026; 2025a;b; Gehring et al., 2025; Qi et al., 2024). These approaches establish the value of refinement, but multi-turn optimization introduces a different credit-assignment problem. Independently sampled trajectories quickly diverge in both code and rendered state, making same-turn intermediate scores poor state-matched comparisons. Normalizing such scores across trajectories can therefore confound edit quality with differences inherited from earlier states. At the other extreme, assigning every turn the same terminal reward preserves the trajectory-level objective but cannot identify which individual refinements were helpful or harmful (Kazemnejad et al., 2024; Cui et al., 2025; Wang et al., 2025).

We address this tension by treating screenshot-to-code refinement as a dependent trajectory rather than a collection of isolated corrections. Each rollout begins from an independently sampled reconstruction

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.

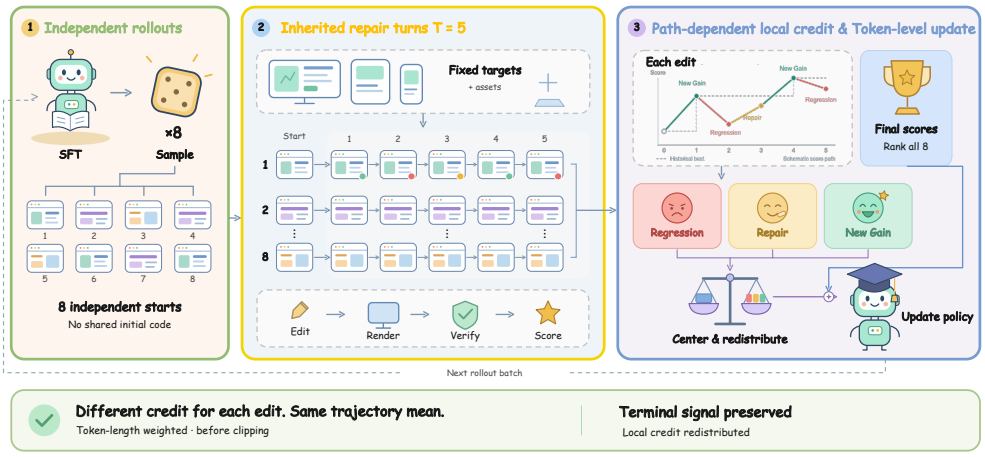


Figure 1: Overview of StepWeb. Independent trajectories evolve through their own rendered states. Terminal rewards rank complete trajectories, while history-aware local signals redistribute credit across refinement turns without changing the mean trajectory advantage.

and then evolves through its own browser-rendered states. At every refinement turn, the agent observes the reference design, its current implementation and rendering, and correction feedback from a frozen VLM feedback generator. The revised webpage is then executed, rendered, and scored before becoming the input to the next turn. This formulation preserves diversity across trajectories while exposing the sequential dependencies that make local credit assignment necessary.

We introduce StepWeb, a trajectory-aware reinforcement learning framework that separates *trajectory quality* from *within-trajectory credit*. Terminal webpage scores rank complete rollouts through a group-relative trajectory advantage (Shao et al., 2024; Ahmadian et al., 2024; Luo et al., 2025). For refinement turns, StepWeb further characterizes local score evolution as *Regression*, when an edit decreases quality; *Repair*, when it recovers previously attained quality; and *New Gain*, when it establishes a new historical best. These history-aware events yield magnitude-sensitive local credits, which are centered within each trajectory before being combined with the terminal advantage. The resulting trajectory-conserving step-level credit (TC-SLC) redistributes supervision across refinement turns while preserving the token-weighted mean trajectory advantage.

Our contributions are threefold. First, we formulate screenshot-to-code refinement as a dependent multi-turn process in which independently sampled implementations evolve through their actual rendered states. Second, we introduce TC-SLC, which combines terminal trajectory comparison with history-aware Regression, Repair, and New Gain signals while conserving trajectory-level credit. Third, we evaluate StepWeb on Vision2Web Level 1, Design2Code, Design2Code-HARD, and Interaction2Code, demonstrating consistent gains over the Qwen3.5-9B backbone together with fewer harmful regressions during multi-turn refinement.

2 Related Work

Screenshot-to-code generation and evaluation. Screenshot-to-code has evolved from task-specific models (Beltramelli, 2017) to multimodal models trained on scalable webpage data (Laurençon et al., 2024; Yun et al., 2024; Gui et al., 2025a). Recent systems improve generation through spatial decomposition, hierarchical generation, and multi-agent coordination (Wan et al., 2025; Gui et al., 2025b; Jiang et al., 2026; Yue et al., 2025). Benchmarks evaluate structural and perceptual fidelity (Si et al., 2025), interactive code generation (Xiao et al., 2025), functional websites (Lu et al., 2026; 2025a;b; He et al., 2026), and iterative visual refinement (Xu et al., 2026). Unlike work centered on the final implementation, we study how an agent repairs its own webpage across dependent rendering–editing turns.

Rendering-aware refinement. Recent methods improve generated interfaces using compiler, multimodal, or rendered feedback (Wu et al., 2024; Deng et al., 2026; Yang et al., 2026). Related

work uses visual critics for frontend refinement (Sansford et al., 2026), multi-level screenshot and interaction feedback (Lu et al., 2026; 2025a;b), visual self-distillation objectives (Dong, 2026), grounded design critique (Duan et al., 2025), and iterative layout correction (Guo et al., 2026). General-purpose self-correction methods likewise alternate generation, critique, and revision using model-generated or tool-grounded feedback (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024). Most such methods optimize individual corrections or states; we model the full inherited trajectory and preserve its terminal preference.

Execution-guided code refinement. Execution feedback has long been used to rank, repair, or optimize generated programs. CodeT generates tests for candidate selection (Chen et al., 2022), while CodeRL and RLTF learn from execution outcomes and fine-grained unit-test signals (Le et al., 2022; Liu et al., 2023). Self-debugging methods use execution traces, explanations, or learned debugging behavior to revise failed programs (Chen et al., 2023; Zhong et al., 2024; Jiang et al., 2024). StepCoder masks unexecuted regions for finer-grained RL updates (Dou et al., 2024), and RLEF explicitly trains code models to exploit multi-turn execution feedback (Gehring et al., 2025). Real-repository benchmarks and agents further expose the long-horizon dependencies of iterative code editing (Jimenez et al., 2024; Yang et al., 2024). These methods motivate feedback-driven repair; our focus is conserving trajectory-level preference while assigning credit to visual web edits.

3 Problem Formulation

Let a task $x = (P, A, V)$ contain prototype screenshots $P = \{P^v\}_{v \in V}$, local assets A, and required viewports V. For Vision2Web Level 1, V contains desktop, tablet, and mobile; all three views belong to one task. A deterministic environment applies code, launches a pinned browser, and returns renders and verification artifacts:

$$I(C) = \mathcal{R}(C; A, V), \qquad s(C) = \mathcal{S}(I(C), P) \in [0, 1]. \tag{1}$$

The scalar s is used only for training credit and turn tracking. At evaluation, every benchmark retains its native metrics.

For task x, the rollout policy independently samples K initial implementations,

$$C_{i,0} \sim \pi_{\theta_{\text{old}}}(\cdot \mid x), \qquad i = 1, \ldots, K, \tag{2}$$

then renders $I_{i,0} = I(C_{i,0})$ and obtains $s_{i,0} = s(C_{i,0})$. Crucially, the branches do not share a pre-generated C_0. For repair turn $t \in \{1, \ldots, T\}$, the observable state is

$$o_{i,t} = \big(P, A, C_{i,t-1}, I_{i,t-1}, s_{i,t-1}, b_t\big), \tag{3}$$

where b_t optionally records the remaining budget. The policy emits a complete updated implementation by default,

$$a_{i,t} \sim \pi_{\theta_{\text{old}}}(\cdot \mid o_{i,t}), \quad C_{i,t} = \text{Apply}(C_{i,t-1}, a_{i,t}), \tag{4}$$

and the environment produces $I_{i,t}$, $s_{i,t}$, and a verifier record. A structured diff is an output-representation ablation, not a requirement of the main method. The main setting uses $K = 8$ and exactly $T = 5$ repair turns; early stopping is evaluated separately.

A trajectory is $\tau_i = \{C_{i,0}, I_{i,0}, s_{i,0}; o_{i,t}, a_{i,t}, C_{i,t}, I_{i,t}, s_{i,t}\}_{t=1}^{T}$. Future scores, other branches, group statistics, path labels, and advantages are hidden from the policy. We optimize final fidelity while assigning turn-level credit to the inherited edits that produced it.

4 Method

4.1 StepWeb Pipeline

StepWeb consists of three stages. **Initial reconstruction SFT** learns to generate a webpage directly from the reference screenshot and reconstruction instruction, $P \to C_0$. **Refinement SFT** optionally introduces verified same-page correction transitions,

$$(P, C_{t-1}, I_{t-1}, s_{t-1}, f_{t-1}) \to C_t, \qquad t \geq 1,$$

where I_{t-1} is the rendering of the current implementation, s_{t-1} is its visual score, and f_{t-1} is correction feedback produced by a frozen VLM feedback generator. Finally, **online multi-turn RL** samples complete editing trajectories and optimizes them using the credit assignment described below.

The initial generation differs from the subsequent refinement process. For trajectory i, the policy first produces an initial reconstruction

$$C_{i,0} \sim \pi_\theta(\cdot \mid P). \tag{5}$$

After rendering $C_{i,0}$, the environment obtains $I_{i,0}$ and its score $s_{i,0}$. For each of the T refinement turns,

$$C_{i,t} \sim \pi_\theta(\cdot \mid P, C_{i,t-1}, I_{i,t-1}, s_{i,t-1}, f_{i,t-1}), \qquad t = 1, \ldots, T. \tag{6}$$

Thus, T0 denotes the initial reconstruction and T1–TT denote rendering-conditioned refinements.

Given these trajectories, TC-SLC combines terminal outcomes with within-trajectory score changes to assign turn-level credit. It computes a trajectory-level advantage, redistributes credit among refinement turns while preserving that overall preference, and applies the resulting advantages to token-level policy optimization.

4.2 Terminal Trajectory Advantage

For each task, we independently sample K complete trajectories,

$$C_{i,0} \to C_{i,1} \to \cdots \to C_{i,T}, \qquad i = 1, \ldots, K. \tag{7}$$

Importantly, the initial reconstructions are sampled independently across trajectories, preserving diversity in both starting implementations and subsequent editing paths.

We use the final webpage score as the trajectory reward:

$$R_i = s_{i,T}. \tag{8}$$

The group-relative trajectory advantage is

$$A_i^{\mathrm{traj}} = \frac{R_i - \mu_R}{\sigma_R + \epsilon}, \qquad \mu_R = \frac{1}{K} \sum_{j=1}^{K} R_j. \tag{9}$$

This advantage ranks *complete solutions* to the same task. All turns in trajectory i inherit the same terminal trajectory preference; in particular, the initial reconstruction receives credit according to the final outcome of the trajectory rather than its initial score $s_{i,0}$:

$$A_{i,0} = A_i^{\mathrm{traj}}. \tag{10}$$

This distinction is important because trajectories need not share the same intermediate state. After independently sampling their initial reconstructions, branches may differ substantially in both code and rendering. We therefore apply group-relative normalization only to terminal outcomes and avoid interpreting same-index intermediate states as state-matched actions.

4.3 Trajectory-Local Path Credit

Terminal credit captures whether a complete trajectory is successful, but cannot distinguish which refinement steps contributed to that success. We therefore introduce a trajectory-local signal for refinement turns $t = 1, \ldots, T$.

Let

$$h_{i,t-1} = \max_{0 \le u < t} s_{i,u} \tag{11}$$

denote the best score reached before turn t. Each non-tied refinement transition is assigned one of three path events:

$$\begin{aligned} \textsc{Regression}: &\quad s_{i,t} < s_{i,t-1}, \\ \textsc{Repair}: &\quad s_{i,t} > s_{i,t-1} \ \wedge \ s_{i,t} \le h_{i,t-1}, \\ \textsc{NewGain}: &\quad s_{i,t} > h_{i,t-1}. \end{aligned} \tag{12}$$

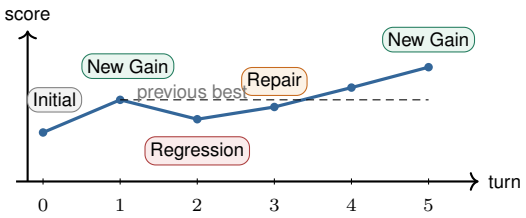


Figure 2: Trajectory-local path events. T0 is the initial reconstruction; subsequent turns are classified as Regression, Repair, or New Gain according to their score evolution.

A REGRESSION directly degrades the current webpage. A REPAIR improves the current state but only recovers quality that was already achieved earlier in the trajectory. A NEWGAIN establishes a new historical best. This distinction allows us to separate genuine progress from recovery after an earlier mistake.

We weight each event by its score magnitude:

$$d^c_{i,t} = |s_{i,t} - s_{i,t-1}| \, \mathbb{I}[(i,t) \text{ has class } c] \, . \tag{13}$$

The resulting local credit is

$$q_{i,t} = -\lambda_{\text{reg}} d^{\text{reg}}_{i,t} + \lambda_{\text{repair}} d^{\text{repair}}_{i,t} + \lambda_{\text{new}} d^{\text{new}}_{i,t}, \qquad t = 1, \ldots, T, \tag{14}$$

where $\lambda_{\text{new}} > \lambda_{\text{repair}} \geq 0$ and $\lambda_{\text{reg}} > 0$.

The initial reconstruction at $t = 0$ receives no local path label because no preceding webpage state exists. Its score $s_{i,0}$ instead serves as the starting point from which later refinement progress is measured.

4.4 TRAJECTORY-CONSERVING CREDIT REDISTRIBUTION

Directly adding $q_{i,t}$ to the policy advantage would change not only the relative importance of individual turns, but also the overall preference assigned to the trajectory. We instead use local credit only to *redistribute* the terminal trajectory advantage.

Let $L_{i,t}$ denote the number of valid generated tokens at turn t. We center local credit over the refinement portion of trajectory i:

$$\bar{q}_i = \frac{\sum_{t=1}^{T} L_{i,t} q_{i,t}}{\sum_{t=1}^{T} L_{i,t}}, \qquad \widetilde{q}_{i,t} = q_{i,t} - \bar{q}_i. \tag{15}$$

The final advantage assigned to turn t is

$$A_{i,t} = \begin{cases} A^{\text{traj}}_i, & t = 0, \\ A^{\text{traj}}_i + \beta \dfrac{\widetilde{q}_{i,t}}{\delta_{\text{scale}}}, & t = 1, \ldots, T, \end{cases} \tag{16}$$

where β controls the strength of local redistribution and $\delta_{\text{scale}} > 0$ controls its numerical scale.

This construction preserves the trajectory-level preference. Since

$$\sum_{t=1}^{T} L_{i,t} \widetilde{q}_{i,t} = 0, \tag{17}$$

we have

$$\frac{\sum_{t=0}^{T} L_{i,t} A_{i,t}}{\sum_{t=0}^{T} L_{i,t}} = A^{\text{traj}}_i. \tag{18}$$

Hence, the terminal reward determines *how much* credit a trajectory receives, while the local signal determines *where* that credit is placed among its refinement turns. The initial reconstruction remains governed solely by the terminal outcome.

Algorithm 1 One StepWeb training update

for $i = 1, \ldots, K$ **do**
  Sample initial reconstruction $C_{i,0}$ from P
  Render and obtain $s_{i,0}$
  for $t = 1, \ldots, T$ **do**
    Obtain VLM feedback $f_{i,t-1}$ from the current render
    Sample revised implementation $C_{i,t}$
    Render, verify, and obtain $s_{i,t}$
  end for
end for
Compute $R_i = s_{i,T}$ and trajectory advantages A_i^{traj}
Set $A_{i,0} = A_i^{\mathrm{traj}}$
for each refinement turn $t = 1, \ldots, T$ **do**
  Classify Regression, Repair, or New Gain
  Compute local credit $q_{i,t}$
end for
Center local credits within each trajectory
Compute $A_{i,t}$ using Equation (16)
Update generated tokens using Equation (20)

4.5 Token-Level Policy Optimization

For generated token $y_{i,t,k}$, we define the importance ratio

$$\rho_{i,t,k} = \exp(\log \pi_\theta(y_{i,t,k} \mid h_{i,t,k}) - \log \pi_{\theta_{\mathrm{old}}}(y_{i,t,k} \mid h_{i,t,k})) \,. \tag{19}$$

We optimize the clipped surrogate objective

$$\mathcal{L}_{\mathrm{actor}} = -\frac{1}{\sum_{i,t} L_{i,t}} \sum_{i,t,k} m_{i,t,k} \min \Big[\rho_{i,t,k} A_{i,t}, \; \mathrm{clip}(\rho_{i,t,k}, 1-\eta, 1+\eta) A_{i,t} \Big], \tag{20}$$

together with a KL penalty to the SFT reference policy. The mask $m_{i,t,k}$ excludes prompt, padding, and environment tokens.

All generated code tokens are optimized. Each turn-level advantage is broadcast to its valid generated tokens; T0 receives only terminal credit, while refinement turns also receive the centered local signal. We do not estimate within-turn token-level causal credit. Since no learned value model is used, we refer to the update as PPO/GRPO-style rather than PPO with generalized advantage estimation.

5 Experiments

We evaluate StepWeb on responsive webpage reconstruction, component-level fidelity, and interactive refinement. Tables 1 and 2 compare its reconstruction and interaction performance, while Table 3 examines the contribution of trajectory-local credit assignment.

5.1 Experimental Setup

Benchmarks and Evaluation Metrics. Vision2Web Level 1 (He et al., 2026) evaluates static webpage reconstruction across desktop, tablet, and mobile viewports, reporting viewport-specific scores and average over the three device types. For Vision2Web, all compared models use Claude Code (Anthropic, 2025b) as the common coding framework. Design2Code (Si et al., 2025) reports Block, Text, Position, Color, and CLIP, measuring component recovery, textual fidelity, spatial alignment, color consistency, and overall visual-semantic similarity, respectively. Design2Code-HARD (Deng et al., 2026) applies the same metrics to denser and more challenging layouts. We additionally evaluate interactive webpage generation on Interaction2Code (Xiao et al., 2025), using full-page CLIP, SSIM, and Text together with interaction-region CLIP, SSIM, Text, Position, and Implement Rate (IR). The reconstruction benchmarks use a 0–100 scale, whereas Interaction2Code retains its original 0–1 scale.

General-Purpose and Specialized Baselines. We compare with proprietary models GPT-4o (OpenAI, 2024), GPT-5 (Singh et al., 2025), Gemini-2.5-Pro (Comanici et al., 2025), and Claude-4-Sonnet (Anthropic, 2025a), together with open-source vision-language models Qwen2.5-VL-72B (Bai et al., 2025), Qwen3.5-9B (Qwen Team, 2026a), Qwen3.6-35B-A3B (Qwen Team, 2026b), and Qwen3.8-27B (Qwen Team, 2026c). We further include the available Kimi results for each benchmark. Specialized baselines include VisRefiner-7B (Deng et al., 2026) in single-step and self-refinement modes, UI2CodeN (Yang et al., 2026) after supervised fine-tuning (SFT) and reinforcement learning (RL), and ScreenCoder (Jiang et al., 2026) in agentic and finetuned configurations. Qwen3.5-9B, marked with †, is the backbone used by StepWeb.

Trajectory Metrics and Controlled Comparisons. For the refinement ablations, T0 denotes the pre-refinement score, Final the score of the final state, and Best the highest mean score among the reported turns. All trajectory scores use the same scale as the Avg metric in Table 1. T0 denotes the initial state of each trained variant and should not be interpreted as the raw Qwen3.5-9B baseline. Best−Final measures post-peak degradation; Reg. is the percentage of edits that decrease the score, and Valid is the percentage of executable trajectories. Terminal RL uses terminal trajectory credit without local redistribution. Unless otherwise specified, all variants use the same rollout, rendering, evaluation, and sampling budgets.

5.2 Main Results

Table 1: Comparison on Vision2Web Level 1, Design2Code, and Design2Code-HARD. All scores are on a 0–100 scale; higher is better. Desk., Tab., Mob., Pos., and Avg denote Desktop, Tablet, Mobile, Position, and the average over the three device types. Green shading indicates the best result in each column, including ties; boldface highlights StepWeb. † denotes the Qwen3.5-9B backbone.

Model	Vision2Web (+ Claude Code)				Design2Code (Standard)					Design2Code (Hard)				
	Desk.	Tab.	Mob.	Avg	Block	Text	Pos.	Color	CLIP	Block	Text	Pos.	Color	CLIP
Qwen2.5-VL-72B	42.1	34.0	29.1	35.1	90.6	96.4	77.5	78.1	87.9	75.0	88.2	71.5	73.2	86.6
GPT-4o	43.9	36.9	32.8	37.9	93.0	98.2	85.5	84.1	90.4	56.6	89.8	78.6	81.9	87.1
GPT-5	45.3	44.2	39.9	43.1	89.1	94.2	86.4	78.0	81.6	76.6	89.3	81.3	73.3	79.4
Gemini-2.5-Pro	41.2	32.6	27.3	33.7	89.1	93.5	85.5	71.4	80.9	76.6	88.6	80.4	66.7	78.7
Claude-4-Sonnet	41.1	32.4	27.0	33.5	88.7	93.2	84.6	72.0	80.5	76.2	88.3	79.5	67.3	78.3
Qwen3.5-9B†	40.7	31.7	26.2	32.9	74.7	89.0	78.8	83.6	88.2	58.9	83.6	67.6	78.7	86.3
Qwen3.6-35B-A3B	42.9	35.3	30.8	36.3	85.9	90.0	84.6	86.9	92.3	65.5	74.8	71.0	71.3	88.4
Qwen3.8-27B	45.1	38.8	35.3	39.7	91.7	96.2	92.3	91.0	93.6	83.3	92.6	88.6	88.1	92.8
Kimi-K2.6	57.3	52.7	50.8	53.6	90.5	94.9	92.9	90.9	91.8	81.4	90.0	87.8	85.1	90.7
VisRefiner-7B (Single-step)	43.2	35.7	31.4	36.8	90.6	97.5	81.7	83.1	90.1	78.1	96.2	82.3	83.7	87.9
VisRefiner-7B (Self-refinement)	43.5	36.2	31.9	37.2	91.4	98.1	82.3	83.8	90.7	80.8	96.2	82.2	84.8	87.7
UI2CodeN-SFT	40.2	30.9	25.2	32.1	86.8	91.5	81.7	69.7	79.0	74.3	86.6	76.6	65.0	76.8
UI2CodeN-RL	41.1	32.3	27.0	33.5	88.7	93.1	83.8	72.6	80.5	76.2	88.2	78.7	67.9	78.3
ScreenCoder (Agentic)	44.6	38.1	34.3	39.0	86.5	97.5	92.5	90.8	92.2	74.0	92.6	87.4	86.1	90.0
ScreenCoder (Finetuned)	44.1	37.2	33.2	38.2	84.9	96.8	91.3	88.6	91.5	72.4	91.9	86.2	83.9	89.3
StepWeb	**45.2**	**39.0**	**32.4**	**37.7**	**93.8**	**96.3**	**88.5**	**92.2**	**95.3**	**85.1**	**93.1**	**88.4**	**87.5**	**91.9**

Table 2: Comparison on Interaction2Code. We report full-page and interaction-region fidelity together with Implement Rate (IR). All scores are on a 0–1 scale; higher is better. Green shading indicates the best result in each column; boldface highlights StepWeb. † denotes the Qwen3.5-9B backbone.

Model	Full Page			Interaction Region				
	CLIP	SSIM	Text	CLIP	SSIM	Text	Position	IR
Qwen2.5-VL-72B	0.6430	0.4234	0.4197	0.4624	0.3207	0.2450	0.3950	0.6524
GPT-4o	0.7114	0.5277	0.5147	0.5605	0.4149	0.3590	0.4888	0.7754
Qwen3.5-9B†	0.6139	0.5133	0.5884	0.5652	0.4071	0.3827	0.3402	0.7460
Qwen3.6-35B-A3B	0.6845	0.5753	0.6720	0.6377	0.4487	0.4823	0.3823	0.8102
Qwen3.8-27B	0.6821	0.5714	0.6727	0.6297	0.4548	0.4617	0.3723	0.8102
Kimi-K2.6	0.6648	0.5536	0.6602	0.6160	0.4385	0.4376	0.3535	0.7796
StepWeb	**0.6830**	**0.5730**	**0.6740**	**0.6320**	**0.4580**	**0.4660**	**0.3750**	**0.8310**

Table 3: Ablation on Vision2Web Level 1 using controlled checkpoints. All trajectory scores are reported on the same scale as the Avg metric in Table 1. T0 and Final denote the initial and final mean scores. Best is the highest mean score among the reported turns. Best−Final measures post-peak degradation; Reg. and Valid report edit regression and trajectory executability rates (%). Bold and underline indicate the best and second-best distinct values.

Variant	T0 ↑	Final ↑	Best ↑	Best−Final ↓	Reg. (%) ↓	Valid (%) ↑
Terminal RL	33.5	35.2	37.4	2.2	28.1	98.5
w/o trajectory centering	33.6	36.0	37.8	1.8	25.4	98.6
Repair = New Gain	33.7	36.4	38.0	1.6	23.9	98.7
Binary local credit	33.7	36.9	<u>38.5</u>	1.6	21.8	98.8
Structured patch	**33.8**	<u>37.2</u>	<u>38.5</u>	**1.3**	<u>20.6</u>	**99.3**
StepWeb	**33.8**	**37.7**	**39.2**	<u>1.5</u>	**19.8**	<u>99.0</u>

Consistent Gains over the Backbone. StepWeb improves every reported metric over its Qwen3.5-9B backbone across Vision2Web Level 1, Design2Code, Design2Code-HARD, and Interaction2Code. Using the benchmark-level aggregation defined above, these gains correspond to a 16.4% macro-averaged relative improvement. The improvements span responsive reconstruction, structural and perceptual fidelity, and interactive generation.

Stronger Structural Generalization under Complexity. The largest static gains are concentrated in structural fidelity. StepWeb raises Block from 74.7 to 93.8 on Design2Code and from 58.9 to 85.1 on Design2Code-HARD, achieving the highest Block score in both settings. It also obtains the highest CLIP score on the standard split at 95.3. The larger structural gain on the harder split suggests that trajectory-based refinement is particularly useful when reconstruction requires coordinated corrections across densely coupled components.

Reliable Gains in Interactive Refinement. The same improvement pattern extends to Interaction2Code, where StepWeb exceeds Qwen3.5-9B on all eight reported metrics. Implement Rate increases from 0.7460 to 0.8310, while both full-page and interaction-region similarity metrics improve consistently. StepWeb further achieves the highest full-page Text, interaction-region SSIM, and Implement Rate among the compared methods.

5.3 Ablation Study

More Reliable Multi-Turn Refinement. Compared with Terminal RL, StepWeb raises the final score from 35.2 to 37.7 while the initial scores remain nearly unchanged (33.5 vs. 33.8). It reduces the regression rate from 28.1% to 19.8%, a 29.5% relative reduction, and narrows the Best−Final gap from 2.2 to 1.5. The improvement emerges primarily during refinement rather than initialization.

History-Aware and Magnitude-Sensitive Credit. Removing trajectory centering reduces Final from 37.7 to 36.0. Treating Repair and New Gain identically lowers it to 36.4, while replacing magnitude-aware credit with binary labels reaches 36.9. All three variants also increase the regression rate relative to the full method, supporting history-aware, and magnitude-sensitive local credit.

Action Flexibility and Execution Validity. Structured patches achieve the highest validity of 99.3% and the smallest Best−Final gap of 1.3, compared with 99.0% and 1.5 for full-code actions. Full-code actions nevertheless attain higher Final and Best scores, 37.7 and 39.2 versus 37.2 and 38.5, and a lower regression rate of 19.8% versus 20.6%. This indicates a trade-off between execution stability and editing flexibility.

6 Analysis and Discussion

We use process-level evidence to examine how trajectory-local credit affects multi-turn refinement and to characterize the source of performance degradation across editing turns.

Table 4: Vision2Web Avg by turn for the controlled ablation-study.

Method	T0	T1	T2	T3	T4	T5
Terminal RL	33.5	35.4	36.9	37.4	36.6	35.2
StepWeb	33.8	35.8	37.1	38.0	39.2	37.7

Table 5: Trajectory-local event statistics on Vision2Web Level 1.

Event	Frequency	ΔScore	Adv. Shift
Regression	19.8%	-1.8	-0.31
Repair	25.6%	$+1.0$	$+0.05$
New Gain	43.2%	$+2.0$	$+0.12$
Tie	11.4%	0.0	-0.02

Table 6: Chained versus independent refinement on Vision2Web Level 1.

Setting	Score Dynamics	Success Rate
Independent	variation < 0.7	$67.4\% \rightarrow 66.8\%$
Chained	gap $0.3 \rightarrow 4.8$	$68.1\% \rightarrow 53.7\%$

6.1 Why Does Step-Level Credit Help?

Table 4 gives the turn-wise values underlying the T0, Best, and Final summaries in Table 3, where Best is the maximum mean score across turns. The two methods begin from comparable initial states but diverge as refinement proceeds. Terminal RL reaches a strong intermediate state and subsequently loses much of this progress, whereas StepWeb reaches a higher peak and retains more quality at the final turn. The benefit therefore lies primarily in controlling the inherited editing state rather than improving initialization. Trajectory-local credit reduces the ambiguity of terminal-only supervision by distinguishing productive progress from regression.

6.2 What Does the Local Signal Learn?

Table 5 summarizes the trajectory-local events. Regression receives strong negative credit, Repair receives a modest positive adjustment, and New Gain receives the largest positive credit. Although Repair and New Gain both increase the immediate score, they play different roles in the trajectory: the former recovers previously attained quality, whereas the latter establishes a new historical best. The local signal captures not only the direction of an edit, but also its role relative to trajectory history.

6.3 What Makes Later Turns Difficult?

Table 6 separates intrinsic turn difficulty from error propagation. When each edit starts from the corresponding clean state, both score variation and success rate remain nearly stable. Under chained refinement, however, the gap to the clean-state control grows from 0.3 to 4.8 points and the successful-edit rate falls from 68.1% to 53.7%. Later turns are therefore not substantially harder in isolation; their difficulty arises primarily from errors inherited from earlier edits. Multi-turn refinement thus induces a compounding state shift across the trajectory.

7 Conclusion

We presented StepWeb, a trajectory-aware reinforcement learning framework for multi-turn screenshot-to-code refinement. It combines terminal trajectory comparison with history-aware Regression, Repair, and New Gain signals, redistributing local credit through token-length-weighted centering while preserving the trajectory-level advantage. Across responsive reconstruction and interactive webpage generation, StepWeb improves over its Qwen3.5-9B backbone and reduces harmful regressions relative to terminal-only optimization. These results support trajectory-local credit assignment for reliable refinement under inherited, state-dependent errors.

REPRODUCIBILITY STATEMENT

The appendix documents the task wrapper, observation and action protocol, reward construction, optimization objective, failure handling, training configuration, and artifact layout. Candidate programs are evaluated in a deterministic browser sandbox, and failed executions remain in the evaluation set. We will release the implementation, fixed environment revisions, split manifests, and per-example outputs with the paper, subject to the licenses of the underlying webpages and assets.

ETHICS AND DATA GOVERNANCE

Website screenshots, source code, fonts, and images may carry separate rights. Training samples therefore require page- and asset-level provenance rather than relying only on a dataset-level license. Official benchmark pages are evaluation-only. Candidate webpages execute in an isolated, network-disabled sandbox, and anti-copy checks prevent direct access to target screenshots. A visually faithful page can still be inaccessible, insecure, or functionally incorrect; generated code should be reviewed before deployment.

AI USE STATEMENT

Generative AI tools assisted with literature organization, language editing, and initial LaTeX drafting. The authors remain responsible for the method, citations, implementation, experiments, and final claims.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Anthropic. Claude 4 system card. Technical report, Anthropic, 2025a. URL https://www.anthropic.com/system-cards.

Anthropic. Claude code documentation, 2025b. URL https://docs.anthropic.com/en/docs/claude-code. Accessed 2026-09-20.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report, 2025. URL https://arxiv.org/abs/2502.13923.

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *arXiv preprint arXiv:1705.07962*, 2017.

Bei Chen, Fengji Zhang, Anh Nguyen, Daoguang Zan, Zeqi Lin, Jian-Guang Lou, and Weizhu Chen. Codet: Code generation with generated tests, 2022. URL https://arxiv.org/abs/2207.10397.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023. URL https://arxiv.org/abs/2304.05128.

Gheorghe Comanici et al. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities, 2025. URL https://arxiv.org/abs/2507.06261.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, et al. Process reinforcement through implicit rewards. *arXiv preprint arXiv:2502.01456*, 2025.

Jie Deng, Kaichun Yao, and Libo Zhang. Visrefiner: Learning from visual differences for screenshot-to-code generation, 2026. URL https://arxiv.org/abs/2602.05998.

Haoyu Dong. Self-distillation policy optimization via visual feedback: Bridging code and visual artifacts, 2026. URL https://arxiv.org/abs/2606.10334.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. StepCoder: Improving code generation with reinforcement learning from compiler feedback. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 4571–4585, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.251. URL https://aclanthology.org/2024.acl-long.251/.

Peitong Duan, Chin-Yi Cheng, Bjoern Hartmann, and Yang Li. Visual prompting with iterative refinement for design critique generation, 2025. URL https://arxiv.org/abs/2412.16829.

Jonas Gehring, Kunhao Zheng, Jade Copet, Vegard Mella, Taco Cohen, and Gabriel Synnaeve. Rlef: grounding code llms in execution feedback with reinforcement learning. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing, 2024. URL https://arxiv.org/abs/2305.11738.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Bohua Chen, Yi Su, 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 on Web Conference 2025*, WWW '25, pp. 1834–1845. ACM, April 2025a. doi: 10.1145/3696410.3714889. URL http://dx.doi.org/10.1145/3696410.3714889.

Yi Gui, Zhen Li, Zhongyi Zhang, Yao Wan, Dongping Chen, Hongyu Zhang, Yi Su, Bohua Chen, Xing Zhou, Wenbin Jiang, and Xiangliang Zhang. Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs, 2025b. URL https://arxiv.org/abs/2505.09904.

Junrong Guo, Shancheng Fang, Yadong Qu, and Hongtao Xie. Seeing is improving: Visual feedback for iterative text layout refinement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 25893–25903, June 2026.

Zehai He, Wenyi Hong, Zhen Yang, Ziyang Pan, Mingdao Liu, Xiaotao Gu, and Jie Tang. Vision2web: A hierarchical benchmark for visual website development with agent verification, 2026. URL https://arxiv.org/abs/2603.26648.

Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya Binta Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. Ledex: training llms to better self-debug and explain code. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Yilei Jiang, Yaozhi Zheng, Yuxuan Wan, Jiaming Han, Qunzhong Wang, Michael R. Lyu, and Xiangyu Yue. Screencoder: Advancing visual-to-code generation for front-end automation via modular multimodal agents. Accepted at the EMNLP 2026 Main Conference, 2026. Accepted at the EMNLP 2026 Main Conference; arXiv:2507.22827.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R Narasimhan. SWE-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VTF8yNQM66.

Amirhossein Kazemnejad, Milad Aghajohari, Eva Portelance, Alessandro Sordoni, Siva Reddy, Aaron Courville, and Nicolas Le Roux. Vineppo: Refining credit assignment in rl training of llms, 2024. URL https://arxiv.org/abs/2410.01679.

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

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven C.H. Hoi. Coderl: mastering code generation through pretrained models and deep reinforcement learning. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, QIANG FU, Xiao Han, Yang Wei, and Deheng Ye. RLTF: Reinforcement learning from unit test feedback. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=hjYmsV6nXZ.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Junting Pan, Mingjie Zhan, and Hongsheng Li. Webgen-agent: Enhancing interactive website generation with multi-level feedback and step-level reinforcement learning, 2025a. URL https://arxiv.org/abs/2509.22644.

Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Webgen-bench: Evaluating llms on generating interactive and functional websites from scratch, 2025b. URL https://arxiv.org/abs/2505.03733.

Zimu Lu, Houxing Ren, Yunqiao Yang, Ke Wang, Zhuofan Zong, Mingjie Zhan, and Hongsheng Li. Fullstack-agent: Enhancing agentic full-stack web coding via development-oriented testing and repository back-translation, 2026. URL https://arxiv.org/abs/2602.03798.

Xufang Luo, Yuge Zhang, Zhiyuan He, Zilong Wang, Siyun Zhao, Dongsheng Li, Luna K. Qiu, and Yuqing Yang. Agent lightning: Train any ai agents with reinforcement learning, 2025. URL https://arxiv.org/abs/2508.03680.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

OpenAI. Gpt-4o system card, 2024. URL https://arxiv.org/abs/2410.21276.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Xinyue Yang, Jiadai Sun, Yu Yang, Shuntian Yao, Tianjie Zhang, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. *arXiv preprint arXiv:2411.02337*, 2024.

Qwen Team. Qwen3.5-9b model card, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-9B.

Qwen Team. Qwen3.6-35b-a3b: Agentic coding power, now open to all, 2026b. URL https://qwen.ai/blog?id=qwen3.6-35b-a3b.

Qwen Team. Qwen3.8-27b model release, 2026c. URL https://github.com/AlibabaCloud-Official/Qwen3.8-27B.

Hannah Sansford, Derek H. C. Law, Wei Liu, Abhishek Tripathi, Niresh Agarwal, and Gerrit J. J. van den Burg. Vision-guided iterative refinement for frontend code generation, 2026. URL https://arxiv.org/abs/2604.05839.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Mingchuan Zhang, Y. K. Li, Yang Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Noah Shinn, Federico Cassano, Edward Berman, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning, 2023.

Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating 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*, pp. 3956–3974, 2025. URL https://aclanthology.org/2025.naacl-long.201/.

Aaditya Singh et al. Openai gpt-5 system card, 2025. URL https://arxiv.org/abs/2601.03267.

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, June 2025. ISSN 2994-970X. doi: 10.1145/3729364. URL http://dx.doi.org/10.1145/3729364.

Zihan Wang, Kangrui Wang, Qineng Wang, Pingyue Zhang, Linjie Li, Zhengyuan Yang, Xing Jin, Kefan Yu, Minh Nhat Nguyen, Licheng Liu, Eli Gottlieb, Yiping Lu, Kyunghyun Cho, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning, 2025. URL https://arxiv.org/abs/2504.20073.

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, 2024. URL https://arxiv.org/abs/2406.07739.

Jingyu Xiao, Yuxuan Wan, Yintong Huo, Zixin Wang, Xinyi Xu, Wenxuan Wang, Zhiyao Xu, Yuhang Wang, and Michael R. Lyu. Interaction2code: Benchmarking mllm-based interactive webpage code generation from interactive prototyping. In *2025 40th IEEE/ACM International Conference on Automated Software Engineering (ASE 2025)*, pp. 241–253, 2025. doi: 10.1109/ASE63991.2025.00028.

Qiao Xu, Yipeng Yu, Chengxiao Feng, and Xu Liu. 1d-bench: A benchmark for iterative ui code generation with visual feedback in real-world, 2026. URL https://arxiv.org/abs/2602.18548.

John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: agent-computer interfaces enable automated software engineering. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2024. Curran Associates Inc. ISBN 9798331314385.

Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiale Cheng, Xiaotao Gu, and Jie Tang. Ui2code^N: Ui-to-code generation as interactive visual optimization. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026.

Chuhuai Yue, Jiajun Chai, Yufei Zhang, Zixiang Ding, Xihao Liang, Peixin Wang, Shihai Chen, Wang Yixuan, Wangyanping, Guojun Yin, and Wei Lin. UIOrchestra: Generating high-fidelity code from UI designs with a multi-agent system. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 2769–2782, Suzhou, China, November 2025. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.150. URL https://aclanthology.org/2025.findings-emnlp.150/.

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

Li Zhong, Zilong Wang, and Jingbo Shang. Debug like a human: A large language model debugger via verifying runtime execution step by step. In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 851–870, 2024.

A Implementation Details

A.1 Rollout Indexing and Information Flow

We index the independently generated initial implementation by $t = 0$ and the five refinement actions by $t = 1, \ldots, T$, with $T = 5$. Thus, each trajectory contains six rendered states and five scored transitions:

$$C_{i,0} \to C_{i,1} \to \cdots \to C_{i,5}.$$

At refinement turn t, the policy receives only information available after executing turn $t-1$. Terminal rewards, future renders, other rollout branches, path-event labels, and group-normalized advantages are computed by the trainer and are never included in the policy observation. This indexing matches the T0–T5 convention used in Tables 3 and 4.

A.2 Observation and Action Templates

The initial-generation prompt contains all task prototypes, the asset manifest, target viewport sizes, and the requirement to return a self-contained executable implementation. The repair prompt contains the same fixed task information together with the branch's current code, previous render(s), previous scalar score, and VLM-generated correction feedback. The policy never observes another branch, a future score, the historical best, a path label, a group statistic, or an advantage.

The main action is the complete updated codebase serialized under a fixed file schema. The structured-patch ablation instead emits a JSON action and unified diff. In both cases, the environment materializes a complete workspace before rendering. Parser errors, prohibited paths, failed patches, missing entry files, blank pages, and execution failures are retained as failed turns rather than silently discarded.

The initial-generation instruction follows the template

> Given the reference screenshots, local asset manifest, and target viewport sizes, produce a self-contained executable webpage implementation. Use only the provided local assets and return the complete codebase in the required file schema.

The refinement instruction follows the template

> Given the same reference screenshots and assets, inspect the current implementation, its latest renders and score, and the supplied visual feedback. Return a complete revised implementation that corrects the visible discrepancies while preserving already correct content and behavior.

These templates describe the information boundary; task-specific file names, viewport dimensions, and serialized artifacts are inserted by the wrapper.

A.3 Training and Evaluation Signals

Training uses the scalar state score $s(C) \in [0, 1]$ defined in Section 3. The score determines terminal trajectory rewards and the direction and magnitude of local path events. All methods in a controlled comparison use the same rendering and scoring pipeline, so TC-SLC does not require extra evaluator calls beyond those already needed to execute the trajectory. Evaluation remains separate from this training signal: each benchmark is reported using its native metrics as described in Section 5.1.

For training, the renderer computes a normalized visual-fidelity score for each required viewport and averages across the available viewports:

$$s(C) = \frac{1}{|V|} \sum_{v \in V} s^v(C), \qquad s^v(C) \in [0, 1]. \tag{21}$$

The same evaluator implementation and viewport set are used for every method within a controlled comparison. Invalid code, a blank render, a browser timeout, or a missing required entry file receives $s(C) = 0$ and remains in the rollout group. Scores are clipped to $[0, 1]$ before constructing rewards.

Correction feedback is generated from the current branch state and is part of the next-turn observation. It is held fixed as an experimental component in the Terminal RL comparison; the compared methods differ in how the resulting trajectories are credited, rather than in their access to feedback. We use a frozen Qwen3.8-27B model as the feedback generator. It is fixed throughout SFT, RL, ablation, and evaluation runs and is not updated jointly with the policy. We use temperature 0.2, top-p 0.9, and a maximum response length of 1,024 tokens. The feedback generator receives the reference screenshots, current renders, current implementation, and viewport metadata, and returns a concise list of visible discrepancies and suggested corrections. It does not receive the reference source code, scalar reward, future states, path labels, group statistics, or policy advantages. The scalar reward is computed separately by the visual evaluator. If feedback generation fails, the turn proceeds with an empty feedback string rather than dropping the trajectory.

Score changes with magnitude below $\epsilon_{\text{tie}} = 10^{-3}$ are treated as ties. The group-normalization constant is $\epsilon = 10^{-6}$. We set the local-credit scale to the standard deviation of non-tied score changes in the current rollout batch, lower-bounded by 10^{-3}:

$$\delta_{\text{scale}} = \max\big(\text{Std}\{s_{i,t} - s_{i,t-1} : |s_{i,t} - s_{i,t-1}| \geq \epsilon_{\text{tie}}\}, 10^{-3}\big) . \tag{22}$$

Illustrative path classification. For the sequence $0.50 \to 0.56 \to 0.52 \to 0.55 \to 0.60$, the transitions are New Gain, Regression, Repair, and New Gain. The $0.52 \to 0.55$ increase is a Repair because it remains below the previous best 0.56. This is an illustration, not an empirical result.

A.4 Training Configuration

Table 7: Default training configuration used by StepWeb.

Component	Setting
Backbone	Qwen3.5-9B
Thinking	off
Sampling	temperature 0.6, top-p 0.95, top-k 20
Maximum generation	6,000 tokens per turn
Initial SFT	1 epoch, LR 1×10^{-5}, bf16
Refinement SFT	1–2 epochs, LR 5×10^{-6} to 1×10^{-5}
RL optimizer	AdamW, LR 1×10^{-6}, gradient clipping 1.0
RL group size	$K = 8$
Refinement horizon	$T = 5$
PPO clip η	0.2
KL coefficient	0.01
Local strength β	0.25
δ_{scale}	batch std. of non-tied score changes; floor 10^{-3}
ϵ_{tie}	10^{-3}
ϵ in group normalization	10^{-6}
λ_{reg}	1.0
λ_{repair}	0.5
λ_{new}	1.0
Failure handling	score 0; retain trajectory
Independent training seeds	3
Feedback generator	Qwen3.8-27B (frozen)
Feedback sampling	temperature 0.2, top-p 0.9
Maximum feedback length	1,024 tokens
Feedback frequency	once per refinement turn
Feedback failure	empty feedback; retain trajectory

A.5 Artifact Layout

Each executed turn stores an immutable record:

```
rollouts/<task>/<rollout>/<turn>/
```

```
input.json
response.raw
action.json or patch.diff
current_code/
render/{desktop,tablet,mobile}.png
verify.json
score.json
token_logprobs.npz
```

The task-level record additionally stores model, tokenizer, processor, browser, evaluator, prompt, and container revisions. The rollout viewer follows the sequence target → previous render → model output → applied code → verification → score. This logging scheme allows trajectory rewards, local path events, and token-level advantages to be reconstructed after execution.

B Data Construction and Governance

Initial SFT draw from WebSight, audited WebCode2M, and other explicitly licensed sources. Same-page refinement supervision is admitted only when the successor implementation is verified to improve or correctly repair the current state. Controlled corruption modify spacing, sizing, layout, typography, color, assets, DOM structure, or responsive breakpoints. Model-generated successors are not automatically treated as gold targets.

All versions of a webpage remain in the same data split. Near-duplicate filtering combines code hashes, perceptual screenshot hashes, OCR text, DOM structure, domain or template metadata, and asset fingerprints. Each record retains source revision, page and asset provenance, renderer version, parent-state hash, and verification evidence.

Vision2Web Level 1, Design2Code, Design2Code-HARD, and Interaction2Code are used only for evaluation and are excluded from training and hyperparameter selection. Dataset-level metadata is not assumed to grant rights over every third-party image, font, script, or webpage asset.

C Evaluation Protocol and Reported Statistics

Native benchmark aggregation. For Vision2Web, the reported Avg score is

$$\mathrm{Avg}_{\mathrm{V2W}} = \frac{\mathrm{Desktop} + \mathrm{Tablet} + \mathrm{Mobile}}{3}.$$

For each Design2Code split, the aggregate used in the backbone-relative summary averages Block, Text, Position, Color, and CLIP. For Interaction2Code, it averages the eight full-page and interaction-region metrics in Table 2. These aggregates summarize relative improvement; the native metrics remain the primary results.

For benchmark b, let M_b be its selected native metrics. We compute

$$g_b = \frac{\frac{1}{|M_b|}\sum_{m\in M_b} s^{\textsc{StepWeb}}_{b,m}}{\frac{1}{|M_b|}\sum_{m\in M_b} s^{\mathrm{base}}_{b,m}} - 1, \qquad g_{\mathrm{macro}} = \frac{1}{4}\sum_b g_b.$$

The values in Tables 1 and 2 give $g_{\mathrm{macro}} = 16.4\%$.

Trajectory statistics. For N evaluated trajectories, let $\bar{s}_t = N^{-1}\sum_{i=1}^{N} s_{i,t}$. We report

$$\mathrm{T0} = \bar{s}_0, \qquad \mathrm{Final} = \bar{s}_T, \qquad \mathrm{Best} = \max_{0\le t\le T} \bar{s}_t.$$

Thus, Best is the peak of the mean turn-wise curve, not the mean of per-trajectory maxima. Best−Final is the corresponding post-peak loss. Regression is the fraction of transitions satisfying $s_{i,t} - s_{i,t-1} < -\epsilon_{\mathrm{tie}}$; transitions with $|s_{i,t} - s_{i,t-1}| \le \epsilon_{\mathrm{tie}}$ are ties. Failed executions remain in the evaluated set.

Checkpoint separation. Table 1 uses the checkpoint selected for the main cross-benchmark comparison. Tables 3–6 use separately trained checkpoints from the controlled ablation suite. They support within-suite comparisons and are not expected to reproduce the absolute main-table Vision2Web score.

D BACKBONE-RELATIVE RESULTS

Table 8 summarizes the aggregate comparison between STEPWEB and its Qwen3.5-9B backbone. For Vision2Web, we average Desktop, Tablet, and Mobile. For Design2Code and Design2Code-HARD, we average Block, Text, Position, Color, and CLIP. For Interaction2Code, we average the eight reported full-page and interaction-region metrics. The final column reports relative improvement within each benchmark.

Table 8: Aggregate improvement of STEPWEB over Qwen3.5-9B. The four benchmark-level relative gains average to 16.4%.

Benchmark	Qwen3.5-9B	STEPWEB	Rel. Gain
Vision2Web Level 1 (Avg.)	32.87	39.90	+21.4%
Design2Code (Avg.)	82.86	93.22	+12.5%
Design2Code-HARD (Avg.)	75.02	89.20	+18.9%
Interaction2Code (Avg.)	0.5196	0.5865	+12.9%
Macro average	–	–	**+16.4%**

Table 9: Native Design2Code metrics for Qwen3.5-9B and STEPWEB.

Method	Block	Text	Position	Color	CLIP
Qwen3.5-9B	74.7	89.0	78.8	83.6	88.2
STEPWEB	**93.8**	**96.3**	**88.5**	**92.2**	**95.3**

Table 10: Native Design2Code-HARD metrics for Qwen3.5-9B and STEPWEB.

Method	Block	Text	Position	Color	CLIP
Qwen3.5-9B	58.9	83.6	67.6	78.7	86.3
STEPWEB	**85.1**	**93.1**	**88.4**	**87.5**	**91.9**

E ADDITIONAL ABLATIONS

The main ablation in Table 3 isolates trajectory centering, history-aware separation of Repair and New Gain, magnitude-sensitive local credit, and the action representation. Table 3 reports the completed controls for trajectory centering, history-aware event separation, magnitude-sensitive credit, and action representation. Further sensitivity analyses—including rollout group size, refinement horizon, local-credit weights, adaptive stopping, and evaluator repeatability—are useful future extensions but are not included in the present empirical claims.

F ADDITIONAL ANALYSIS AND DISCUSSION

Information Boundary. Regression, Repair, and New Gain are labels assigned to executed score paths rather than generated rationales or hidden reasoning. Future scores, historical bests, path labels, group statistics, and advantages remain trainer-side quantities. At inference time, the policy receives only the task specification, assets, its own inherited implementation and render, and the registered feedback. Thus, STEPWEB introduces neither privileged test-time information nor an additional inference module.

Evaluation Controls. Initial reconstruction SFT, verified refinement SFT, and online multi-turn RL are evaluated separately. Official benchmark examples are excluded from training, and every benchmark retains its native evaluation metrics. In the controlled Terminal RL comparison, rollout count, rendering, VLM feedback, model sampling, and refinement horizon are held fixed. Local

credit is computed only from scores and token counts already produced by the same trajectories, so it introduces no additional interaction budget.

Trajectory centering is applied before PPO clipping. Here $L_{i,t} = \sum_k m_{i,t,k}$ is the number of generated tokens included in the policy loss. The local adjustments satisfy $\sum_{t=1}^{T} L_{i,t}\widetilde{q}_{i,t} = 0$, and therefore

$$\frac{\sum_{t=0}^{T} L_{i,t}A_{i,t}}{\sum_{t=0}^{T} L_{i,t}} = A_i^{\mathrm{traj}}.$$

This conservation holds before PPO clipping; it does not imply an exactly conserved clipped objective or parameter update.

Robustness and Efficiency. Trajectory-local credit introduces only lightweight bookkeeping over already executed refinement states and requires no additional model calls at inference time. Its computation uses per-turn scalar scores, historical trajectory statistics, and generated-token counts. The principal additional training cost therefore comes from the multi-turn rollouts themselves rather than from credit redistribution.

Scope and Limitations. Path events characterize executed transitions but do not provide token-level causal attribution. CSS inheritance, browser reflow, shared containers, and cross-element dependencies can make the effect of a code edit highly nonlocal. Moreover, conservation applies to token-weighted advantages before PPO clipping rather than to the exact parameter update. The fixed five-turn horizon is an experimental control rather than an optimal stopping policy. Finally, while our evaluation includes both static reconstruction and interactive webpage generation, extending trajectory-local events to full-stack tasks may require joint visual and functional verification so that improvement on one axis cannot conceal failure on another.

G Qualitative Comparison with the Backbone

We provide three selected qualitative examples to illustrate the improvements of StepWeb over its Qwen3.5-9B backbone. These examples are selected from the Design2Code and Design2Code-HARD benchmarks. In each figure, the left panel shows the ground-truth webpage, the middle panel shows the output generated by the backbone model, and the right panel shows the output generated by StepWeb.

As shown in Figures 3–5, StepWeb more faithfully reconstructs the overall page structure, spatial layout, content hierarchy, and visual styling. The improvement is particularly evident in long-page completion, multi-column layout organization, component sizing, and relative element positioning. These examples demonstrate that trajectory-based reinforcement learning improves not only local visual fidelity but also the model's ability to generate structurally complete and globally coherent webpages.

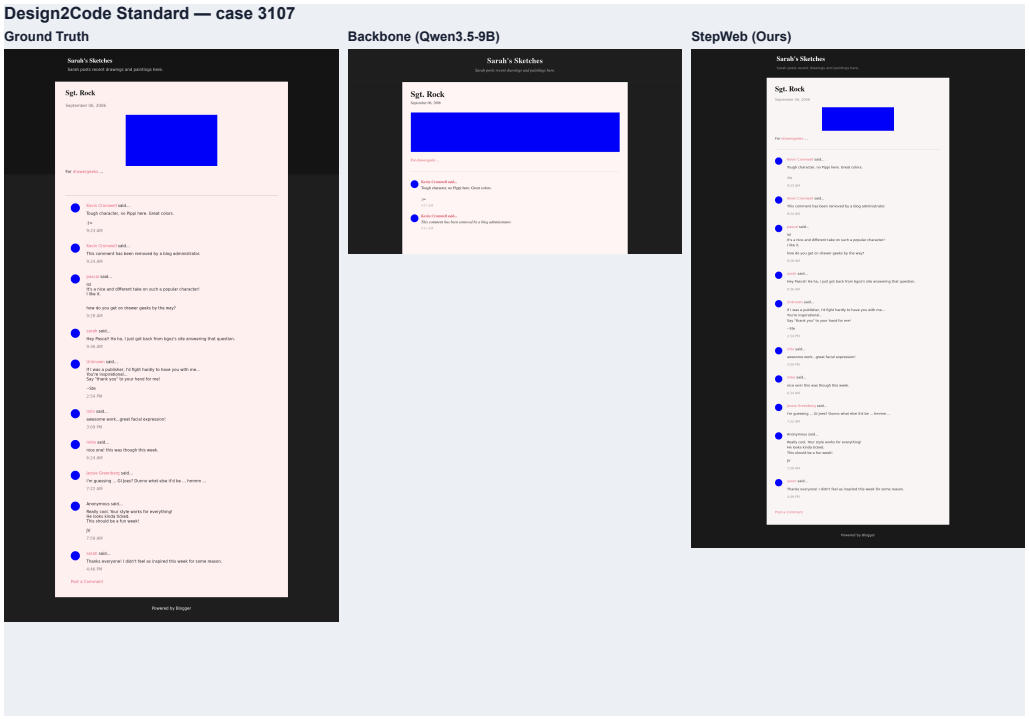


Figure 3: Qualitative comparison on Design2Code Standard, case 3107. The Qwen3.5-9B backbone generates only the upper portion of the webpage and omits most of the long-form content. In contrast, StepWeb reconstructs the complete page structure and more closely matches the ground truth in layout, content coverage, and visual appearance.

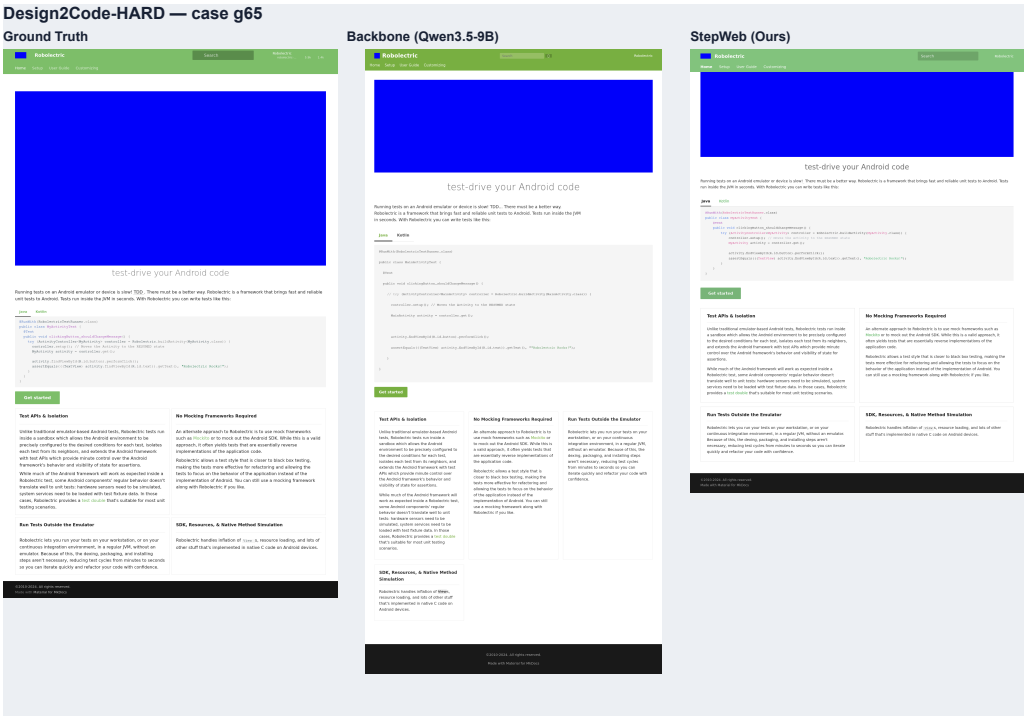


Figure 4: Qualitative comparison on Design2Code-HARD, case g65. Compared with the Qwen3.5-9B backbone, StepWeb produces a more accurate page width, content hierarchy, code-block layout, and multi-column card arrangement. It also better preserves the overall proportions and vertical structure of the reference page.

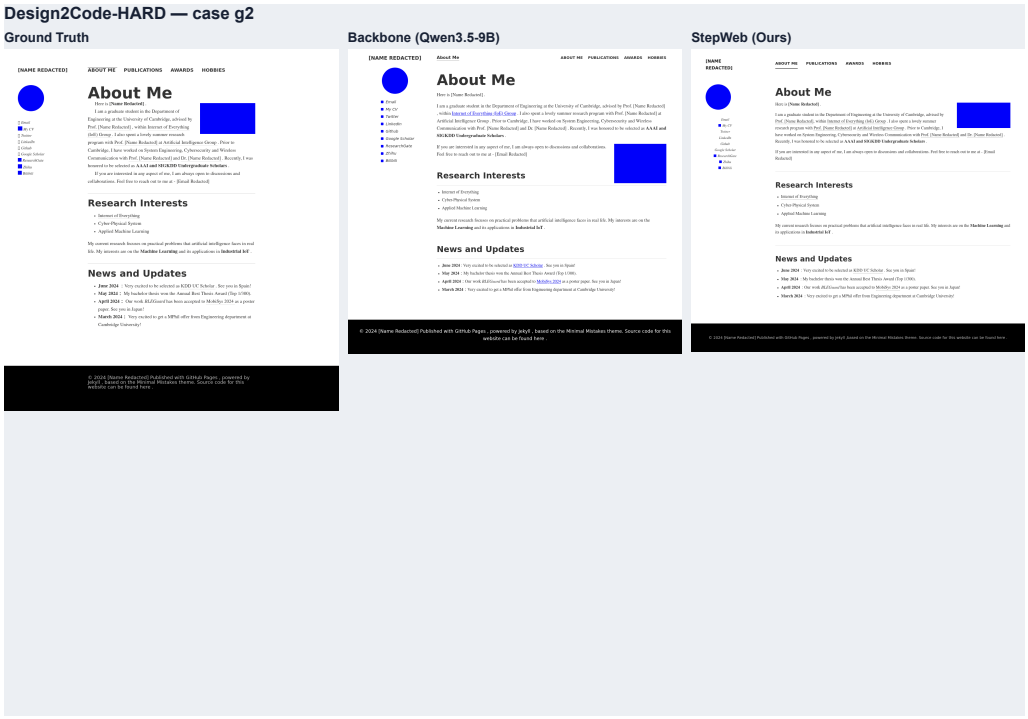


Figure 5: Qualitative comparison on Design2Code-HARD, case g2. StepWeb improves the reconstruction of the sidebar, navigation bar, main-content width, image placement, section spacing, and footer. The resulting webpage is structurally more consistent with the ground truth than the output of the Qwen3.5-9B backbone.