UI2Code^N: UI-to-Code Generation as Interactive Visual Optimization

Zhen Yang*, Wenyi Hong*, Mingde Xu, Xinyue Fan, Weihan Wang,

Jiale Cheng, Xiaotao Gu, Jie Tang

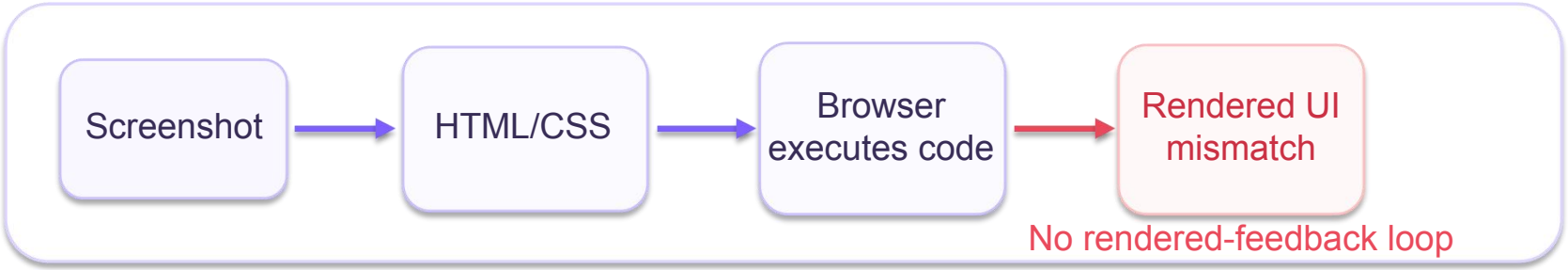
ICML 2026

*Equal contribution, corresponding author: Jie Tang

Why one-shot UI-to-Code is brittle

A screenshot-to-code model only knows whether it succeeded after the browser executes the code.

One-shot formulation hides the real failure point



Three coupled sources of brittleness

1 Rendering gap

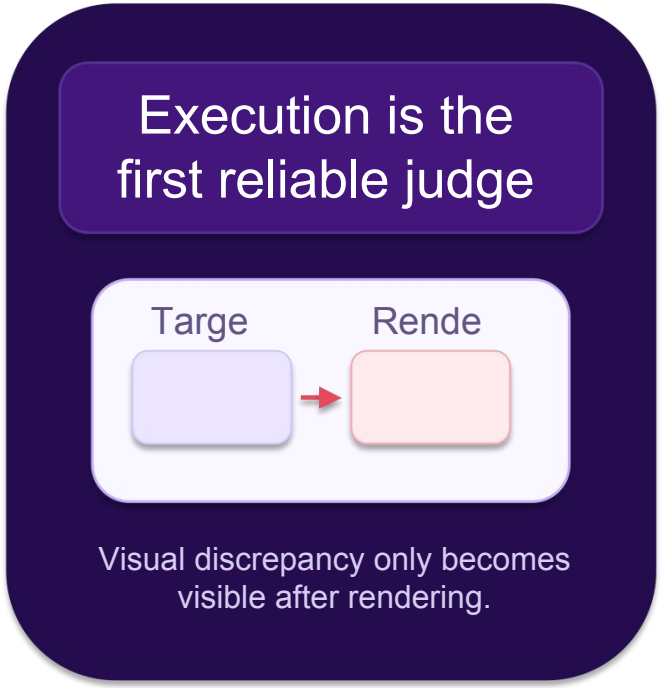
Browser defaults, font fallback, DPI scaling, and layout engines shift the final UI after execution.

2 Coupled visual errors

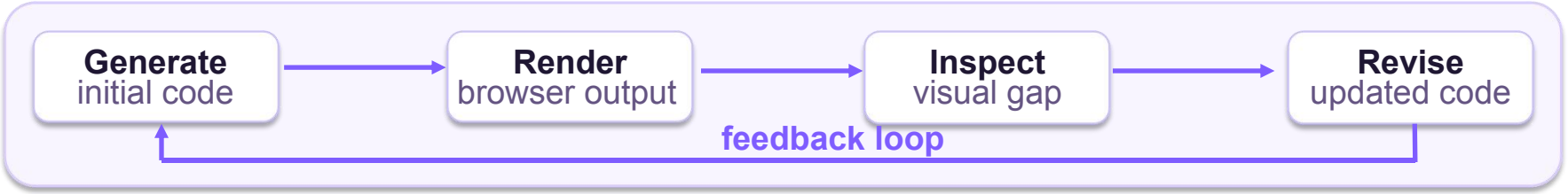
Spacing, alignment, typography, colors, and component sizes interact; fixing one can affect others.

3 No self-verification

One-shot prediction cannot see whether the rendered result actually matches the target.

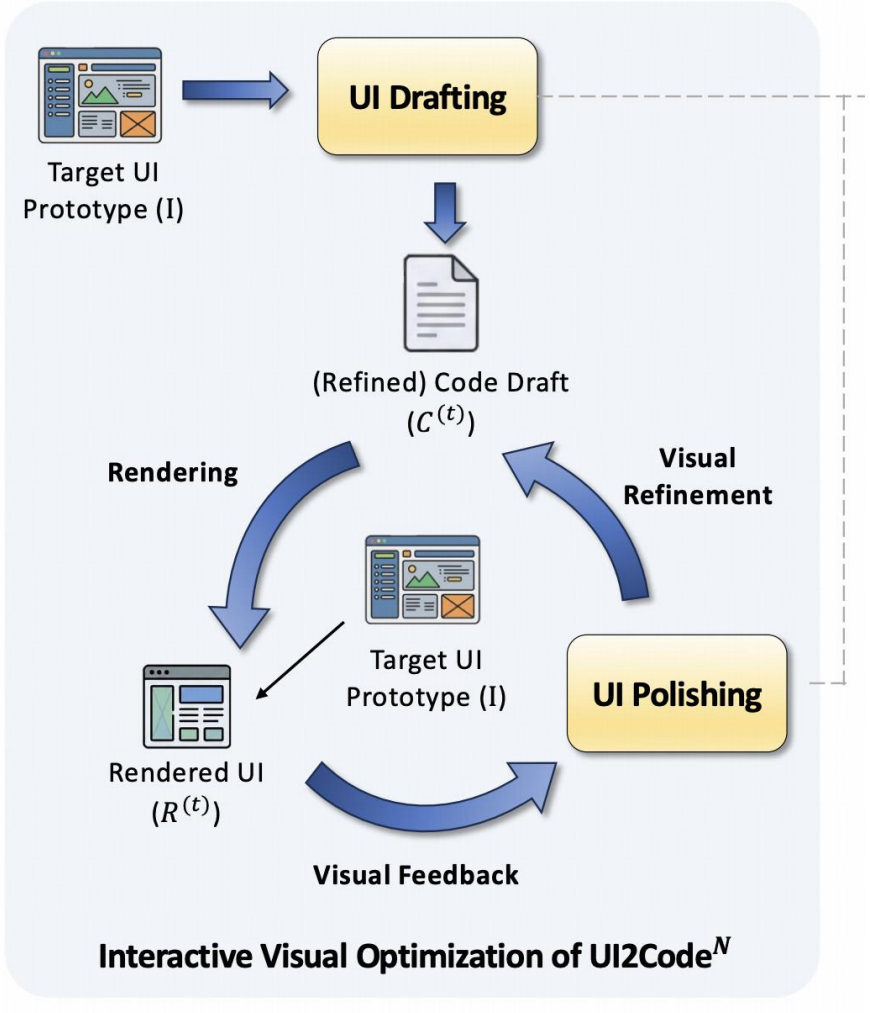


What real UI development actually needs



Interactive visual optimization

Reformulate UI-to-code as a closed loop over executable rendering feedback.



1 Generate — Create an initial code draft from the target UI.

2 Render — Execute the code and observe the rendered UI.

3 Refine — Use visual feedback to improve the next code version.

Objective: minimize implicit visual discrepancy D(I, Render(C))

One loop unifies three UI coding tasks

Drafting, polishing, and editing become different ways of using the same feedback-driven transformation.

UI Drafting

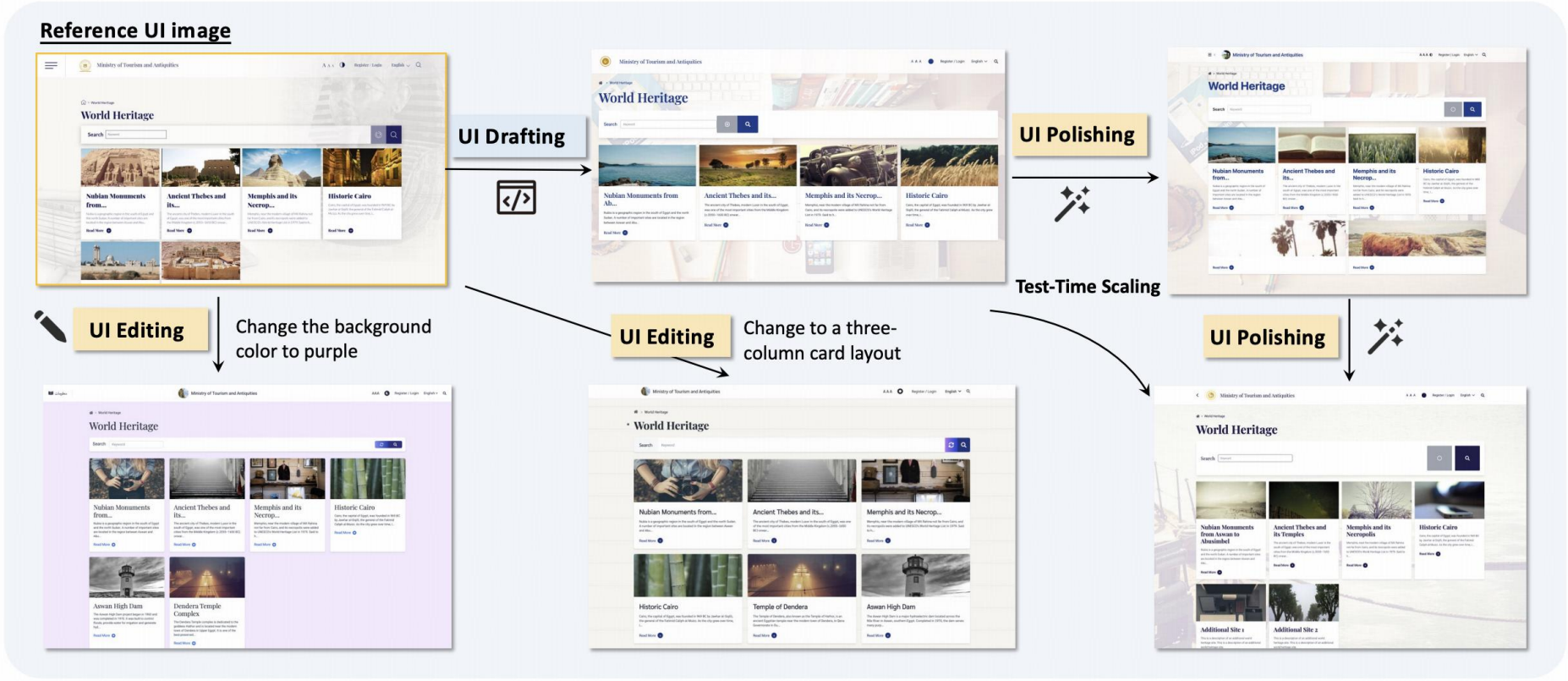
Start from target screenshot and produce first-pass code $C^{(0)}$.

UI Polishing

Iteratively refine code using Render($C^{(t)}$) and visual feedback.

UI Editing

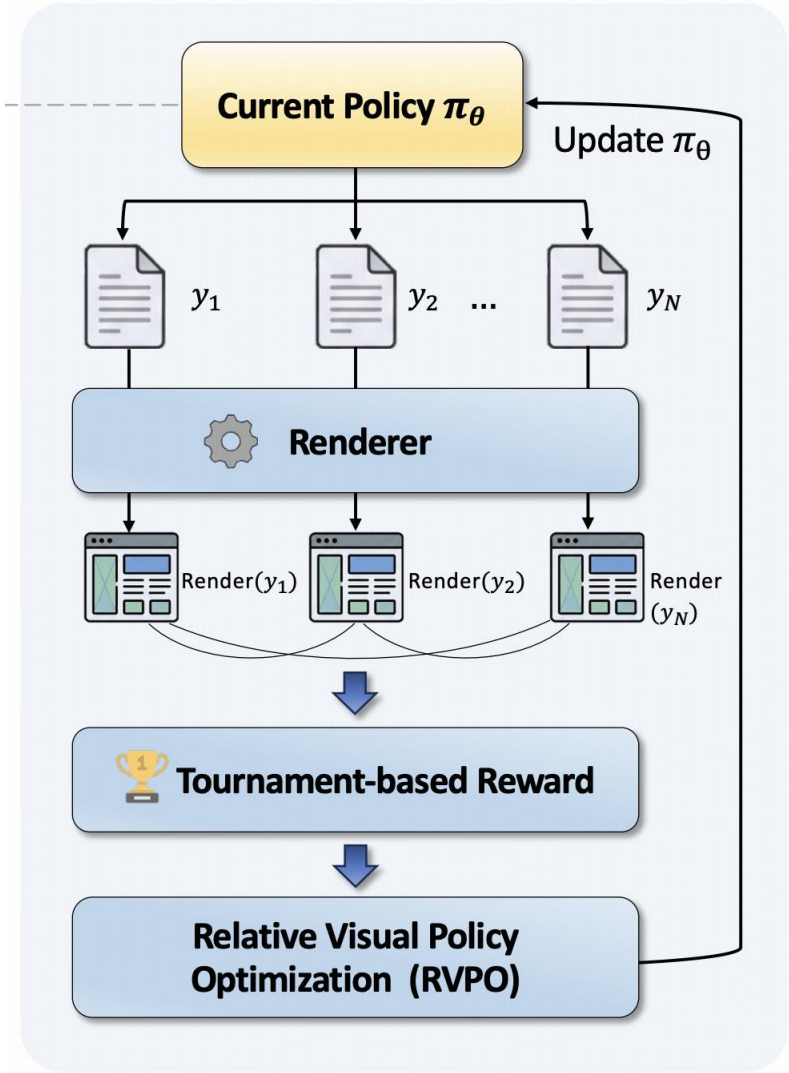
Apply localized instruction-conditioned changes while preserving structure.



UI2Code^N = test-time scaling through N refinement rounds

RVPO: learn from relative visual preference

Absolute VLM scores are noisy; relative comparisons are more stable for rendered UI quality.



1 Sample candidates

Generate multiple code rollouts from the current policy.

2 Render & compare

Evaluate candidates in the rendered visual space.

3 Tournament reward

Aggregate pairwise wins instead of using calibrated absolute scores.

Reward = relative ranking among rendered candidates

Training pipeline for UI2Code^N

A compact 9B VLM trained to perceive UI details, reason over code, and optimize with visual feedback.

1 Continual pre-training

Perceive UI details

- image–HTML pairs
- localized DOM grounding
- global UI-code likelihood

→

2 Supervised fine-tuning

Learn the edit loop

- drafting / polishing / editing
- <think> diagnosis
- <answer> executable code

→

3 RVPO reinforcement learning

Optimize rendered fidelity

- sample multiple candidates
- render & compare
- relative preference reward

What each stage contributes

Perception
fine-grained UI elements
layout, colors, typography

Code reasoning
long structured code
DOM + CSS dependencies

Visual diagnosis
detect rendered mismatch
spacing, alignment, style

Optimization
prefer better renderings
improve over iterations

Training objective stack

L_dom + L_pair → L_SFT(<think>, <answer>) → GRPO with tournament rewards

from UI-code grounding to explicit visual policy improvement

Main result: strong drafting and polishing

UI2Code^N-9B-RL reaches state-of-the-art open-source performance and competes with much larger closed-source systems.

Model	UI Drafting				UI Polishing	
	Design2Code	Flame	Web2Code	UI2Code-Real	UIPolish-Real	UIPolish-Synthetic
Closed-source baselines						
GPT-5	**89.7**	91.3	**93.7**	67.8	**85.0**	68.0
Gemini-2.5-Pro	89.5	87.5	90.6	**68.7**	74.0	68.0
Claude-4.5-Sonnet-thinking	82.9	**92.5**	87.8	67.2	81.0	66.0
Open-source baselines						
Qwen3-VL-32B-Instruct	83.3	82.5	81.2	65.0	46.0	55.0
GLM-4.1V-9B-Thinking	64.7	72.5	71.3	53.0	42.0	46.0
Our models						
UI2Code^{N}-9B-SFT	79.3	85.0	80.8	67.0	76.0	89.0
UI2Code^{N}-9B-RL	**88.6**	**95.0**	**92.5**	**76.5**	**80.0**	**94.0**

★ Best (highest) in each column is highlighted.

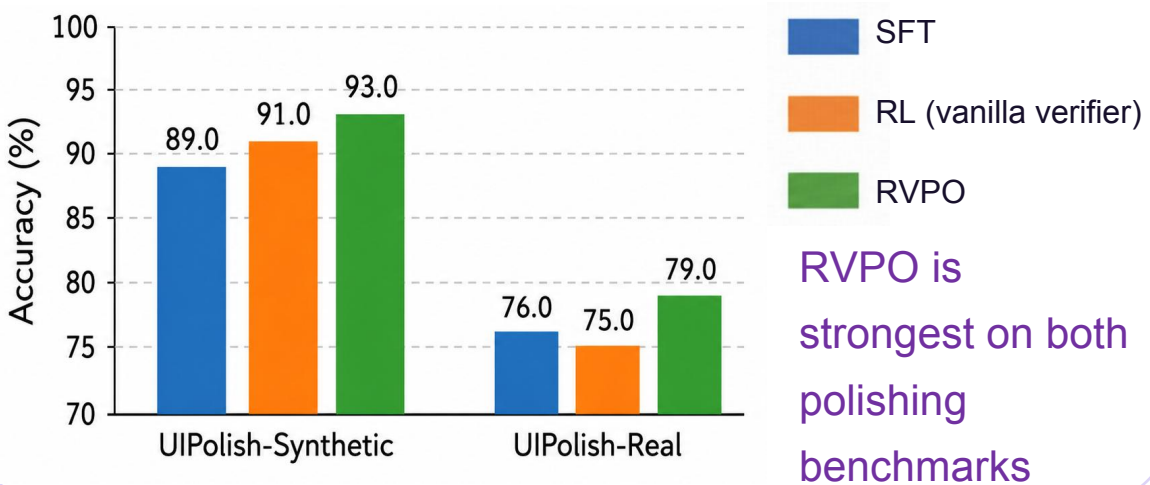
Structural fidelity breakdown

Model	Block	Text	Position	Color	CLIP
GPT-5	89.1	**94.2**	**86.4**	**78.0**	**81.6**
Gemini-2.5-Pro	89.1	93.5	85.5	71.4	80.9
Claude-4-Sonnet	88.7	93.2	84.6	72.0	80.5
Qwen2.5-VL-72B	86.6	91.6	76.8	67.8	77.8
UI2Code^{N}-SFT	86.8	91.5	81.7	69.7	79.0
UI2Code^{N}-RL	**88.7**	**93.1**	**83.8**	**72.6**	**80.5**

Ablations: reward design and iterative refinement

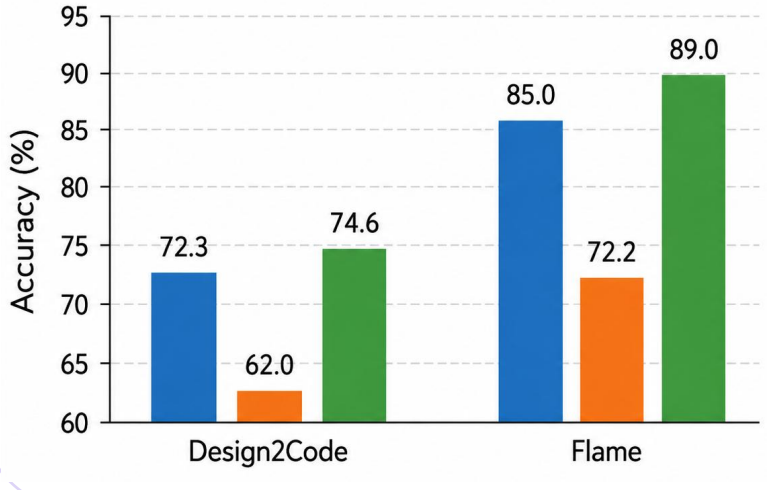
RVPO yields better reward signals and redered feedback helps multi-step refinement.

(a) Reward design matters



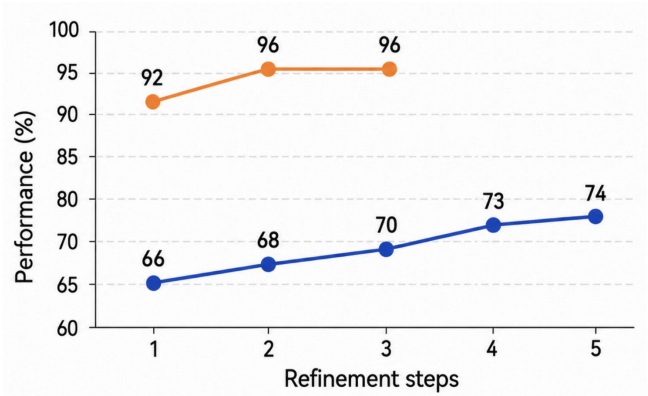
RVPO is strongest on both polishing benchmarks

(b) Absolute rewards are unstable



Direct absolute rewards can hurt; VLM-based relative learning is more reliable

(c) Test-time refinement scales

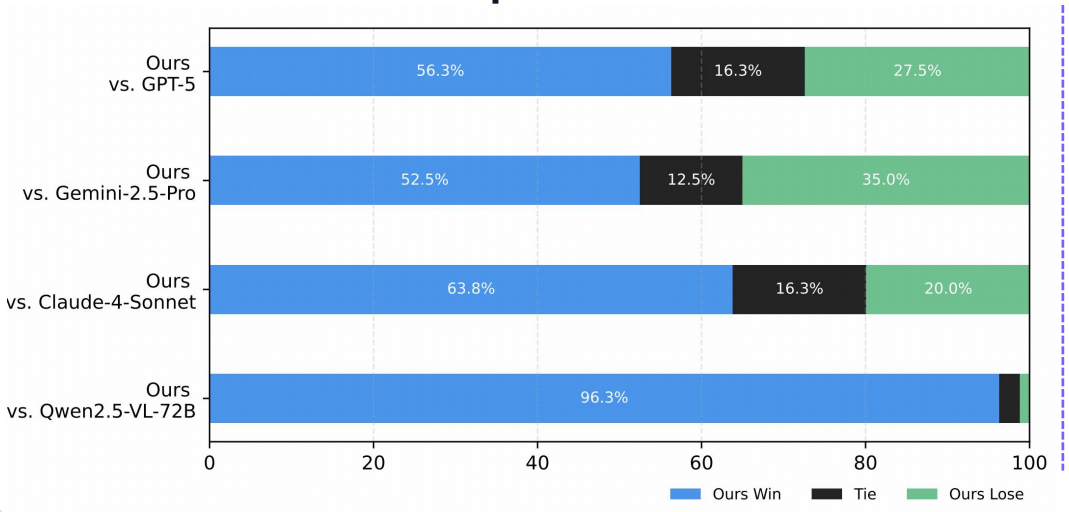


Rendered feedback supports iterative improvement beyond one-shot generation.

Human studies support the evaluation

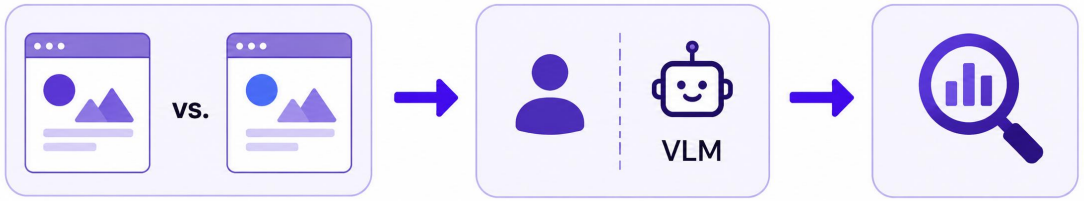
Human preference and judge validation indicate that visual evaluation aligns with human judgments.

Human evaluation setup



- Human raters assess editing quality and structural fidelity.
- Rankings from the automatic visual evaluator are compared with human preference.
- Comparator/verifier consistency is checked at the decision level.

Validation pipeline



Rendered UI pairs

Human prederence / VLM ranking

Agreement analysis

What the validation shows

Preference alignment

Automatic rankings match human judgments well.

Stable decision signals

Verifier decisions agree strongly with humans.

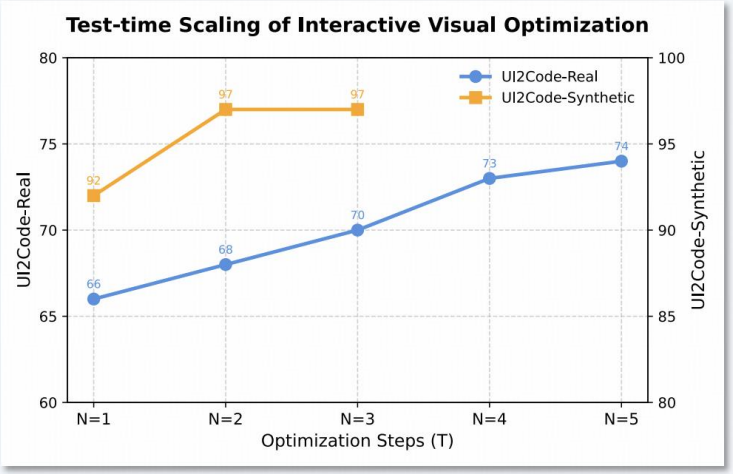
Reliable reward signal

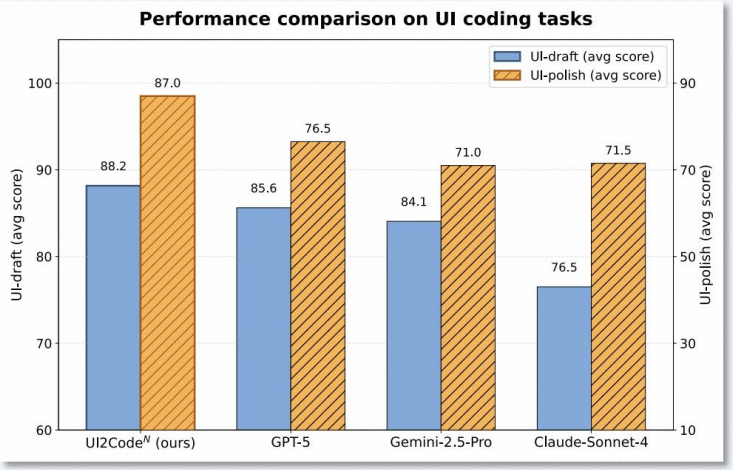
Relative visual feedback remains usable for optimization.

Conclusion

UI-to-code should be optimized in the rendered visual space.

1. From one-shot generation to closed-loop visual optimization
2. RVPO provides stable relative rewards for non-differentiable UI quality
3. A compact 9B model achieves SOTA across drafting, polishing, and editing





Thank you

Website: https://zheny2751-dotcom.github.io/ui2code-n.github.io/