Widget2Code

From Visual Widgets to UI Code via Multimodal LLMs

Houston H. Zhang[1], Tao Zhang[2], Baoze Lin[1], Yuanqi Xue[1], Yincheng Zhu[3], Huan Liu[1], Li Gu[4], Linfeng Ye[2], Ziqiang Wang[4], Xinxin Zuo[4], Yang Wang[4], Yuanhao Yu[1], Zhixiang Chi[2†]

[1] McMaster University [2] University of Toronto [3] University of Waterloo [4] Concordia University

† Project Lead / Corresponding Author

Project page https://djanghao.github.io/widget2code

arXiv: 2512.19918 · Video presentation

TL;DR First widget-level UI2Code benchmark

+ Perceptual Agent · WidgetDSL · Adaptive Rendering

CONTENTS

What we will cover

01 **Quick preview**
What this paper is about

02 **Task setting**
Why widgets are a new UI2Code problem

03 **Benchmark**
Dataset, fine-grained metrics, and baseline results

04 **Method**
Perceptual Agent · WidgetDSL · WidgetFactory

05 **Results**
Full-system comparison and ablation study

06 **Summary**
Takeaways and future work

What is this paper about?

Problem

Generate executable UI code from visual components

Challenge

- Lack of paired (component image, code) data
- Components are compact, context-free, and tightly constrained in layout space
- Requires generating geometry-consistent and controllable code

Solution

- A modular framework:
  - ☐ Perceptual agent (components, icons, color extraction)
  - ☐ WidgetFactory(Structured DSL generation + compilation)
  - ☐ Adaptive rendering (geometry-aware refinement)

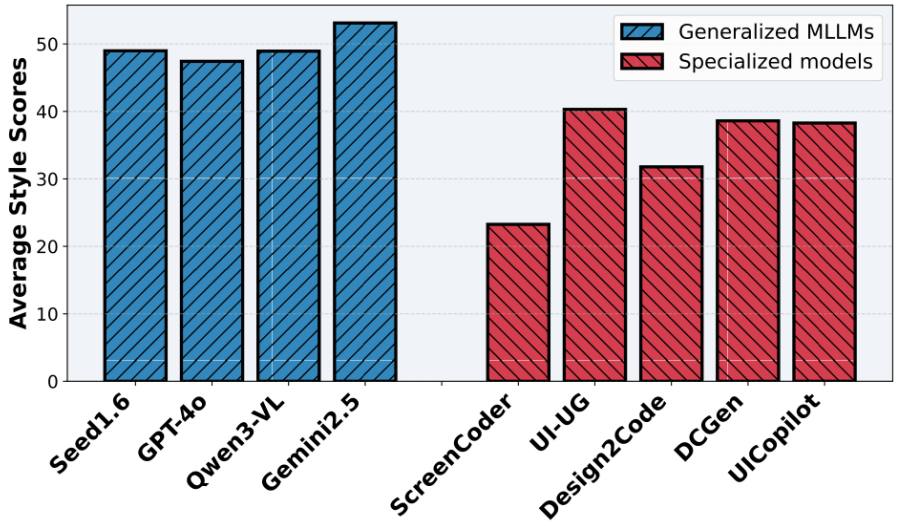


Figure 2. **Style score comparison on our widget benchmark.** Generalized MLLMs outperform specialized UI2Code models, which are tuned for other UI formats instead of widgets.

Results

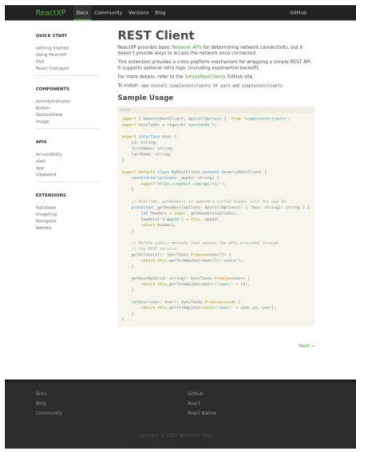
- General MLLMs significantly outperform specialized UI2Code models on widgets.
- The full system clearly improves layout, style, and geometry, reaching 100 on Geometry.
- However, issues like overflow, misalignment, and style mismatch remain, leaving room for future work.

3 / 15

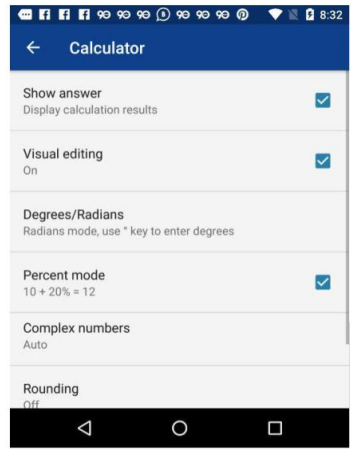
Why are widgets a new UI2Code problem?

- UI2Code aims to generate executable code that faithfully reconstructs the input UI.
- Prior work mainly focuses on web pages and mobile screens.
- Widgets are compact, context-free micro-interfaces: packing icons, charts, text, and highly stylized colors under strict spatial constraints.
- Unlike web / mobile, widget designs are usually proprietary, lacking public markup and usable (image, code) pairs.

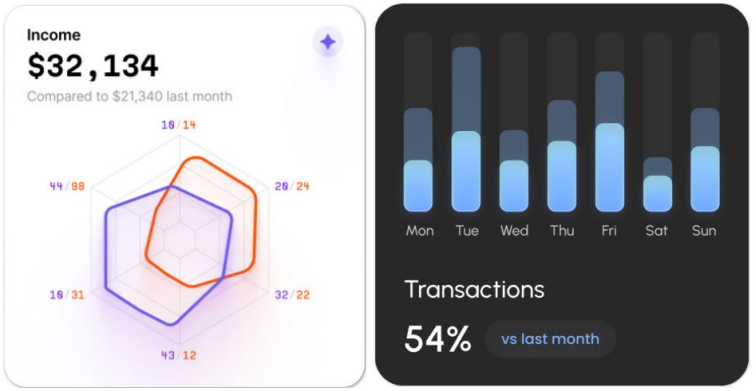
Therefore, we formalize this problem as Widget-to-Code (Widget2Code).



(a) Web UI from Design2Code [23]



(b) App UI from RICO [7]



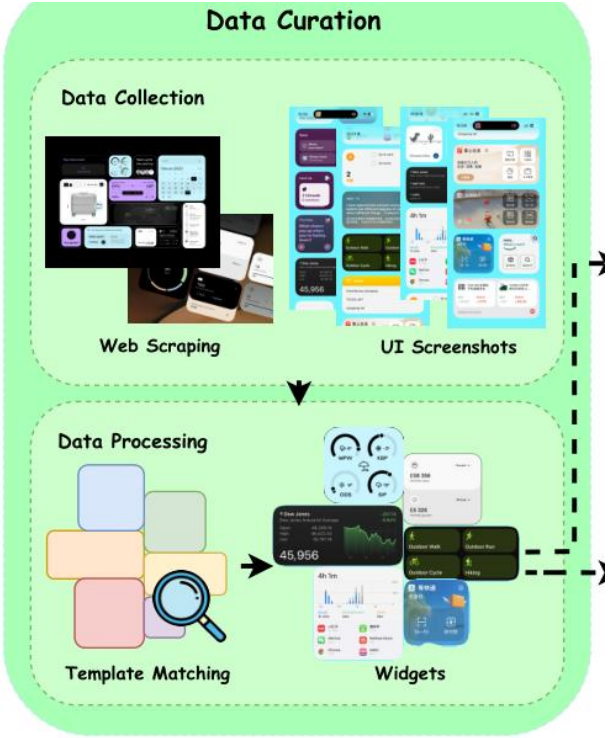
(c) Samples of widgets from our dataset.

Widget2Code benchmark: where does the data come from?

- We crawl widget-related images from Figma, Dribbble, and Refero, and additionally collect screenshots from different devices / OS versions / UI versions.
- Because source images may contain multiple widgets, rotated widgets, or complex backgrounds, we first apply image processing, then detect and crop individual widgets.
- Highly similar samples are then deduplicated and manually verified, yielding the final high-quality set.

Dataset scale

- 12,218 raw samples → 2,825 high-quality widgets
- Final split: 1,000 test / 1,825 development



How to evaluate: 5 groups of fine-grained visual-only metrics

Metric group	Paper metrics	What it checks
Layout	Margin / Content / Area	Whether spacing is symmetric, internal content aspect ratios are ratios are reasonable, and component area proportions are are consistent
Legibility	Text / Contrast / LocCon	Whether text semantics are preserved and overall local foreground-background contrast are consistent
Style	Palette / Vibrancy / Polarity	Whether color distribution, saturation / vividness, and light-dark light-dark polarity match the input
Perceptual	SSIM / LPIPS / CLIP	A reference similarity at the global image level, structural and semantic comparison
Geometry	aspect ratio + normalized size	Whether the original widget size category and overall spatial spatial proportions are reproduced

Key point: these metrics do not rely on GT code, but directly compare the input widget with the final suitable for an image-only benchmark.

6 / 15

Benchmark results: existing methods are still not good enough

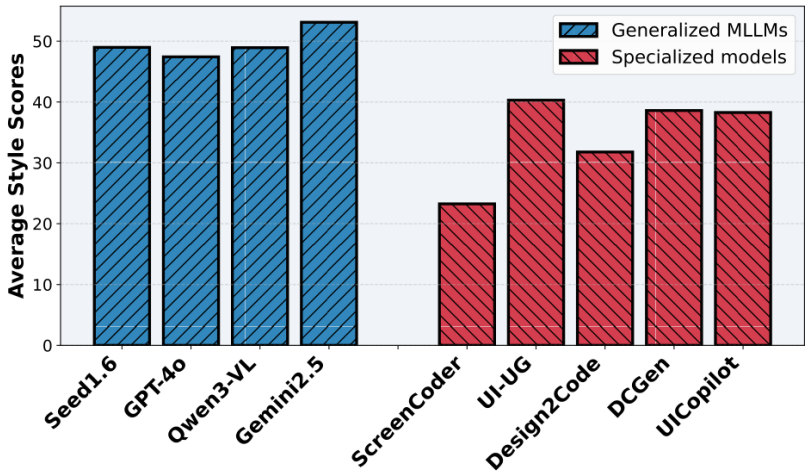


Figure 2. **Style score comparison on our widget benchmark.** Generalized MLLMs outperform specialized UI2Code models, which are tuned for other UI formats instead of widgets.

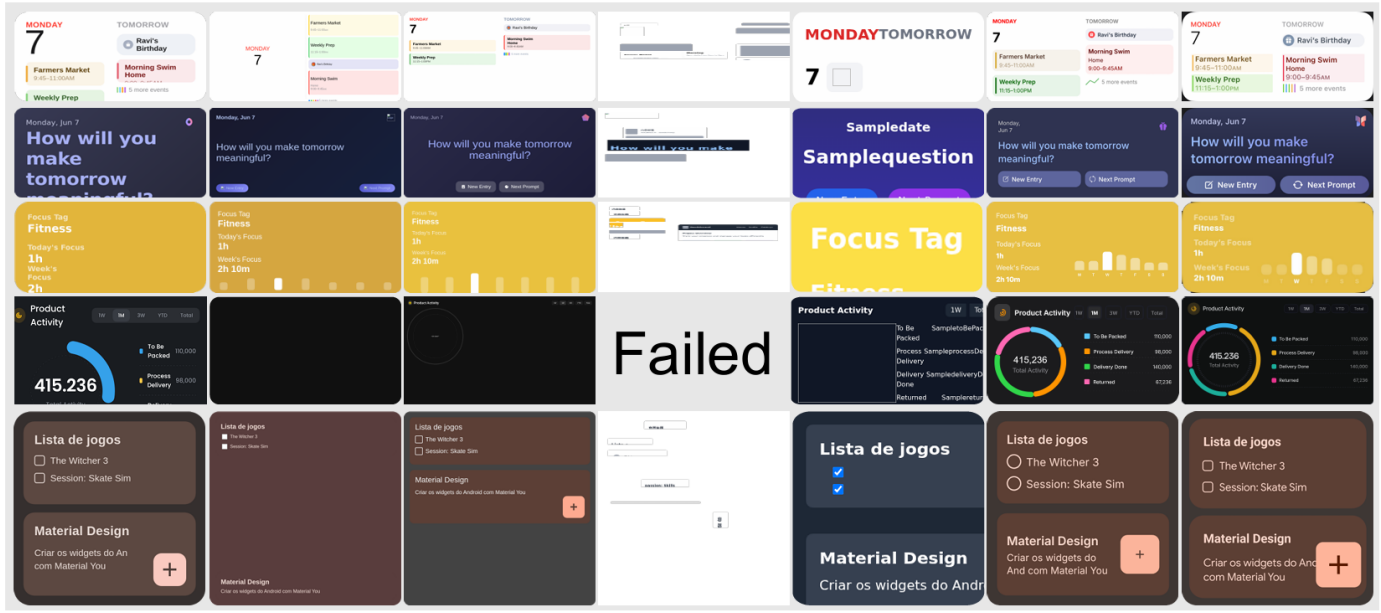
- We benchmark two categories of methods: MLLMs (e.g., GPT-4o / Gemini2.5 / Qwen3-VL) and UI2Code models targeting web/mobile.
- Specialized methods show clear performance degradation on widgets, suggesting they are better suited to looser-structured, context-rich web/mobile UIs.
- Generalized MLLMs have stronger visual grounding but still but still struggle with structural consistency, style accuracy, and accuracy, and precise size reproduction.

The best existing baseline still has Geometry below (Qwen3-VL-235b at 96.28).

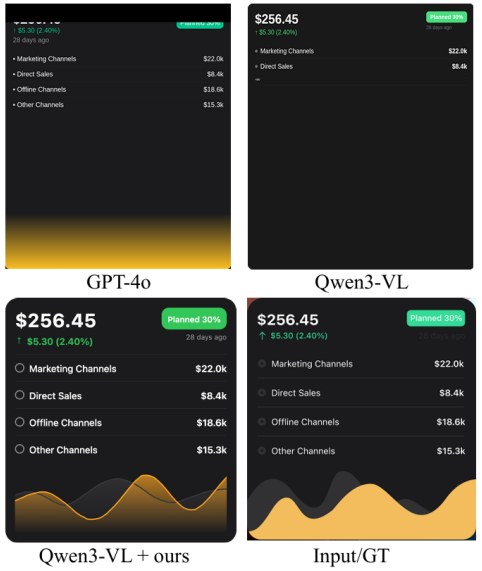
Insight: widgets need more than just "understanding the the screenshot" —they require geometry-aware, controllable controllable code generation.

7 / 15

Where do errors most often occur?



Gemini2.5-Pro | GPT-4o | Qwen3-VL | ScreenCoder | UI-UG | Widget2Code (Ours) | Input / GT



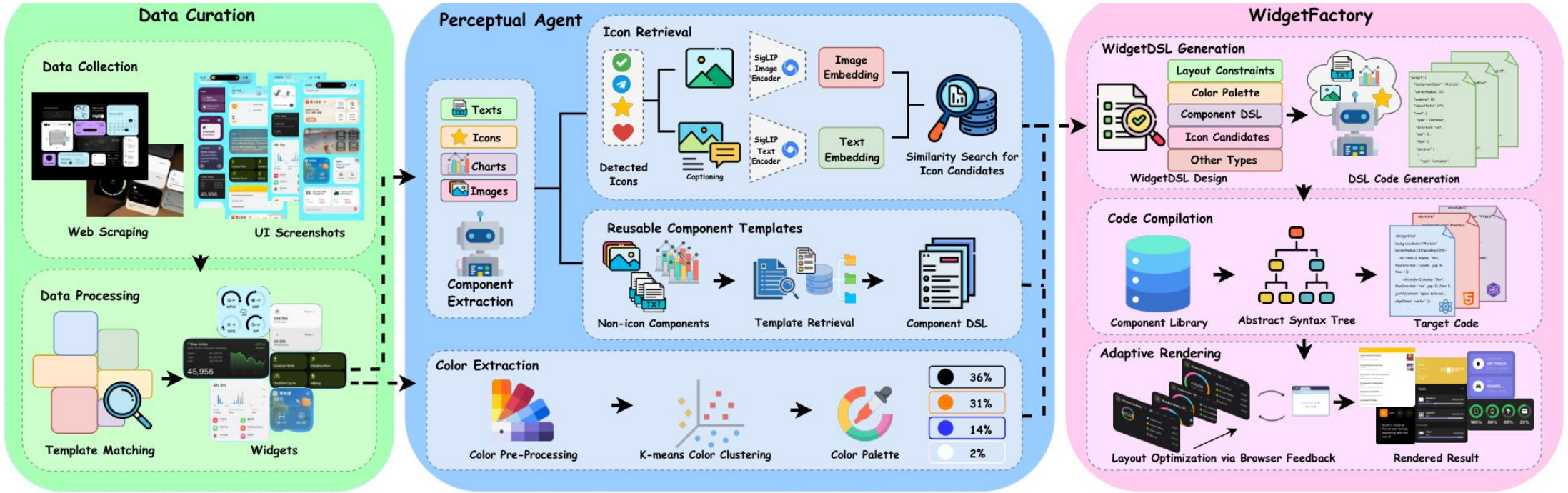
GPT-4o | Qwen3-VL | Qwen3-VL + ours | Input/GT

Typical failure modes

- missing icons / missing components
- distorted graphs / charts
- content overflow / occlusion
- structural misalignment
- color/style inconsistency

Together these errors reveal a core gap: pixel-level perception and geometry-aware, controllable code generation are not yet generation are not yet connected.

Widget2Code overview

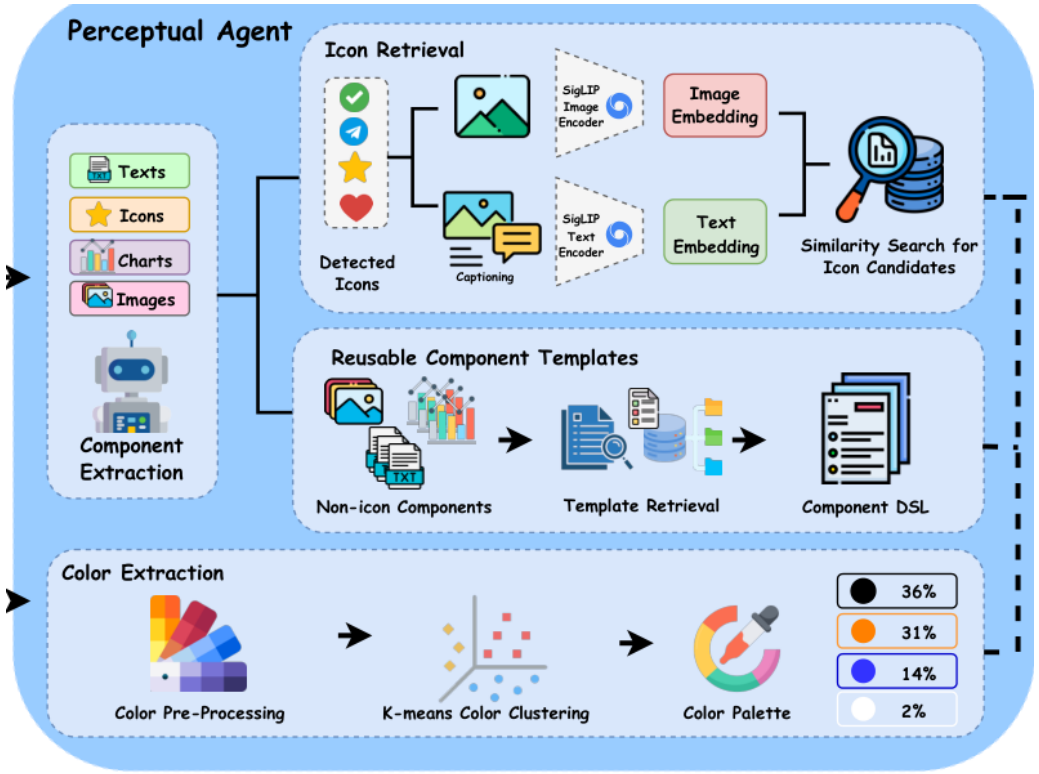


①Data curation: collect and widget images, building the

② Perceptual Agent: decompose into into atomic components and extract icon / extract icon / template / color cues

③ WidgetFactory: generate WidgetDSL, compile, and adaptively render

Perceptual Agent: atomic component decomposition and icon



visual similarity → text reranking → top-5 candidates

- The input widget is first decomposed into atomic components; each detected element is represented t, c], corresponding to cropped region, bounding description, and category.
- We find that: directly letting the MLLM generate icons is unreliable, especially for small icons, thin lines, or abstract graphics, where hallucination and visual inconsistency easily occur.
- Therefore, We build a 50k SVG icon library; first do coarse retrieval with SigLIP visual embedding, then rerank with text embedding.

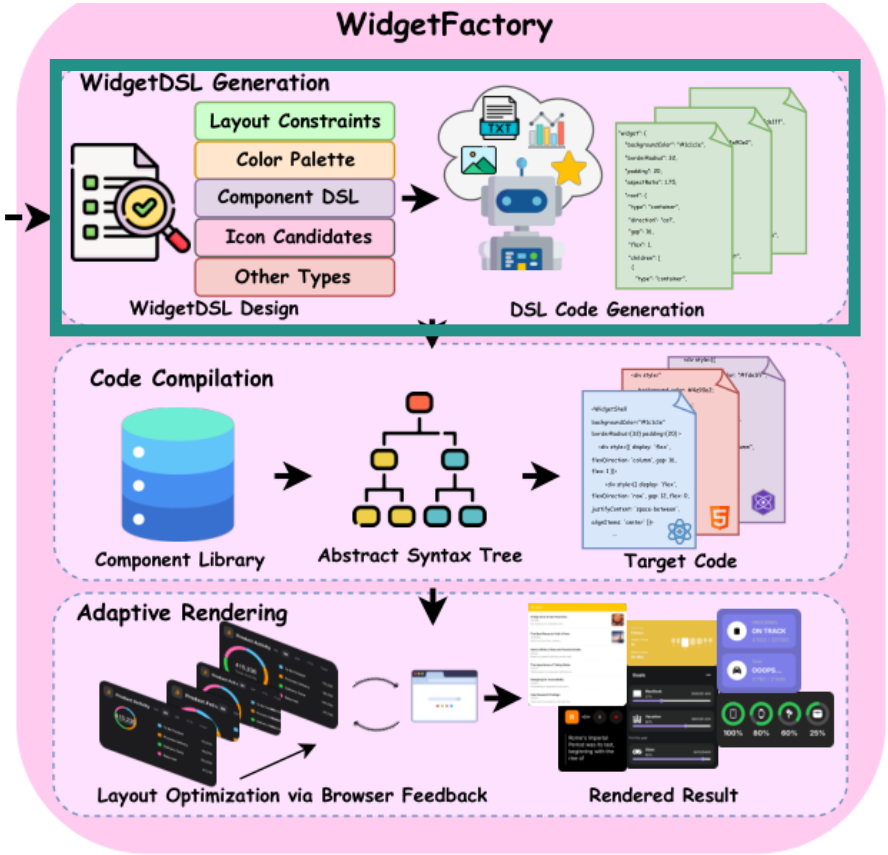
The final top-5 icon candidates are kept for subsequent WidgetDSL context fusion during generation.

WidgetFactory: why generate WidgetDSL first?

- The core intermediate representation is WidgetDSL: compact, interpretable, controllable tree-structured

Each node corresponds to a functional unit (e.g., icon / chart / icon / chart / text block), with attributes explicitly describing describing geometry, color, and style.

Effect: reduces hallucination and redundancy while finer-grained layout / style control.



What does the full system bring?

Table 1. **Benchmarking results of generalized MLLMs and specialized UI2Code methods on our widget test set using the proposed fine-grained evaluation metrics.** Specialized UI2Code models perform poorly as they are optimized for web or mobile layouts rather than compact widget structures, while generalized MLLMs demonstrate stronger adaptability and overall performance.

	Methods	Layout			Legibility			Style			Perceptual			Geometry
		Margin	Content	Area	Text	Contrast	LocCon	Palette	Vibrancy	Polarity	SSIM	LPIPS↓	CLIP	
Generalized	Seed1.6-Thinking	65.02	59.19	74.88	62.82	62.10	55.87	47.06	44.38	55.52	0.644	0.341	0.834	94.24
	Gemini2.5-Pro	65.35	62.74	79.66	59.48	62.64	60.52	48.99	47.77	62.54	0.701	**0.309**	**0.844**	90.25
	GPT-4o	63.48	59.04	64.41	60.20	57.10	53.65	47.03	42.08	53.15	0.698	0.338	0.780	91.93
	Qwen3-VL	64.75	60.15	69.53	61.17	60.87	61.12	47.44	44.50	54.85	0.703	0.334	0.800	95.15
	Qwen3-VL-235b	66.10	60.25	69.93	61.94	60.34	61.73	47.97	45.33	54.06	0.694	0.336	0.803	96.28
Specialized	Design2Code	36.34	47.81	49.68	17.50	52.90	18.31	30.92	31.89	32.54	0.512	0.494	0.610	15.72
	DCGen	43.17	40.14	64.55	50.36	52.13	35.02	35.56	26.21	54.03	0.598	0.400	0.753	31.59
	LatCoder	41.25	43.39	76.35	48.75	56.73	34.90	41.61	36.26	49.97	0.595	0.381	0.764	28.22
	UICopilot	59.20	6.95	33.94	40.94	55.19	42.56	39.22	35.08	40.51	0.709	0.354	0.691	28.40
	WebSight-VLM-8B	32.99	22.46	54.50	1.37	54.24	28.45	31.92	31.78	35.04	0.536	0.478	0.536	27.56
	ScreenCoder	22.19	11.46	31.04	13.77	25.35	24.66	32.15	33.62	3.99	0.101	0.512	0.582	44.56
	UI-UG	52.97	47.90	72.93	12.89	55.66	31.47	38.67	32.36	49.83	0.594	0.403	0.577	23.35
	Widget2Code (Ours)	**72.15**	**66.08**	**82.24**	**70.60**	**66.20**	**64.06**	**58.09**	**51.38**	**63.28**	**0.721**	0.335	0.838	**100.00**

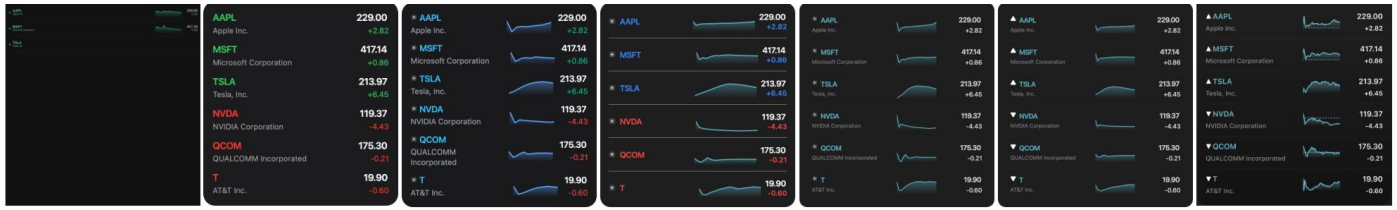
- Compared with generalized / specialized the full system more stably reproduces layout, readability, and color style.
- It reduces missing components and and aligns icons, graphs, and colors more more closely with the input.
- It preserves the original widget input size without losing content or overflowing margins.

Takeaway: it is not simply swapping in a stronger MLLM, but using "perceptual decomposition + structured generation + structured generation + rendering feedback" to obtain better results.

Ablation study: how each module contributes

Table 2. **Ablation analysis on core modules of our baseline.** Start from the baseline, and we integrate one module into the system.

Methods	Layout			Legibility			Style			Perceptual			Geometry
	Margin	Content	Area	Text	Contrast	LocCon	Palette	Vibrancy	Polarity	SSIM	LPIPS↓	CLIP	
Qwen3-VL (baseline)	64.75	60.15	69.53	61.17	60.87	61.12	47.44	44.50	54.85	0.703	0.334	0.800	95.15
+ WidgetFactory	69.97	64.60	82.46	67.99	61.53	57.05	42.36	42.44	58.81	0.683	0.339	0.837	100
+ Components	70.83	64.90	82.30	67.49	61.44	57.75	42.61	41.10	59.30	0.676	0.344	0.835	100
+ Color analysis	71.29	65.43	83.03	68.62	63.25	64.16	57.56	50.71	62.67	0.705	0.338	0.845	100
+ Layout	71.49	65.33	81.92	68.89	63.64	64.21	56.10	49.84	62.54	0.710	0.340	0.837	100
+ Icon (Full)	72.15	66.08	82.24	70.60	66.20	64.06	58.09	51.38	63.28	0.721	0.335	0.838	100



Baseline +WidgetFactory +Components +Layout +Color +Icon (Full) Input/GT

Baseline = Qwen3-

- Starting from the Qwen3-VL baseline, we baseline, we add modules back one by one: by one: WidgetFactory → Components → Components → Color / Layout → Icon (Full). Icon (Full).
- After +WidgetFactory, Geometry already rises to 100, showing that system-level infrastructure and the rendering mechanism are themselves crucial.
- + Components improves chart reconstruction; + Color analysis brings the overall palette closer to the input; the paper provides intuitive examples in Fig. 8.

Although a few combinations show minor conflicts, full integration is still consistently best.

Summary: 3 core takeaways of this paper

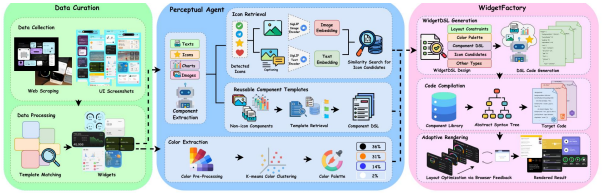
1. It is the first systematic study of widget-focused UI2Code, providing UI2Code, providing an image-only dataset and fine-grained metrics grained metrics truly tailored to widget visual properties.

2. Benchmarking results show that UI2Code methods designed for designed for web/mobile do not transfer naturally to compact, compact, context-free widgets.

3. The key to a strong baseline is not " directly producing longer code" , but the synergy of perceptual decomposition, WidgetDSL, compiler, and adaptive rendering.

Future work

- Stronger fine-grained visual grounding, especially for dense layouts and micro-visualizations.
- Better intermediate representations to further reduce hallucination and extend to interactive / dynamic widgets.
- Larger-scale, more realistic data with greater style and device diversity.



Thank you!

We welcome questions, discussion, and collaboration.

Project page
https://djanghao.github.io/widget2code

Paper
arXiv: 2512.19918