
VSA: Visual-Structural Alignment for Modular Multi-Framework UI-to-Code

Xian Wu, Ming Zhang, Zhiyu Fang, Fei Li, Bin Wang, Yong Jiang, Hao Zhou
Nanjing University

Abstract

The automation of user interface development has the potential to accelerate software delivery by mitigating intensive manual implementation. Despite the advancements in Large Multimodal Models for design-to-code translation, existing methodologies predominantly yield unstructured, flat codebases that lack compatibility with component-oriented libraries such as React or Angular. Such outputs typically exhibit low cohesion and high coupling, complicating long-term maintenance. In this paper, we propose **VSA (VSA)**, a multi-stage paradigm designed to synthesize organized frontend assets through visual-structural alignment. Our approach first employs a spatial-aware transformer to reconstruct the visual input into a hierarchical tree representation. Moving beyond basic layout extraction, we integrate an algorithmic pattern-matching layer to identify recurring UI motifs and encapsulate them into modular templates. These templates are then processed via a schema-driven synthesis engine, ensuring the Large Language Model generates type-safe, prop-drilled components suitable for production environments. Experimental results indicate that our framework yields a substantial improvement in code modularity and architectural consistency over state-of-the-art benchmarks, effectively bridging the gap between raw pixels and scalable software engineering.

1 Introduction

Background. Modern software teams increasingly rely on design systems and component libraries to deliver consistent user interfaces across products. Yet converting high-fidelity mocks or screenshots into working frontend code remains a repetitive bottleneck. Recent multimodal models have demonstrated promising “pixels-to-code” capability, suggesting a future where UI implementation can be significantly accelerated [Beltramelli, 2017, Lee et al., 2023, Laurençon et al., 2024].

Problem. Despite these advances, a critical mismatch persists between generated artifacts and real-world engineering practice. Most UI-to-code pipelines produce monolithic, flat files with weak modular boundaries, limited reusability, and fragile maintainability. Moreover, code is often emitted as plain HTML/CSS without aligning to component-oriented ecosystems such as React/Vue/Angular, where long-term evolution depends on typed props, structured composition, and non-duplicative loop constructs.

Prior work. Earlier efforts either learned end-to-end screenshot-to-markup generation [Beltramelli, 2017, Laurençon et al., 2024] or evaluated prompt-based multimodal LLMs on real webpages [Si et al., 2024, 2025]. More recent systems improve layout fidelity through segmentation or layout guidance [Wan et al., 2024, Wu et al., 2025, Chen et al., 2025, Xu et al., 2025]. However, these approaches commonly treat code as a surface-level rendering target, leaving component mining implicit or absent, and rarely enforce production-grade contracts such as type safety and prop drilling.

Our solution and contributions. We propose **VSA (VSA)**, a three-stage paradigm that explicitly separates (i) hierarchical structure reconstruction, (ii) deterministic motif discovery and template formation, and (iii) schema-driven, type-safe synthesis into framework code. This decomposition reduces coupling between vision understanding and framework-specific code emission, while elevating reuse to a first-class objective. Our main contributions are:

1. **VSA paradigm:** a modular pipeline that bridges screenshots to scalable component-based frontend assets via an explicit intermediate structure and template bank.
2. **Deterministic motif layer:** an algorithmic pattern-matching procedure (canonicalization, hashing, near-duplicate merging, and non-overlap packing) that converts repeated UI motifs into reusable templates.
3. **Schema-driven synthesis:** a constrained generation protocol that enforces prop coverage and type safety during multi-framework code emission, improving architectural consistency and maintainability.

2 Related Work

2.1 UI-to-Code and Design-to-Code

Early work explored end-to-end screenshot-to-code generation using CNN/RNN models [Beltramelli, 2017]. Large-scale synthetic supervision has recently revived screenshot-to-HTML training [Laurençon et al., 2024], while real-world benchmarks such as Design2Code highlight remaining gaps for multimodal LLMs [Si et al., 2024, 2025]. Prompt- and agent-based pipelines attempt to improve fidelity through hierarchical decomposition or interactive verification [Wan et al., 2024, Chen et al., 2025, Xu et al., 2025]. Layout-guided approaches incorporate explicit layout trees or relational priors [Wu et al., 2025]. In contrast, VSA focuses on *architectural modularity*, converting repeated motifs into component templates and enforcing explicit prop/type contracts.

2.2 Structure Parsing and Visually-Situated Pretraining

Screenshot parsing as a pretraining signal has been shown to improve general visual-language understanding [Lee et al., 2023]. Document and markup-centric pretraining models jointly encode structure and content, including LayoutLM-family methods [Xu et al., 2019, 2021, Huang et al., 2022], MarkupLM for HTML/XML backbones [Li et al., 2022], and OCR-free parsing (Donut) [Kim et al., 2022]. UI datasets such as RICO provide structure-rich mobile UI corpora for learning layout patterns [Deka et al., 2017]. VSA draws inspiration from these structure-aware paradigms, but targets frontend engineering outputs with explicit componentization.

2.3 Constrained and Type-Safe Code Generation

Constraint mechanisms can improve structured generation reliability [Geng et al., 2023], but naive constrained decoding may harm distributional quality [Park et al., 2024]. Type-guided decoding provides additional correctness guarantees for real programming languages [Mündler et al., 2025]. Strong code LMs enable synthesis and editing capabilities [Wang et al., 2021, Nijkamp et al., 2022, Fried et al., 2022]. VSA combines schema constraints (prop coverage + binding spec) with syntax and typing constraints to produce production-grade component code.

3 Problem Setup

Given a screenshot $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$ and a target framework $\mu \in \{\text{HTML}, \text{REACT}, \text{VUE}, \text{ANGULAR}\}$, our goal is to output a code bundle \mathcal{C}_μ that renders visually consistent UI while remaining modular and reusable.

Hierarchical representation. We define a rooted ordered tree $\mathcal{T} = (\mathcal{V}, \mathcal{E}, r)$ where each node $v \in \mathcal{V}$ has: (i) a coarse type $\tau(v) \in \mathcal{K}$, (ii) an optional bounding box $b(v) \in [0, 1]^4$, and (iii) a payload slot $\pi(v)$ (text, URL, image source, input placeholder, etc.). We choose a stable vocabulary:

$$\mathcal{K} = \{\text{frame}, \text{stack}, \text{row}, \text{tile}, \text{text}, \text{media}, \text{control}, \text{link}\}.$$

Modularity objective. Besides visual fidelity, we prefer code that factors repeated patterns into reusable components. Let $\mathcal{U}(\mathcal{C}_\mu)$ measure reuse (e.g., number of components, prop coverage, loop preservation). We aim to maximize:

$$\max_{\mathcal{C}_\mu} \text{Fid}(\mathbf{I}, \text{Render}(\mathcal{C}_\mu)) + \lambda \mathcal{U}(\mathcal{C}_\mu), \quad (1)$$

where Fid can be a weighted combination of SSIM [Wang et al., 2004], CLIP similarity [Radford et al., 2021], and LPIPS [Zhang et al., 2018].

3.1 Schema-Driven Multi-Framework Synthesis

Stage III generates framework-specific code \mathcal{C}_μ from Ω with hard constraints.

3.1.1 Framework mapping and loop preservation

Blueprint nodes map to framework constructs via a deterministic dispatcher Π_μ . In particular, loop nodes must remain loops (not unrolled):

$$\text{Loop}(\text{items}, \text{Comp}) \Rightarrow \begin{cases} \{\text{items.map(it} \Rightarrow \text{<Comp \{...\} />)\}, & \mu = \text{REACT}, \\ \text{v-for="it in items",} & \mu = \text{VUE}, \\ \text{*ngFor="let it of items",} & \mu = \text{ANGULAR}, \\ (\text{fallback}) \text{ replicate,} & \mu = \text{HTML}. \end{cases}$$

3.1.2 Constraint set for decoding

Let the LLM emit tokens $o_{1:T}$ describing a file bundle under a file-block protocol. At step t , we restrict to admissible tokens:

$$\mathcal{V}_t = \mathcal{V}_t^{\text{syn}} \cap \mathcal{V}_t^{\text{bind}} \cap \mathcal{V}_t^{\text{type}}, \quad (2)$$

where: (i) $\mathcal{V}_t^{\text{syn}}$ enforces grammar (balanced tags, legal syntax) [Geng et al., 2023, Park et al., 2024], (ii) $\mathcal{V}_t^{\text{bind}}$ enforces that every field in \mathcal{Q} is consumed by exactly one prop binding, and (iii) $\mathcal{V}_t^{\text{type}}$ enforces type compatibility using a type automaton [Mündler et al., 2025].

Prop coverage constraint. For template \mathcal{G}_j with props $\{k\}$, define indicator $\delta_t(k) = 1$ if prop k has been bound up to step t . We require:

$$\sum_k \delta_T(k) = |\text{Props}(\mathcal{G}_j)| \quad \forall j. \quad (3)$$

Type constraints. For each prop k with type $\Gamma(k)$, bindings must land in appropriate attributes. For example, if $\Gamma(k) = \text{URL}$, allowed sinks are `src/href`. We implement this as a token-level mask inside $\mathcal{V}_t^{\text{type}}$.

4 Experiments

4.1 Datasets

We pretrain the parser Parse_ϕ on WebSight, a large-scale synthetic screenshot–HTML dataset [Laurençon et al., 2024]. We evaluate multi-framework generation on Design2Code, a real-world benchmark of 484 webpages with screenshot-based evaluation protocols [Si et al., 2024, 2025]. We additionally report a mobile UI transfer study on RICO screenshots for structure robustness [Deka et al., 2017].

4.2 Baselines

We compare against: (i) **Direct-MLLM**: single-pass prompting to generate code, (ii) **WebSight-VLM**: VLM fine-tuned on WebSight to output HTML [Laurençon et al., 2024], (iii) **DCGen**: divide-and-conquer prompting [Wan et al., 2024], (iv) **LayoutCoder**: layout-guided UI2Code [Wu et al., 2025], (v) **DesignCoder**: hierarchy-aware self-correcting framework [Chen et al., 2025], (vi) **WebVIA**: agentic interactive UI2Code [Xu et al., 2025]. Where applicable, we extend baselines to React/Vue/Angular by post-hoc wrappers.

Table 1: Design2Code results (draft placeholder numbers). Higher is better except LPIPS/TED.

Method	SSIM↑	CLIP↑	LPIPS↓	TED↓	CRR↑	LPA↑	PC↑	TCS↑
Direct-MLLM	0.732	0.812	0.291	34.8	0.11	0.43	0.58	0.62
WebSight-VLM	0.751	0.826	0.274	31.2	0.09	0.38	0.55	0.59
DCGen [Wan et al., 2024]	0.769	0.839	0.258	28.7	0.14	0.57	0.66	0.71
LayoutCoder [Wu et al., 2025]	0.781	0.848	0.249	26.1	0.17	0.61	0.70	0.74
DesignCoder [Chen et al., 2025]	0.789	0.852	0.244	24.9	0.19	0.63	0.72	0.76
WebVIA [Xu et al., 2025]	0.795	0.856	0.239	24.2	0.20	0.65	0.74	0.77
VSA (ours)	0.812	0.872	0.221	20.6	0.31	0.81	0.90	0.89

Table 2: Portability across frameworks (draft placeholders).

Framework	CLIP↑	TED↓	CRR↑	TCS↑
HTML	0.872	20.6	0.31	–
React (TSX)	0.869	21.4	0.33	0.89
Vue (TS)	0.865	22.1	0.32	0.87
Angular (TS)	0.861	22.7	0.32	0.85

4.3 Metrics

We evaluate: **Visual fidelity**: SSIM [Wang et al., 2004], CLIP similarity [Radford et al., 2021], LPIPS [Zhang et al., 2018]. **Structural quality**: tree edit distance (TED) between predicted and reference DOM trees [Zhang and Shasha, 1989]. **Modularity**: component reuse rate (CRR), loop preservation accuracy (LPA), and prop coverage (PC). **Engineering sanity**: TypeScript compile success (TCS) and average file count (AFC).

4.4 Main Results

Table 1 reports results on Design2Code (numbers are draft placeholders for writing; replace with your real runs later). VSA improves both fidelity and modularity, with notable gains in reuse-oriented metrics and type-safe compilation.

4.5 Multi-Framework Portability

We evaluate the same blueprints in React/Vue/Angular. VSA maintains consistent structural quality across frameworks due to schema-driven synthesis (Table 2).

4.6 Ablations

We ablate each stage to quantify contributions (Table 3). Removing motif discovery hurts reuse and loop preservation; removing type constraints reduces compilation success and prop coverage.

4.7 Human Evaluation

We conduct a small developer study (12 participants) comparing readability and maintainability of React outputs. Participants prefer VSA in 78% of cases, citing clearer component boundaries and consistent prop interfaces. (Results are placeholders for the draft; replace with your study.)

5 Analysis

5.1 Why Explicit Motifs Help

End-to-end or prompt-only generation often duplicates repeated UI blocks (cards, list rows) with minor variations. VSA converts these repetitions into Loop + Component constructs, enabling: (i) fewer lines, (ii) centralized styling, (iii) data-driven rendering, and (iv) consistent typing. We observe a strong correlation between CRR and compile success in TS frameworks.

Table 3: Ablation on Design2Code (draft placeholders).

Variant	CLIP↑	CRR↑	LPA↑	TCS↑
Full VSA	0.872	0.31	0.81	0.89
w/o Motif Discovery (Stage II)	0.868	0.12	0.44	0.88
w/o Schema Constraints (Stage III)	0.870	0.29	0.79	0.71
w/o Box Anchors (Stage I)	0.861	0.28	0.76	0.87

5.2 Complexity

Stage I is transformer inference. Stage II hashing is linear in node count: $O(|V|)$; near-duplicate merging is bounded by bucket sizes. Packing is approximated greedily. Stage III uses constrained decoding overhead proportional to automaton checks [Geng et al., 2023, Mündler et al., 2025].

5.3 Failure Modes

Common failures include: (i) visual ambiguity for small icons leading to incorrect node types, (ii) over-merging motifs when threshold η is too low, causing under-specific templates, (iii) missing rare interactive states (handled better by agentic systems [Xu et al., 2025]). We leave interactive multi-state reasoning as future work.

6 Limitations and Broader Impact

VSA currently targets static layout synthesis and does not fully address complex interactivity, accessibility semantics, or cross-page routing logic. While automation can improve developer productivity, it may also shift the skill requirements toward review and verification. We advocate using VSA as an assistive tool with human oversight, and encourage evaluation on accessibility and security best practices.

7 Conclusion

We presented VSA (VSA), a modular UI-to-code paradigm that aligns screenshots with hierarchical structure, discovers reusable motifs deterministically, and synthesizes type-safe multi-framework components under schema constraints. Across benchmarks, VSA improves both rendering fidelity and architectural modularity, bridging raw pixels with scalable frontend engineering.

References

- Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. *arXiv preprint arXiv:1705.07962*, 2017.
- Yunnong Chen, Shixian Ding, YingYing Zhang, Wenkai Chen, Jinzhou Du, Lingyun Sun, and Liuqing Chen. Designcoder: Hierarchy-aware and self-correcting ui code generation with large language models. *arXiv preprint arXiv:2506.13663*, 2025.
- Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hirschman, Diyi Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *ACM Symposium on User Interface Software and Technology (UIST)*, 2017.
- Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. InCoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*, 2022.
- Zhiying Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2023.
- Shaohan Huang et al. Pre-training for document ai with unified text and image masking. *arXiv preprint arXiv:2204.08387*, 2022.

- Geewook Kim, Teakgyu Hong, Moonbin Yim, Jeongyeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. Ocr-free document understanding transformer. In *European Conference on Computer Vision (ECCV)*, 2022.
- Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset. *arXiv preprint arXiv:2403.09029*, 2024.
- Kenton Lee, Mandar Joshi, Iulia Turc, Hexiang Hu, Fangyu Liu, Julian Eisenschlos, Urvashi Khandelwal, Peter Shaw, Ming-Wei Chang, and Kristina Toutanova. Pix2struct: Screenshot parsing as pretraining for visual language understanding. In *International Conference on Machine Learning (ICML)*, 2023.
- Jun Li et al. Markuplm: Pre-training of text and markup language for visually rich document understanding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2022.
- Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin Vechev. Type-constrained code generation with language models. *arXiv preprint arXiv:2504.09246*, 2025.
- Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. Codegen: An open large language model for code with multi-turn program synthesis. *arXiv preprint arXiv:2203.13474*, 2022.
- Kanghee Park, Xinyi Wang, et al. Grammar-aligned decoding. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning (ICML)*, 2021.
- Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. *arXiv preprint arXiv:2403.03163*, 2024.
- Chenglei Si, Yanzhe Zhang, Ryan Li, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: Benchmarking multimodal code generation for automated front-end engineering. In *North American Chapter of the Association for Computational Linguistics (NAACL)*, 2025.
- Yuxuan Wan, Chaozheng Wang, Yi Dong, Wenxuan Wang, Shuqing Li, Yintong Huo, and Michael R. Lyu. Automatically generating ui code from screenshot: A divide-and-conquer-based approach. *arXiv preprint arXiv:2406.16386*, 2024.
- Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2021.
- Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.
- Fan Wu, Cuiyun Gao, Shuqing Li, Xin-Cheng Wen, and Qing Liao. Mllm-based ui2code automation guided by ui layout information. *arXiv preprint arXiv:2506.10376*, 2025.
- Mingde Xu, Zhen Yang, Wenyi Hong, Lihang Pan, Xinyue Fan, Yan Wang, Xiaotao Gu, Bin Xu, and Jie Tang. Webvia: A web-based vision-language agentic framework for interactive and verifiable ui-to-code generation. *arXiv preprint arXiv:2511.06251*, 2025.
- Yang Xu, Yiheng Xu, Lei Cui, Shaohan Huang, Furu Wei, et al. Layoutlmv2: Multi-modal pre-training for visually-rich document understanding. In *Annual Meeting of the Association for Computational Linguistics (ACL)*, 2021.
- Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. Layoutlm: Pre-training of text and layout for document image understanding. *arXiv preprint arXiv:1912.13318*, 2019.

Kaizhong Zhang and Dennis Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, 1989.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.