WebUI-95: A Large-Scale Dataset of Normalized Web Interfaces via UI-to-Code Generation

ANONYMOUS AUTHOR(S)

Large-scale web UI datasets are essential resources for training and evaluating machine learning models in user interface research, yet real-world web scraping produces data consisting of framework boilerplate, build artifacts, and deeply nested elements, resulting from software engineering practices. These complexities conflict with learning design patterns and complicate code understanding and manipulation. Recent advances in vision-language modelling and verifiable reward have enabled a new generation of UI-to-code systems that produce high-fidelity reproductions of web interfaces from screenshots. In this work, we apply a state-of-the-art UI-to-code model to the WebUI dataset. Our analysis shows that this transformation achieves 95% visual fidelity to the original website while reducing code size by 95% and improving visual quality. We release the dataset and generation pipeline to support research in layout modeling, code generation, and design research.

CCS Concepts: • **Human-centered computing** → **Interactive systems and tools**; **Web-based interaction**; • **Computing methodologies** → *Natural language generation.*

Additional Key Words and Phrases: web interfaces, datasets, UI-to-code generation, code normalization, visual fidelity, HTML

ACM Reference Format:

Anonymous Author(s). 2026. WebUI-95: A Large-Scale Dataset of Normalized Web Interfaces via UI-to-Code Generation. 1, 1 (January 2026), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Data-driven approaches to user interface modeling have long faced a fundamental trade-off between data quality and scale. Early work such as Pix2Code [1] relied on small synthetic datasets with clean, well-structured code, which enabled controlled experiments but limited the diversity of designs that models could learn from. As the field matured, researchers turned to web-scale scraping to assemble larger collections like RICO [2] for mobile applications and WebUI [11] and WebCode2M [4] for web interfaces, enabling progress on tasks from layout prediction to code generation. However, these real-world datasets introduced a different problem as the underlying HTML documents carry substantial complexity unrelated to visual design, including framework-specific boilerplate, build system artifacts, and deeply nested wrapper elements that exist to satisfy software engineering constraints but do not directly relate to design intent, therefore introducing noise into the modelling objectives [3].

Several approaches have attempted to fill the gap between synthetic simplicity and real-world diversity. Some works apply aggressive HTML pruning to remove external dependencies [8], while others generate synthetic data from templates [5]. However, pruning often discards styling information that affects visual appearance, and synthetic data lacks the diversity found in real-world designs. UI-to-code generation models offer an alternative path as they reconstruct clean code directly from screenshots. Early prototypes such as Pix2Code [1] demonstrated small-scale feasibility but lacked sufficient accuracy for practical use. Recent work combining vision-language models [7] with

Manuscript submitted to ACM

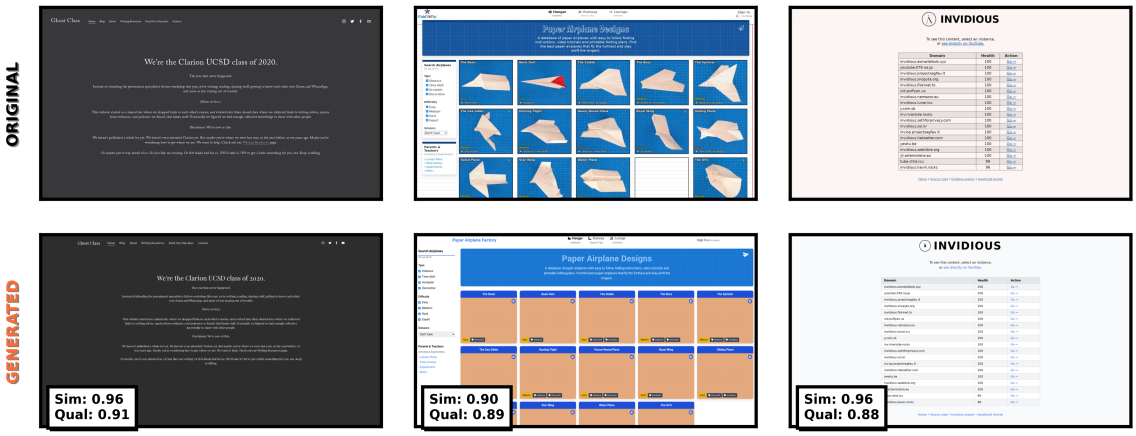

Fig. 1. Qualitative comparison of original (top row) and generated (bottom row) interfaces across three randomly sampled examples demonstrating reconstruction quality. Each generated output shows the visual similarity score (Sim) and quality score (Qual) computed via UIClip.

reinforcement learning from visual feedback has improved fidelity considerably, with models such as Design2Code [8] and UI2Code_N [12] now reproducing complex web interfaces with high accuracy.

In this work, we apply UI2Code_N[1] to create WebUI-95, a dataset derived from the WebUI-75k corpus. We analyse a sample of 10,000 interfaces and related code, showing that this transformation preserves 95% visual fidelity while reducing code volume by approximately 95% (Figure 1). Figure 1 presents three representative examples demonstrating the model's normalization capabilities.

The generated interfaces show slightly improved visual quality scores on average, which we attribute to the model reconstructing broken or partially rendered pages into well-formed designs. To verify that our approach maintains representational diversity necessary for downstream generalization, we compare visual and code diversity against purely synthetic alternatives, which represent the current standard for clean frontend datasets [5, 6]. Results show that WebUI-95 maintains comparable visual diversity to both original and synthetic datasets while exhibiting higher code diversity than synthetic baselines. We release WebUI-95 along with our evaluation code[2] and plan to extend coverage to other datasets, such as WebCode2M and RICO.

2 Methodology

We constructed WebUI-95 by processing the WebUI dataset in its 75k release using UI2Code_N[3]. The source dataset provides pairs of web screenshots and their corresponding HTML source code in different aspect ratios. For each sample, we provided the original screenshot to the UI-to-code model and collected its generated React component output. The model generates JSX-style React code with Tailwind CSS classes, which is structurally very close to plain HTML. To obtain the final DOM for analysis, we use Playwright[4] to render each component and extract the resulting static

[1]https://huggingface.co/zai-org/UI2Code_N
[2]The WebUI-95 dataset and generation pipeline are available at https://repo.paperbackwriters.club:8443/code/WebUI-95.
[3]https://huggingface.co/zai-org/UI2Code_N
[4]https://playwright.dev

HTML, stripping away script imports and converting JSX attributes to standard HTML. All interfaces, both original and generated, are rendered at a fixed resolution of 1920 by 1080 pixels to standardize the visual analysis.

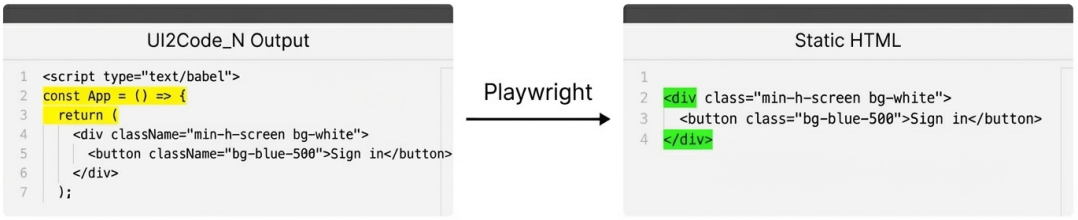

Fig. 2. React-to-HTML transformation pipeline. UI2Code_N generates JSX components with Tailwind CSS utility classes defined via className attributes (left). Playwright renders the component in a headless browser and extracts the final static HTML (right), preserving Tailwind classes while removing React boilerplate.

Visual Quality and Fidelity Metrics. To assess the visual characteristics of the generated interfaces, we employed UIClip [10], a CLIP-based model fine-tuned specifically on UI screenshots. UIClip provides two complementary measurements: a scalar quality score that predicts the visual appeal of an interface based on learned design principles, and a cosine similarity score between the embeddings of generated and original screenshots that quantifies visual fidelity. We report this similarity score as our primary measure of how faithfully the generation preserves the visual appearance of the source interface.

HTML Structure Analysis. We performed a static analysis of both the original and generated HTML documents to quantify the structural differences with the original website code. Using BeautifulSoup[5] to parse the markup, we extracted measurements of code complexity including file size in kilobytes, element count, unique tag count, and maximum nesting depth. We also categorized elements by their semantic role, distinguishing between structural tags, semantic content tags, interactive elements, and media components to understand how the model's output differs in its use of HTML primitives.

Diversity Evaluation. To understand the trade-offs inherent in the normalization process, we assessed intra-dataset diversity in both the visual and code domains. For visual diversity, we computed UIClip embeddings for all screenshots and calculated the mean pairwise cosine distance: $\bar{d} = \frac{2}{n(n-1)} \sum_{i<j} (1 - \cos(e_i, e_j))$, where e_i denotes the embedding of sample i. Since this computation is $O(n^2)$, we evaluated a random subset of 500 samples per dataset. For code diversity, we generated embeddings using Qwen2.5-Coder[6] and similarly computed mean pairwise distances. Alongside the original and generated datasets, we also compared with a synthetic dataset generated using Qwen3[7] with different random seeds and fixed design prompts. This comparison allows us to quantify to what extent our approach preserves diversity compared to purely synthetic generation, a factor that directly impacts downstream model generalization [9].

3 Results

Visual Quality and Fidelity. Our analysis shows that generated interfaces achieve higher predicted visual quality on average, with the distribution shifting towards a mean of 0.518 compared to the original websites' mean of 0.487 (Figure

[5]https://www.crummy.com/software/BeautifulSoup/
[6]https://huggingface.co/Qwen/Qwen2.5-Coder
[7]https://huggingface.co/Qwen/Qwen3-32B

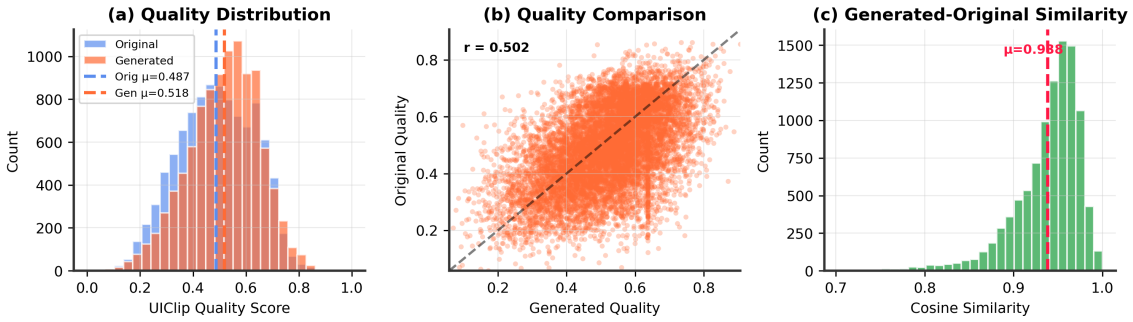

Fig. 3. Visual quality analysis. (a) Distribution of UIClip quality scores. (b) Scatter plot of original vs generated quality. (c) Histogram of cosine similarity scores.

3a). This 6.4 percent improvement is accompanied by a decrease in standard deviation, indicating more consistent design quality across the generated outputs. The quality comparison scatter plot (Figure 3b) shows a moderate positive correlation (r=0.502) between original and generated quality scores, with many generated outputs achieving higher quality than their corresponding originals. Fidelity remains high with a mean cosine similarity of 0.938 between the generated and original screenshots (Figure 3c), indicating that the quality improvement does not compromise visual accuracy.

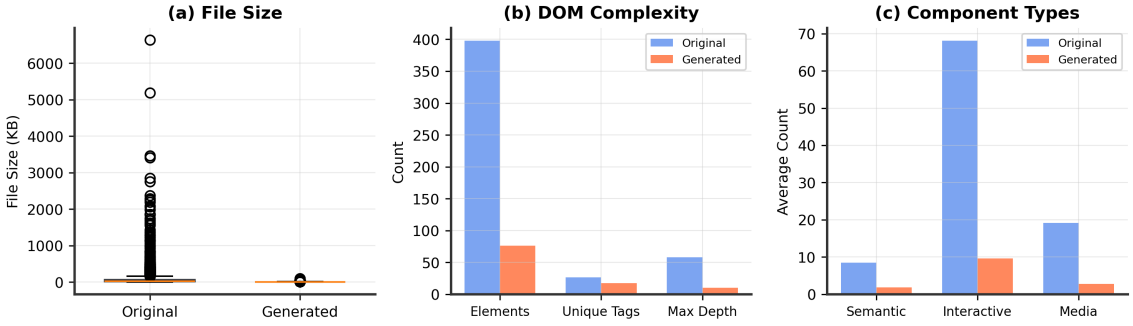

Fig. 4. HTML structure comparison. (a) File size distribution. (b) DOM complexity metrics. (c) Component type analysis.

HTML Structure Analysis. The largest difference lies in the code structure where generated files are on average less than five percent of the size of the originals (Figure 4a). The element count is reduced by approximately 80 percent and the nesting depth is cut by more than half (Figure 4b). The analysis of component types reveals that the model increases the use of semantic HTML elements while reducing reliance on generic divs and media tags (Figure 4c).

Diversity Analysis. We assessed both visual and code diversity to understand the trade-offs of this normalization (Figure 5). Visual diversity remains comparable across all three datasets: original (0.150), generated (0.137), and synthetic (0.140), indicating that the normalization process preserves the visual variety of the source data. In the code domain, the generated dataset exhibits higher diversity (0.054) than the synthetic baseline (0.009), despite using a consistent implementation style.

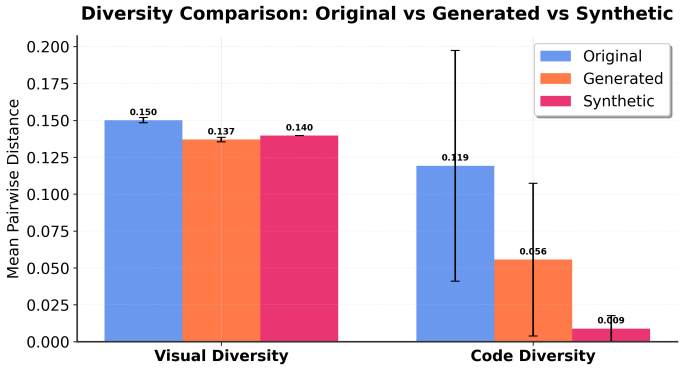

Fig. 5. Diversity comparison across visual and code spaces for original, generated, and synthetic baseline datasets.

4 Discussion

The results indicate that UI-to-code models can serve as effective normalizers, decoupling visual appearance from implementation complexity. The reduction in code volume while maintaining high visual fidelity shows that these models capture design structure, opening several potential application areas.

One direction involves training models for UI element completion, where given a partial interface layout, a model predicts plausible completions for missing regions. Normalized code provides consistent structural patterns that make such predictions more tractable than learning from the heterogeneous implementations found in raw web data. A related application concerns web agents that must navigate and interact with interfaces to accomplish user goals. These agents rely on parsing DOM structures to identify actionable elements, understand page state, and plan interaction sequences. Cleaner document structures reduce the gap between what an agent observes and the underlying semantic intent of the interface, potentially improving both action selection and generalization across websites.

Another promising direction involves reverse engineering interfaces to understand their underlying task structure. By training on pairs of normalized screenshots and code, models may learn to infer the workflow or user goals that a given interface was designed to support, which could then inform the generation of alternative designs for similar tasks. The simplified HTML also has implications for accessibility, since cleaner DOM trees map more directly to the accessibility trees used by screen readers and assistive technologies.

The diversity analysis provides context for interpreting these results. While the generated outputs exhibit reduced code diversity compared to the originals, they maintain higher diversity than purely synthetic alternatives. This distinction matters for downstream training applications, as it indicates that WebUI-95 preserves variation across interface types and design styles. The reduction in code diversity reflects the consolidation of equivalent implementation approaches into a consistent vocabulary.

5 Limitations and Future Work

One limitation of our current approach is the loss of interactivity in the generated outputs. While the original web interfaces contain functional elements such as buttons, forms, and navigation links, the generated code produces static pages that preserve visual appearance but do not retain the underlying behaviors. This preclude the immediate applicability of WebUI-95 to tasks that rely on interaction modeling. Future work should explore methods for recovering

or synthesizing interactive behaviors, potentially through combining UI-to-code generation with program synthesis techniques and automatic navigation.

A secondary limitation concerns the output format of the generation model. UI2Code_N produces static HTML with Tailwind CSS utility classes, which may not suit all downstream applications. Some use cases might prefer minimal React code with reusable components, alternative CSS frameworks, or different markup conventions. The choice of output format is determined by the underlying UI-to-code model, and future work could explore adapting the pipeline to support multiple target formats depending on the intended application.

Looking forward, we plan to extend this approach to mobile interfaces, adapting the methodology to handle the distinct layout constraints and interaction patterns of mobile applications. In addition, we intend to scale the dataset by processing larger web corpora including WebCode2M and extended versions of WebUI, which would increase both the coverage and diversity of WebUI-95.

6 Conclusion

We have presented WebUI-95, a dataset of normalized web interfaces created by applying a UI-to-code generation model to real-world web pages. Our analysis shows that this transformation preserves visual fidelity while reducing code volume to less than five percent of the original, producing semantic HTML that captures design structure without implementation complexity. The generated dataset maintains comparable visual diversity to both original and synthetic alternatives while exhibiting higher code diversity than synthetic baselines, indicating that meaningful structural variation is preserved through the normalization process.

We release WebUI-95 along with our evaluation pipeline to support research in layout modeling, code generation, web agents, and accessibility. Future work will expand coverage to the full WebUI-350k corpus, extend the approach to WebCode2M, and evaluate normalization for mobile interfaces using RICO. We also plan to investigate the impact of normalized training data on downstream task performance.

References

[1] Tony Beltramelli. 2018. pix2code: Generating Code from a Graphical User Interface Screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*. ACM, 1–6. https://doi.org/10.1145/3220134.3220135

[2] Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. 2017. Rico: A Mobile App Dataset for Building Data-Driven Design Applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 845–854. https://doi.org/10.1145/3126594.3126651

[3] Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting Large Webtext Corpora: A Case Study on the Colossal Clean Crawled Corpus. , 1286–1305 pages.

[4] Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Bohua Chen, Dongping Chen, Siyuan Wu, Xing Zhou, Wenbin Jiang, Hai Jin, and Xiangliang Zhang. 2025. WebCode2M: A Real-World Dataset for Code Generation from Webpage Designs. In *Proceedings of the ACM Web Conference 2025*. ACM. https://doi.org/10.1145/3696410.3714889

[5] Hugo Laurençon, Léo Masson, and Alexandre Sablayrolles. 2024. Unlocking the Conversion of Web Screenshots into HTML Code with the WebSight Dataset. *arXiv preprint arXiv:2403.09029* (2024).

[6] Zimu Lu, Yunqiao Yang, Houxing Ren, Haotian Hou, Han Xiao, Ke Wang, Weikang Shi, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. 2025. WebGen-Bench: Evaluating LLMs on Generating Interactive and Functional Websites from Scratch. *arXiv preprint arXiv:2505.03733* (2025).

[7] OpenAI. 2023. GPT-4V(ision) System Card. *OpenAI Technical Report*. https://openai.com/research/gpt-4v-system-card

[8] Chenglei Si, Yanzhe Li, Ruiyang Qian, Yuhao Liu, Zhengyuan Qu, Jiahao Zhu, Pengwei Huang, Nan Ding, Zhou Yu, Wenya Yao, and Yuan Li. 2024. Design2Code: How Far Are We From Automating Front-End Engineering?. In *Proceedings of the 2024 Conference on Neural Information Processing Systems*.

[9] Ben Sorscher, Robert Geirhos, Shashank Shekhar, Surya Ganguli, and S Ariyu. 2022. Beyond neural scaling laws: beating power law scaling via data pruning. In *Advances in Neural Information Processing Systems*, Vol. 35. 24191–24204.

[10] Jason Wu, Yi-Hao Peng, Amanda Xin Yue Li, Amanda Swearngin, Jeffrey Bigham, and Jeffrey Nichols. 2024. UIClip: A Data-driven Model for Assessing User Interface Design. In *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST)*.

[11] Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P. Bigham. 2023. WebUI: A Dataset for Enhancing Visual UI Understanding with Web Semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, 1–14. https://doi.org/10.1145/3544548.3581158

[12] Zhen Yang, Wenyi Hong, Mingde Xu, Xinyue Fan, Weihan Wang, Jiele Cheng, Xiaotao Gu, and Jie Tang. 2025. UI2CodeN: A Visual Language Model for Test-Time Scalable Interactive UI-to-Code Generation. *arXiv preprint arXiv:2505.00000* (2025). Under review.