

Envisioning Future Interactive Web Development: Editing Webpage with Natural Language

Truong Hai Dang
Singapore Management University
Singapore
hdtruong.2022@scis.smu.edu.sg

Jingyu Xiao
The Chinese University of Hong Kong
Hong Kong SAR
jyxiao@link.cuhk.edu.hk

Yintong Huo*
Singapore Management University
Singapore
ythuo@smu.edu.sg

Abstract—The evolution of web applications relies on iterative code modifications, a process that is traditionally manual and time-consuming. While Large Language Models (LLMs) can generate UI code, their ability to edit existing code from new design requirements (e.g., “center the logo”) remains a challenge. This is largely due to the absence of large-scale, high-quality tuning data to align model performance with human expectations. In this paper, we introduce a novel, automated data generation pipeline that uses LLMs to synthesize a high-quality fine-tuning dataset for web editing, named Instruct4Edit. Our approach generates diverse instructions, applies the corresponding code modifications, and performs visual verification to ensure correctness. By fine-tuning models on Instruct4Edit, we demonstrate consistent improvement in translating human intent into precise, structurally coherent, and visually accurate code changes. This work provides a scalable and transparent foundation for natural language-based web editing, demonstrating that fine-tuning smaller open-source models can achieve competitive performance with proprietary systems. We release all data, code implementations, and model checkpoints for reproduction¹.

I. INTRODUCTION

Transforming webpage designs into UI code is a crucial but time-consuming stage in web development [1]. Recent advances in Multimodal Large Language Models (MLLMs) have shown impressive capabilities in generating code from visual inputs [2], [3], [4], [5], [6], [7], [8], [9], opening new possibilities for automating design-to-code conversion [10], [11], [12], [13].

However, editing existing code, rather than creating it from scratch, is central to how the web application evolves. Developers rarely build entire interfaces in a single pass, instead, they iteratively refine existing codebases in response to new design requirements such as “make the layout more minimalist”, “increase spacing here”, or “center the logo”. These modifications are time-consuming for developers and demand deep understanding of the web structure and content. To accelerate this process for software evolution, this paper envisions a future web application development paradigm: *interactive web UI editing through natural language* (i.e., web editing in this paper).

While LLMs have recently excelled at generating UI code, their capacity to edit real-world code based on instructions remains unsatisfied (shown in Fig. 1). This task is challenging

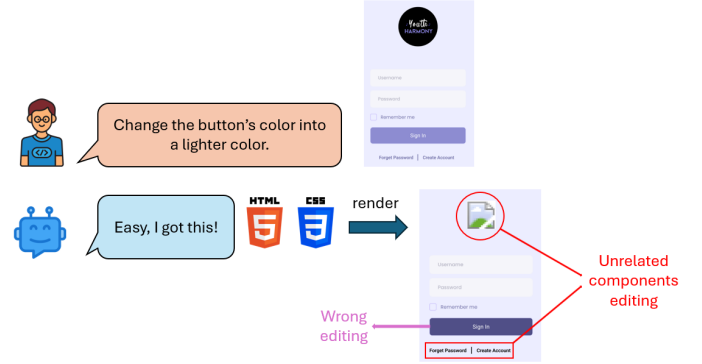


Fig. 1. Example of a Failed HTML Edit Based on a Design Instruction

because it requires LLMs to fully understand HTML, reason about GUI layouts, and comprehend human intent to apply changes. Specifically, we identify three key challenges in automatic web editing:

- **Abstract design instructions** such as “make the layout more minimalist” are inherently vague and require grounding in visual semantics.
- Models must generate **coherent HTML** to apply changes while keeping the rest of the webpage stable, which demands structural consistency and correctness.
- The output must exhibit **visual alignment**, ensuring that the rendered webpage reflects the user’s intended modification.

A core reason for the lack of reliable models is the absence of large-scale, high-fidelity datasets that pair natural language instructions with corresponding HTML/CSS modifications in an end-to-end fashion. Such datasets enable LLMs to learn UI domain knowledge and align with user preferences. However, creating these datasets manually is impractical due to the extensive effort from designers, developers, and annotators to craft instructions, implement edits, and verify correctness.

To bridge this gap, we propose a novel paradigm for automatically synthesizing instruction-tuning datasets. Our pipeline leverages large language models across the full supervision flow—first by eliciting latent design-editing knowledge to generate diverse, human-like instructions, then by producing corresponding HTML code modifications, and finally by conducting automated visual cross-checking to validate

*Yintong Huo is the corresponding author.

¹<https://github.com/dangtruong01/Instruct4Edit>

whether the intended edits have been correctly applied. This self-contained loop allows LLMs to act both as synthetic supervisors and evaluators, transforming implicit UI-editing knowledge into explicit, verifiable training data. The resulting dataset, **Instruct4Edit**, consists of high-quality (instruction, original HTML, modified HTML) samples that are semantically aligned and visually faithful, offering a scalable foundation for fine-tuning code-editing agents.

Using Instruct4Edit, we fine-tune LLMs to perform single-shot HTML rewrites grounded in natural language commands. These models show improved performance in both structure preservation and visual correctness, demonstrating that high-quality synthetic data can effectively translate human intent into concrete code changes. The goal is not to outperform frontier commercial systems, but to demonstrate that smaller open-source models can achieve competitive performance through targeted fine-tuning. Our contributions are as follows:

- We envision a natural language-enabled web editing paradigm for future interactive software development.
- We develop a transparent, fully-automated data synthesis pipeline that trains LLMs to follow user instructions for web editing.
- We evaluate our pipeline on real-world editing jobs, showing approximately a notable 10% improvement for existing models.

II. RELATED WORK

Dataset Synthesis for Instruction Tuning. Instruction tuning of LLMs relies on large-scale datasets where instructions are aligned with desired outputs. Existing datasets like WebSight [14] synthesize screenshot-to-code generation without human-provided instructions. While SelfCodeAlign [15] offers synthetic validation pipelines, it targets functional code rather than UI semantics. Our approach is novel in using an LLM-driven pipeline to synthesize a high-quality dataset for enhancing web UI editing.

UI Code Intelligence. Most existing research [16], [17], [18], [19], [20], [21], [22] focuses on automated UI code generation from visual inputs like screenshots. For instance, DeclarUI [23], LayoutCoder [24] and DCGen [25] use visual segmentation to map layouts to HTML code for one-shot generation from scratch. Other works like Interaction2Code [11] generate the interactive applications in iterative refinement with a focus on functionality testing and DesignRepair [26] and Nighthawk [27], [28] focus on the UI issues detection and repairing. Previous research does not address the continuous instruction-following needs of ongoing web editing.

III. PROBLEM FORMULATION

This work addresses the design editing task, where the inputs are a natural language instruction I describing a visual design change, and an existing HTML code file C_0 that renders a webpage W_0 , the goal is to produce a fully modified HTML code file that accurately reflects the intended edit. Given I and C_0 , a LLM M generates code $C_g = M(I, C_0)$ to render the modified webpage W_g . The rendered webpage W_g must

preserve instruction-unrelated portions of the original HTML while modifying only the relevant components, ensuring *semantic fidelity*, *visual correctness*, and *structural integrity*.

IV. METHODOLOGY

We present a two-stage framework for enabling instruction-grounded HTML rewriting: (1) a scalable, automated pipeline for synthesizing high-quality design-edit instructions paired with corresponding HTML modifications, and (2) fine-tuning large language models on the resulting dataset. The core insight lies in leveraging LLMs throughout the entire supervision loop — not only to simulate human-like design modification instructions and generate coherent code edits, but also to verify visual fidelity through cross-modal evaluation. This section outlines the full data synthesis pipeline and the subsequent model adaptation strategy.

A. Dataset Generation

1) Pipeline: We build **Instruct4Edit**, a clean and diverse dataset of instructional HTML edits, without involving human annotators. To avoid the high cost and scalability issues of manual data creation, we design a fully automated data generation pipeline that consists of three LLM-based components (shown in Fig 2): instruction generator, editor, and verifier. Technically, all three components can be implemented using the same underlying LLM.

Data Collection. We build our pipeline upon a set of real-world UIs. Specifically, we choose *WebCode2M* [2] dataset, which provides high-quality and real HTML/CSS code paired with rendered webpage screenshots. From this dataset, we randomly sample 500 examples to serve as seeds in this step.

Instruction Generation. Then, we generate natural language edit instructions that capture human design intent, simulating real-world modification requirements such as (e.g., “add spacing between sections”). To achieve this, we follow existing research [15] by supplying the LLM with a few carefully selected examples drawn from the *DesignEdit* dataset [3] that encourage diverse instruction generation. These examples provide diverse, human-authored design-edit pairs that help the model learn how to phrase visually grounded modification requests in natural language. Specifically, we provide the generator with original HTML and examples from prior real-world UI edits to avoid any mention of code or syntax. Each sample produces five diverse, human-like instructions.

HTML Editing. The next step is to apply each instruction to the original HTML in a coherent and complete way. Instead of generating differences, we request the editor to return a fully rewritten HTML document. This ensures the output remains renderable and self-contained.

Rendering & Verification. To verify the quality of the edits, we render both the original and modified HTML files in a fixed viewport setting for consistency. Then, these image pairs, along with the original instruction, are passed to the verifier that acts as a strict visual reviewer. The verifier component is implemented using an LLM configured as a visual-text reasoning agent. It then conducts “cross-modal verification”

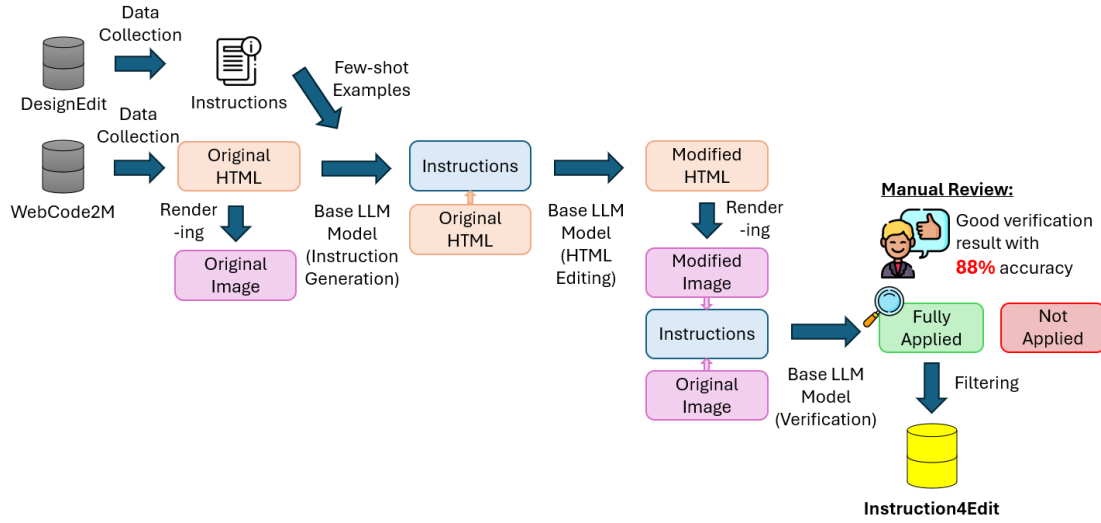


Fig. 2. End-to-end Pipeline to synthesize dataset with LLMs

by comparing visual changes with original instructions, and classifies the edit as fully applied, or not applied. Only samples where the verification step confirms visual alignment are included in the final dataset. Each retained example includes: a human-like instruction I , an original HTML document C_0 , and a modified HTML document C_M .

2) *Manual Quality Review*: While LLM verification accelerates filtering, it may introduce noise. So we conduct a manual inspection to validate the filtering quality. Two independent human reviewers cross-checked a random subset of 50 accepted samples by checking if the edit instruction was applied in the modified HTML content. This step confirms that the automatic verification yields **88%** agreement with human checking, indicating the high quality of our dataset. Inter-annotator reliability between the two reviewers was measured using Cohen’s Kappa, yielding $\kappa \approx 0.84$, which corresponds to “almost perfect agreement” according to the benchmark interpretation by Landis & Koch [29]. Disagreements were resolved through consensus discussion.

3) *Dataset Statistics*: We begin with 500 seed HTML files sampled from WebCode2M. For each, we generate 5 instructions, resulting in 2,500 instruction–HTML pairs. These pairs are processed through our editing and rendering pipeline, yielding 2,500 modified HTML candidates. After automated verification, 1,150 samples are accepted with an acceptance rate of **46%**. Most rejections are due to partial instruction alignment, visual rendering mismatches, or edits targeting hidden components (e.g., CSS-only visibility). We found these cases to be semantically ambiguous, suggesting opportunities for refining verifier precision in future iterations.

4) *Dataset Implementation Details*: To enhance reproducibility, we provide additional implementation details of *Instruct4Edit*.

Source corpus. We used *WebCode2M* [2], which contains approximately 2.56 million webpage code pairs with realistic layouts and styling.

Sampling. We randomly sampled 500 HTML/CSS examples from *WebCode2M* to serve as base templates for generating edits.

Instruction generation. For each base sample, we generated five natural-language editing instructions using the Gemini-2.5-Pro model, resulting in 2,500 instruction–HTML pairs. Prompts were explicitly constrained to avoid mentioning technical identifiers (e.g., class or ID names) to promote natural, human-like phrasing. Example instruction types include:

- *Layout edits*: “Make the layout more minimalist.”
- *Spacing edits*: “Increase padding between sections.”
- *Styling edits*: “Change the header font to something bold and modern.”
- *Color edits*: “Use a softer background color.”

Token statistics. The combined length of each instruction–HTML input sequence averaged approximately 2,800 tokens (median $\approx 2,500$; maximum $\approx 5,500$), motivating our choice of Qwen2.5-7B due to its extended context window (128,000 tokens).

Training setup. Fine-tuning was performed on a single NVIDIA A100 (80 GB) using LoRA adapters, with a batch size of 8 and peak memory usage around 60 GB. This setup ensures efficient training without gradient checkpointing or model parallelism.

Our pipeline ensures that all samples in *Instruct4Edit* are verifiable and consistent with human instructions, thus reducing noise and enhancing the robustness of the instruction-tuning process.

B. Tuning Models with Efficiency

Although pre-trained LLMs have strong generative abilities, they often struggle with precise, context-aware edits, especially for ambiguous or visually related instructions. Tuning on instruction-aligned examples helps models better align their output with user intent. To enable full-page HTML rewriting from high-level design instructions, we fine-tune LLMs using

(instruction, original HTML, modified HTML) triplets from the Instruct4Edit dataset.

For efficiency, we apply LoRA adapters [30] during tuning - a parameter-efficient technique that injects low-rank trainable weights into frozen pre-trained models- framing each task as single-shot text generation. The model input includes the original HTML and a natural language instruction, while the output is the fully modified HTML file.

V. EXPERIMENT

To explore the future of interactive web development via Instruct4Edit, we evaluate our approach from two perspectives: (RQ1) How effectively does Instruct4Edit support interactive edits? (RQ2) How many edit requirements are successfully applied, verified by humans? Our goal is to assess whether fine-tuning on Instruct4Edit improves instruction-grounded editing performance for any LLM, using Qwen2.5-7B as a representative open-source model.

A. Experiment Settings

1) *Evaluation Dataset*: For evaluation, we additionally sampled webpages from WebCode2M with the generated diverse instructions, ensuring these samples were excluded from the training data. Each edit request was manually implemented by an expert, and then verified by a second verifier to confirm that the changes were correctly applied to HTML.

2) *Baselines*: We choose the Qwen2.5 series [31] instead of alternatives like LLaMA2-7B [32] because it supports a larger maximum token limit. This is essential for our use case, where entire HTML documents serve as both input and output. Models with limited context windows, such as LLaMA-series, would require truncation, breaking the structural integrity. Specifically, we tune Qwen2.5-7B on the Instruct4Edit dataset, serving as our main approach, denoted as Qwen2.5-7B-Instruct.

Additionally, we include two open-sourced models (tuning-free) and two large commercial LLMs (inference-only) as baselines for comparison, to further underscore the contribution of Instruct4Edit: (1) Qwen2.5-7B: receives only the instruction and original HTML as input. (2) Qwen2.5-7B-VL (Multimodal): additionally incorporates a rendered screenshot of the original HTML. (3) Gemini-2.5-Pro: a proprietary multimodal model accessed via API, used here in a zero-shot setting. (4) GPT-4o-mini: a commercial instruction-tuned model with vision capabilities, also evaluated with zero-shot. All prompts are included in the replication package.

3) *Metrics*: Evaluation is conducted using both automatic metrics (RQ1) and human judgment (RQ2) to assess results. Specifically, we adopt two visual metrics following previous work:

- **Structural Similarity Index Measure (SSIM) [33]**: Captures perceptual differences between the original and modified renderings, focusing on layout consistency and low-level structural preservation. Given two grayscale images I_1 and I_2 , SSIM is defined as:

TABLE I
QUANTITATIVE EVALUATION OF EDITING RESULTS

Model	SSIM	CLIP
GPT-4o-mini	0.896	0.987
Gemini-2.5-Pro	0.883	0.979
Qwen2.5-7B-VL	0.764	0.960
Qwen2.5-7B-Base	0.796	0.975
Qwen2.5-7B-Instruct (Ours)	0.952	0.993

$$\text{SSIM}(I_1, I_2) = \frac{(2\mu_1\mu_2 + C_1)(2\sigma_{12} + C_2)}{(\mu_1^2 + \mu_2^2 + C_1)(\sigma_1^2 + \sigma_2^2 + C_2)}$$

where μ , σ^2 , and σ_{12} denote local means, variances, and covariances of the image patches, and C_1, C_2 are constants to stabilize the division. SSIM scores range from 0 to 1, with higher values indicating stronger structural similarity.

- **CLIP-Based Semantic Similarity [34]**: Evaluates high-level perceptual and semantic consistency using the CLIP ViT-B/32 model [34]. Let $f(I)$ be the CLIP embedding of image I , then cosine similarity is calculated as:

$$\text{CLIP}(I_1, I_2) = \frac{f(I_1) \cdot f(I_2)}{\|f(I_1)\| \|f(I_2)\|}$$

We normalize this to the $[0, 1]$ range by computing $(\text{cosine_sim} + 1)/2$.

B. Implementation

We implement our tuning pipeline with PyTorch and HuggingFace’s transformers [35], utilizing peft library to support LoRA. All experiments are conducted on a single NVIDIA A100 GPU with 80GB memory. Each model variant is trained for 3 epochs with a batch size of 8, a learning rate of $2e-5$, and a maximum sequence length of 8192 tokens to handle full HTML documents without truncation. Average inference time per edit is 1-2 minutes, enabling practical deployment. We use Gemini-2.5-Pro as the base LLM model for dataset generation.

C. Experiment results

1) *How effectively does Instruct4Edit support interactive edits?*: We report quantitative results with SSIM (for structural layout similarity) and CLIP (for semantic visual similarity) between the original and modified HTML renderings. Table I summarizes the average scores of our models across the evaluation, with Qwen2.5-7B-Instruct achieving the **highest visual similarity in both structural and semantic metrics**.

The gap between the tuned and basic (w/o tuned) Qwen2.5-7B model (0.952 vs. 0.796 SSIM, 0.993 vs. 0.975 CLIP) highlights the significance of Instruct4Edit. The model learns to apply semantically aligned edits that preserve the original layout unless instructed otherwise.

Surprisingly, the Qwen2.5-7B-VL model underperforms its textual counterpart, suggesting that **raw visual input might provide limited benefit for this task [3]**. Since HTML already encodes structure and semantics, adding screenshots

TABLE II
HUMAN EVALUATION RESULTS ACROSS MODELS.

Model	Passes	Fails	Passing Rate (%)
GPT-4o-mini	29	21	58
Gemini-2.5-pro	26	24	52
Qwen2.5-7B-VL	18	32	36
Qwen2.5-7B-Base	24	26	48
Qwen2.5-7B-Instruct (Ours)	28	22	56

may introduce redundancy. Therefore, we regard the vision-free approach as superior, both in effectiveness and efficiency, as it reduces memory usage and accelerates inference, making it better suited for practical real-world deployment in the future.

Despite their large model sizes, both GPT-4o-mini and Gemini underperform compared to our fine-tuned model. This highlights that, even with strong base models, **domain-specific fine-tuning can yield substantial improvements for structured generation tasks** such as UI code editing.

2) *How many edit requirements are successfully applied, verified by humans?*: To complement the automated evaluation, we conducted a manual evaluation on the same set of 50 design-edit samples, judging whether the modified HTML satisfied the instruction intent. Each output was independently labeled as a *pass* (instruction correctly applied) or *fail* (instruction not fully reflected or misapplied).

As shown in Table II, fine-tuning the base Qwen2.5-7B model with Instruct4Edit significantly **improves instruction-following ability**, increasing the pass rate from 48% to 56%. This confirms that even a relatively small, high-quality dataset like Instruct4Edit can enhance the model’s precision in UI-specific code editing.

The vision-enabled variant (Qwen2.5-7B-VL) underperforms its text-only version, with the lowest pass rate (36%) among all evaluated models. This aligns with our earlier observation that visual inputs may not introduce meaningful grounding for code-focused tasks, and could even distract generation.

While GPT-4o-mini achieves a slightly higher pass rate (58%) than our fine-tuned Qwen model (56%), its advantage is marginal. Our approach achieves **competitive results with a significantly smaller model size and open-source accessibility**, making it a more practical and customizable choice for real-world deployment in UI editing.

VI. CASE STUDY

To illustrate the practical performance of evaluated models, we present a case study where three baselines are tasked with the same instruction in Fig 3: *Make the logo smaller and position it on the left side of the header*. As shown in Figure 3, the base Qwen2.5-7B shifts unrelated components and misaligns the overall layout, while the Qwen2.5-7B-VL repeats the entire page multiple times vertically, failing to maintain the current design. In contrast, our fine-tuned Qwen2.5-7B-Instruct correctly resizes and repositions the logo, while preserving all other UI elements.

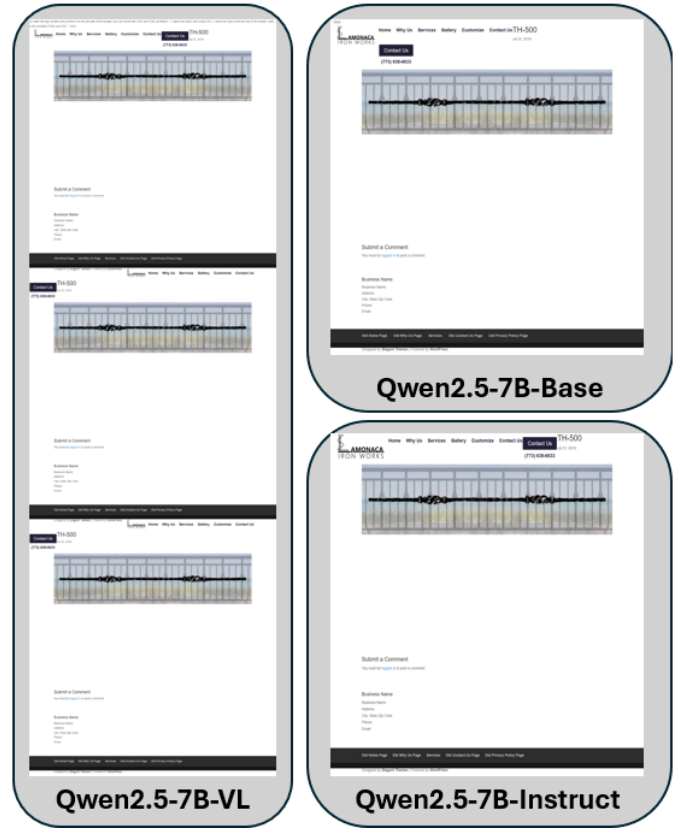


Fig. 3. Design edit outputs across model variants

VII. CONCLUSION AND FUTURE PLAN

This paper addresses a novel and practical task in continuous web development: automated editing of web HTML code to meet new requirements. To this end, we introduce Instruct4Edit, a transparent and automated dataset generation pipeline that constructs a high-quality instruction-tuning dataset to enable LLMs to better align with human expectations in web development. By fine-tuning open-source models on such dataset, we demonstrate significant improvements in layout preservation and edit satisfaction.

Looking forward, we plan to extend the pipeline for broader evaluation and exploit the LLM’s reasoning capabilities to enable more practical automated web editing. Future work includes expanding our framework to support diverse front-end frameworks, such as React and Vue, allowing instruction-driven edits in component-based frontends. Additionally, we aim to incorporate relevant UI programming knowledge through retrieval-augmented generation to further improve the LLM’s reasoning process.

ACKNOWLEDGEMENT

The work was supported by the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

REFERENCES

- [1] K. Moran, B. Li, C. Bernal-Cárdenas, D. Jelf, and D. Poshyvanyk, "Automated reporting of gui design violations for mobile apps," *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 165–175, 2018. [Online]. Available: <https://api.semanticscholar.org/CorpusID:3634687>
- [2] Y. Gui, Z. Li, Y. Wan, Y. Shi, H. Zhang, B. Chen, Y. Su, D. Chen, S. Wu, X. Zhou *et al.*, "Webcode2m: A real-world dataset for code generation from webpage designs," in *Proceedings of the ACM on Web Conference (WWW)*, 2025, pp. 1834–1845.
- [3] J. Xiao, M. Wang, M. H. Lam, Y. Wan, J. Liu, Y. Huo, and M. R. Lyu, "Designbench: A comprehensive benchmark for mllm-based front-end code generation," *arXiv preprint arXiv:2506.06251*, 2025.
- [4] W. Tang, J. Xiao, W. Jiang, X. Xiao, Y. Wang, X. Tang, Q. Li, Y. Ma, J. Liu, S. Tang *et al.*, "Slidecoder: Layout-aware rag-enhanced hierarchical slide generation from design," *arXiv preprint arXiv:2506.07964*, 2025.
- [5] J. Yang, C. E. Jimenez, A. L. Zhang, K. Lieret, J. Yang, X. Wu, O. Press, N. Muennighoff, G. Synnaeve, K. R. Narasimhan *et al.*, "Swe-bench multimodal: Do ai systems generalize to visual software domains?" in *Forty-second International Conference on Machine Learning (ICLR)*, 2025.
- [6] V. Liu, R. H. Kazi, L.-Y. Wei, M. Fisher, T. Langlois, S. Walker, and L. Chilton, "Logomotion: Visually-grounded code synthesis for creating and editing animation," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI)*, 2025, pp. 1–16.
- [7] Y. Lu, A. Leung, A. Swearngin, J. Nichols, and T. Barik, "Misty: Ui prototyping through interactive conceptual blending," in *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI)*, 2025, pp. 1–17.
- [8] K. Li, Y. Tian, Q. Hu, Z. Luo, Z. Huang, and J. Ma, "Mmcode: Benchmarking multimodal large language models for code generation with visually rich programming problems," in *Findings of the Association for Computational Linguistics (EMNLP 2024)*, 2024, pp. 736–783.
- [9] S. Zhang, Z. Xing, R. Guo, F. Xu, L. Chen, Z. Zhang, X. Zhang, Z. Feng, and Z. Zhuang, "Empowering agile-based generative software development through human-ai teamwork," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, Jan. 2025. [Online]. Available: <https://doi.org/10.1145/3702987>
- [10] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. Lyu, "Divide-and-conquer: Generating ui code from screenshots," *Proceedings of the ACM on Software Engineering*, vol. 2, no. FSE, pp. 2099–2122, 2025.
- [11] J. Xiao, Y. Wan, Y. Huo, Z. Xu, and M. R. Lyu, "Interaction2code: How far are we from automatic interactive webpage generation?" *arXiv preprint arXiv:2411.03292*, 2024.
- [12] J. Xiao, Z. Zhang, Y. Wan, Y. Huo, Y. Liu, and M. R. Lyu, "Efficientuicoder: Efficient mllm-based ui code generation via input and output token compression," *arXiv preprint arXiv:2509.12159*, 2025.
- [13] Y. Wan, T. Liang, J. Xu, J. Xiao, Y. Huo, and M. R. Lyu, "Automatically generating web applications from requirements via multi-agent test-driven development," *arXiv preprint arXiv:2509.25297*, 2025.
- [14] H. Laurençon, L. Tronchon, and V. Sanh, "Unlocking the conversion of web screenshots into html code with the websight dataset," *arXiv preprint arXiv:2403.09029*, 2024.
- [15] Y. Wei, F. Cassano, J. Liu, Y. Ding, N. Jain, Z. Mueller, H. de Vries, L. Von Werra, A. Guha, and L. Zhang, "Selfcodealign: Self-alignment for code generation," *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37, pp. 62 787–62 874, 2024.
- [16] S. Yun, H. Lin, R. Thushara, M. Q. Bhat, Y. Wang, Z. Jiang, M. Deng, J. Wang, T. Tao, J. Li, H. Li, P. Nakov, T. Baldwin, Z. Liu, E. P. Xing, X. Liang, and Z. Shen, "Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms," in *Advances in Neural Information Processing Systems (NeurIPS)*, vol. 37. Curran Associates, Inc., 2024, pp. 112 134–112 157.
- [17] C. Si, Y. Zhang, R. Li, Z. Yang, R. Liu, and D. Yang, "Design2code: Benchmarking multimodal code generation for automated 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 (NAACL)*. Albuquerque, New Mexico: Association for Computational Linguistics, Apr. 2025, pp. 3956–3974.
- [18] Y. Gui, Y. Wan, Z. Li, Z. Zhang, D. Chen, H. Zhang, Y. Su, B. Chen, X. Zhou, W. Jiang, and X. Zhang, "Uicopilot: Automating ui synthesis via hierarchical code generation from webpage designs," in *Proceedings of the ACM on Web Conference (WWW)*. New York, NY, USA: Association for Computing Machinery, 2025, p. 1846–1855.
- [19] K. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Transactions on Software Engineering (TSE)*, vol. 46, no. 2, pp. 196–221, 2018.
- [20] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with remaui (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 248–259.
- [21] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From ui design image to gui skeleton: a neural machine translator to bootstrap mobile gui implementation," in *Proceedings of the 40th International Conference on Software Engineering (ICSE)*, 2018, pp. 665–676.
- [22] Y. Gui, Z. Li, Y. Wan, Y. Shi, H. Zhang, Y. Su, S. Dong, X. Zhou, and W. Jiang, "Vision2ui: A real-world dataset with layout for code generation from ui designs," *arXiv preprint arXiv:2404.06369*, 2024.
- [23] T. Zhou, Y. Zhao, X. Hou, X. Sun, K. Chen, and H. Wang, "Bridging design and development with automated declarative ui code generation," vol. 1, no. FSE. ACM New York, Trondheim, Norway, 2025, pp. 1–24.
- [24] F. Wu, C. Gao, S. Li, X.-C. Wen, and Q. Liao, "Mllm-based ui2code automation guided by ui layout information," in *2025 International Symposium on Software Testing and Analysis (ISSTA)*. ACM New York, Trondheim, Norway, 2025, pp. 1–23.
- [25] Y. Wan, C. Wang, Y. Dong, W. Wang, S. Li, Y. Huo, and M. R. Lyu, "Divide-and-conquer: Generating ui code from screenshots," vol. 1, no. FSE. ACM New York, Trondheim, Norway, 2025, pp. 1–24.
- [26] M. Yuan, J. Chen, Z. Xing, A. Quigley, Y. Luo, T. Luo, G. Mohammadi, Q. Lu, and L. Zhu, "DesignRepair: Dual-Stream Design Guideline-Aware Frontend Repair with Large Language Models," in *IEEE/ACM 47th International Conference on Software Engineering (ICSE)*. Los Alamitos, CA, USA: IEEE Computer Society, May 2025, pp. 646–646.
- [27] Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "Nighthawk: Fully automated localizing ui display issues via visual understanding," *IEEE Transactions on Software Engineering (TSE)*, vol. 49, no. 1, pp. 403–418, 2022.
- [28] Z. Liu, C. Chen, J. Wang, Y. Huang, J. Hu, and Q. Wang, "Owl eyes: Spotting ui display issues via visual understanding," in *Proceedings of the 35th IEEE/ACM international conference on automated software engineering (ASE)*, 2020, pp. 398–409.
- [29] J. R. Landis and G. G. Koch, "The measurement of observer agreement for categorical data," *biometrics*, pp. 159–174, 1977.
- [30] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen *et al.*, "Lora: Low-rank adaptation of large language models," *ICLR*, vol. 1, no. 2, p. 3, 2022.
- [31] S. Bai, K. Chen, X. Liu, J. Wang, W. Ge, S. Song, K. Dang, P. Wang, S. Wang, J. Tang *et al.*, "Qwen2. 5-vl technical report," *arXiv preprint arXiv:2502.13923*, 2025.
- [32] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.
- [33] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [34] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *International conference on machine learning*. PMLR, 2021, pp. 8748–8763.
- [35] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, 2020, pp. 38–45.