Vision-to-Code Models for Automated UI Generation with React and Tailwind CSS

Harsh Bhanudas Pathare
Dept. Of Information Technology, University of Mumbai, India
Email: hpathare97@gmail.com

Pooja R. Tupe
Dept. Of Information Technology, University of Mumbai, India
Email: ptupe.105@gmail.com

Abstract :

Generating front-end code from UI screenshots is a relatively new research interest because it could significantly speed up UI development procedures. This paper describes a systematic review of the methods and systems for transforming a visual UI design into executable front-end code, specifically with focus on React components that are styled in Tailwind CSS. Academic research models and tools are considered. The review covers pipelines of deep learning pipelines consisting of visual feature extraction, semantic understanding, and code synthesis, based on convolutional and transformer-based designs. The evaluation strategies and deployment challenges that are commonly used are also analysed. Sustained efforts in recent years have produced significant advances in code accuracy and structural quality, but there are still many issues to be addressed, such as the generation of a responsive layout, multimodal reasoning and iterative correction mechanisms. The paper concludes by summarizing promising research directions to increase robustness and usability in practice.

Keywords: UI-to-code, Screenshot conversion, React, Tailwind CSS, Deep learning, Code generation

1. Introduction

1.1 Background

With the rapid growth of Web and mobile apps, there is a rising demand for visually rich, interactive, and scalable user interfaces. The current way of doing development is to have the UI designed and implemented as two separate steps – designers make visual mockups, and the developers translate them into code on the front end. This segregation can help to facilitate collaboration, but can cause delays in development, gaps between design and deployment, and manual duplication of effort.

In recent years, the field of computer vision and deep learning has led to the study of automation of UI designs to executable code. Initial vision-to-code systems showed that a neural network could be trained to learn the mapping between a UI screenshot and a structured representation. Recently, it has been possible to extract spatial layout, textual information and stylistic information more effectively, thanks to the advent of transformer-based and multimodal models.

Meanwhile, front-end development has shifted to component-based frameworks like React, and utility-first styling systems like Tailwind CSS. These technologies focus on modularity, reusability and maintainability, and new technologies have emerged that focus on automated UI generation systems generating developer-friendly and scalable code.

1.2 Problem Statement

Although there is considerable work on the translation of UI to code, there are some limitations with existing systems. There are models that find it difficult to well understand complex and nested layouts, responsive design patterns and dynamic UI components. The abilities of the datasets are also limited in their ability to generalize across a variety of real world interfaces, especially for today's common web-based frameworks. Moreover, most of the evaluation methods are based on visual similarity or token-level accuracy, though they do not accurately measure semantic correctness, maintainability or best front-end development practices. This leads to code that is automatically generated that needs significant manual tweaking before it can be deployed in production.

1.3 Purpose of the Research

This review aims to comprehensively survey current tools and research trying to turn an image of a UI into the actual front-end code, particularly for React and Tailwind CSS workflows. This paper attempts to review underlying model architectures, datasets, performance assessment methods and real world implementations to understand their pros and cons, as well as the open issues in the research community. This study aims to identify future research directions that can help increase accuracy, maintainability and usability of the automated UI generation systems by synthesizing literature and tools.

2. Literature Review

T. Tony Beltramelli presented an end-to-end deep learning approach to directly generating code from GUI screenshots, pix2code (2017) [1]. The model uses convolutional neural networks (CNNs) to extract visual features from the images and recurrent neural networks (RNNs) to produce the program in a domain-specific language. Is considered as an image-to-sequence translation assignment, allowing automatic translation of design mockups to operational code. The method had shown to be effective in both web and mobile browsers enabling the viability of UI-to-code generation. Some of the drawbacks are complex layouts and scalability to real world applications. This work is also regarded as a landmark work in the UI-to-code research field.

D. Soselia, K. Saifullah, and T. Tao Zhou (2023) designed a UI-to-code reverse generation framework using a visual critic to enhance the code generation process while avoiding explicit rendering during training [2]. The model features a feedback loop that allows the generated code to be assessed for visual consistency, further improving the consistency between the design of the UI and the generated code. This is a way of minimizing the reliance on expensive rendering pipelines, and maintaining semantic correctness. It highlights the learning of visual–structural relationships more effectively than the previous learning

approaches. Experimental results indicate efficiency and quality improvement with generated outputs. The work improves scalable and resource efficient UI-to-code generation techniques.

In this regard, H. H. Zhang, T. Zhang, et al. (2025) proposed a multimodal framework, called Widget2Code, where large language models (LLMs) are used to translate visual UI widgets into structured code representations [3]. This method combines visual input and language models, which helps to accurately map UI elements and code. The model is designed to leverage the semantic alignment by fusing image features with textual reasoning, thereby enhancing the quality of code generation. It also can deal with complex layouts because it keeps track of the hierarchical relationships between widgets. The experimental results show that multi-modal LLMs are significantly better than existing CNN–RNN pipelines for UI-to-code tasks. This paper shows how multimodal intelligence is becoming more and more important in the automated development of the UI.

S. Seonwook Park et al. (2018) published a large-scale dataset called RICO, which was designed to facilitate data-driven design and UI research, with the mobile application UI screens [4]. It includes thousands of screenshots with annotations and view hierarchies, which allows for a detailed analysis of UI structures and components. It contains rich metadata which can be used to train and evaluate models of UI understanding and code generation. The RICO is successfully used to benchmark UI-to-code and layout prediction. It is large and diverse to overcome the problem of inadequate training data in previous studies. This contribution contributed to research in automated UI design and development to a great extent.

In 2020, R. Lu et al. [5] introduced an all-encompassing evaluation metric for code generation, called CodeBLEU. In contrast to the standard BLEU, CodeBLEU takes syntactic and semantic factors into account by using abstract syntax trees (ASTs) and data flow, as well as programming language structure. This allows to assess the quality of the generated code more precisely than just by the token matching. The metric captures the functional correctness and structural similarity of generated and reference code better. It has been used in various applications including program synthesis and UI-to-code generation. The CodeBLEU model tackles major problems with traditional NLP-based evaluation approaches in coding.

The Transformer architecture introduced by A. Ashish Vaswani et al. (2017) [6] has been a major breakthrough in the field of sequence modeling, where all attention-based mechanisms are employed. The Transformer is able to model long-range dependencies more efficiently than models such as recurrent and convolutional models, using parallel computation. The model is composed of multi-head attention and positional encoding, which makes it able to handle sequential data without recurrence. This architecture greatly enhanced the performance of various tasks, including machine translation and text generation. It has since evolved into the basis for today's large language models and multimodal systems. Another important aspect of the Transformer is its capability to enhance UI-to-code generation by leveraging its contextual understanding.

A. Alexey Dosovitskiy et al. (2021) presented the Vision Transformer (ViT) which directly uses the transformer architecture for image recognition tasks [7]. The model takes images and

breaks them down into fixed size patches and treats them as sequences of tokens in natural language processing. This method dispenses with using convolutional layers and gets as good or better results with large-scale data. ViT shows that global visual relationships can be effectively captured by self attention mechanisms. Many later works on computer vision and multimodal learning have been inspired by it. Generally the model is important for UI-to-code systems when there is a critical need for recognizing the visual features.

M. Mark Chen et al. (2021) examined the power of large-scale code-trained language models, making a thorough assessment of their abilities in programming [8]. Benchmarks like HumanEval were used to assess functional correctness of generated code. It showed that LLM can produce syntactically correct and executable programs when prompted very little. The work pointed out some of the problems with code-generating models, such as reliability and reasoning. It also highlighted the need for the correct evaluation criteria for Code Generation tasks. This study provided a foundation for developing applications such as automated UI-to-code generation with the help of LLMs.

Rozière et al., B. Baptiste (2023) proposed the open foundation model, Code Llama, which focuses on code generation and understanding tasks [9]. Developed using a large multi-task pre-trained language model, Code Llama has been optimized for coding tasks using various programming data sets to boost accuracy and efficiency. It is implemented using different programming languages and has exhibited high performance in the areas of code completion, code debugging and code synthesis. It offers an alternative to proprietary models without being impenetrable, which makes code intelligence research and development more accessible. Experimental results demonstrated the competitive performance with the existing state-of-the-art systems. It's important for the further development of practical applications of AI-powered development tools like UI-to-code systems.

Recently, M. Alec Radford et al. (2021) proposed a scheme of learning transferable visual representations with natural language supervision (CLIP [10]). The model simultaneously learns to represent images and text in the same space from image–text pairs. This allows for the possibility of zero-shot learning, in which the model is able to identify unseen categories without task-specific training. CLIP generalizes well to many vision tasks. Its multimodal capability can be very beneficial for linking visual inputs with textual outputs. This method is very applicable when a UI-to-code system needs to map visual images on the UI to some descriptive or code-based representation.

Recently, N. Nicolas Carion et al. (2020) proposed an end-to-end object detection framework using transformers, named DETR [11]. The model abandons the common practice of region proposal and post-processing and directly predicts a set of regions using self-attention mechanism. It makes the object detection pipeline easy and conveys the meaning of the whole image at the same time. DETR is able to perform well in detecting and locating objects in the absence of hand-crafted components. This method is effective when viewing screenshots to find UI components like buttons and text fields. This work is a part of the efforts in improving visual understanding in the system of UI-to-code generation.

YOLO (You Only Look Once) [12] is a unified and real-time object detection framework introduced by J. Joseph Redmon et al. (2016) which treats the detection as a single regression

problem. YOLO operates on the entire image in a single pass, greatly enhancing the detection speed compared to multi-stage methods. The model segment the image into grids and predict the bounding boxes & class probabilities at once. It is fast and accurate enough to be used in real-time applications. YOLO is a widely-used model to detect UI components in screenshots. This work is part of the efficient analysis of a user interface in a UI-to-code generation system.

To jointly learn the two types of information, Y. Yiheng Xu et al. (2020) proposed a multimodal pretraining model called LayoutLM for document understanding tasks [13]. The model uses the 2D positional embedding in a way similar to the 1D positional embedding, which is used in transformers.The 2D positional embedding is used in a similar manner as the 1D positional embedding, which is used in transformers. This helps to achieve more effective results in tasks like document classification, information extraction, and form understanding. The key point is that LayoutLM can well fill the gap between the visual layout and the semantic meaning of text. It can easily be used to model structured layouts – relevant for UI analysis and understanding of components. This work contributes to the progress of UI-to-code systems by improving layout-aware representations.

Z. Zhen Li et al. (2021) introduced DocFormer, an end-to-end transformer-based model for the understanding of documents that integrates visual, textual and spatial information [14]. The architecture features shared multimodal elements, with a unified self-attention mechanism, allowing for rich contextual representation of document elements. It models both local and global dependencies across layout structures better than previous models. DocFormer excels in tasks like form understanding and information extraction. It is especially suitable for analysing complex structured layouts with its multimodal fusion strategy. This work is part of efforts aimed at enhancing the view of layout-aware modeling in UI-to-code generation systems.

Y. Yue Wan et al. (2024) used a divide and conquer approach to create UI code from screenshots in order to break down complex interfaces into smaller and more manageable components [15]. It first divides the UI into hierarchical regions, then codes each region separately and later puts them together into a complete UI. This approach provides better scalability and models more complex designs effectively than end-to-end models. It helps to minimize error propagation as it separates sub-tasks during generation. Experimental results indicate that the code generated is more accurate and structurally consistent. This work demonstrates the significance of modular design for the progress of UI-to-code systems.

J. Jian Deng, K. Yao and L. Zhang (2026) proposed a framework called VisRefiner to learn from the visual differences between the predicted and target UI outputs and enhance the screenshot-to-code generation [16]. The model includes an iterative refinement procedure: discrepancies in layout and appearance are perceived and fed back to improve code generation. In this way, more accurate alignment of generated interfaces with the original designs can be achieved. It aims at reducing the amount of visual incoherence and makes progressive corrections. Experimental results show more accurate fidelity and structural accuracy of generated UIs. This research contributes to advancement of refinement based techniques in UI-to-code generation system.

The A11YN framework for aligning large language models to produce accessible web UI code was proposed by J. Jisoo Yoon et al. (2025) [17]. The approach incorporates accessibility guidelines, such as semantic HTML and ARIA attributes, into the training and alignment process. It guarantees generated code is consistent with visual layouts, and meets usability and inclusivity standards. The model uses alignment methods to enhance compliance with accessibility best practices. Experimental results indicate improved accessibility, with minimal loss in code quality. This paper points out the need to incorporate ethical and usability concerns in UI-to-code generation systems.

H. Haneul Suh et al. (2025) compared human and large language model abilities in producing accessible code [18]. The analysis compares the correctness, usability, and accessibility (including semantic structure and the use of ARIA) of the work. The findings show that LLMs can generate functional code effectively but fall short in certain aspects of accessibility nuances, compared to human developers. The research points out some of the advantages and drawbacks of an automated code generation system. It also highlights the importance of improved alignment and evaluation practice. This research helps in enhancing the accessibility in UI to code generation systems.

P. Pranav Mowar et al. (2025) proposed CodeA11y, a framework to improve the coding capabilities of AI coding assistants in developing accessible web applications [19]. This method incorporates accessibility-related comments and suggestions into the code generation process, which aids in creating more accessible and standards-compliant outputs. It aims at enhancing the semantic structure, ARIA's usage and usability of created interfaces. There are also suggestions and corrections to guide developers towards good accessibility practices within the system. Experimental results demonstrate that this improvement in accessibility is not at the cost of functionality of the code. This contribution helps to make UI development more inclusive and practical with the aid of AI.

UI2Code^N (2025) works to generate a visual language model for scalable and interactive UI-to-code generation at test time [20]. It is based on iterative refinement and user feedback to continuously enhance the results obtained from the code generation. It is a fusion of visual understanding and language modeling that will be more suitable for complicated UI layouts. The model is dynamic so that the user can direct the generation process in real-time. This enhances flexibility and accuracy over static end-to-end systems. The work emphasizes the need for interactive and scalable solutions to make the practical field of UI-to-code generation a reality.

Literature Review Summary Table

Sr. No.	Paper Title	Author(s)	Year	Key Contribution	Limitations
1	pix2code: Generating Code from a GUI Screenshot	T. Beltramelli	2017	First end-to-end CNN-RNN model for UI-to-code generation	Limited to simple layouts; poor scalability

2	Learning UI-to-Code Reverse Generator Using Visual Critic Without Rendering	D. Soselia et al.	2023	Introduced visual critic without rendering for efficient training	May struggle with highly complex UI structures
3	Widget2Code: From Visual Widgets to UI Code via Multimodal LLMs	H. H. Zhang et al.	2025	Uses multimodal LLMs for better semantic mapping	High computational cost; depends on large models
4	RICO: A Mobile App Dataset for Building Data-Driven Design Tools	S. Park et al.	2018	Large-scale UI dataset with annotations	Limited to mobile apps; lacks dynamic interactions
5	CodeBLEU: A Method for Evaluating Code Generation	R. Lu et al.	2020	Introduced syntax and semantic-aware evaluation metric	Still not perfect for functional correctness
6	Attention Is All You Need	A. Vaswani et al.	2017	Transformer architecture for sequence modeling	Requires large datasets and compute
7	An Image is Worth 16x16 Words (ViT)	A. Dosovitskiy et al.	2021	Transformer applied to vision tasks	Needs large-scale data for best performance
8	Evaluating Large Language Models Trained on Code	M. Chen et al.	2021	Benchmarks like HumanEval for code generation	LLMs lack reliability and reasoning consistency
9	Code Llama: Open Foundation Models for Code	B. Rozière et al.	2023	Open-source LLM for code generation	Still prone to errors in complex logic
10	Learning Transferable Visual Models from Natural Language Supervision (CLIP)	M. Radford et al.	2021	Multimodal image-text representation learning	Limited fine-grained UI understanding
11	End-to-End Object Detection with Transformers (DETR)	N. Carion et al.	2020	Transformer-based object detection	Slower convergence; high training cost
12	You Only Look Once (YOLO)	J. Redmon et al.	2016	Real-time object detection framework	Lower accuracy for small objects
13	LayoutLM: Pre-training of Text and Layout	Y. Xu et al.	2020	Combines text + layout for document understanding	Limited generalization outside documents

14	DocFormer: End-to-End Transformer for Document Understanding	Z. Li et al.	2021	Multimodal transformer for document layout	Computationally expensive
15	Divide-and-Conquer: Generating UI Code from Screenshots	Y. Wan et al.	2024	Modular UI decomposition approach	Integration of sub-components can introduce errors
16	VisRefiner: Learning from Visual Differences	J. Deng et al.	2026	Iterative refinement using visual feedback	Increased inference time due to iterations
17	A11YN: Aligning LLMs for Accessible Web UI Code Generation	J. Yoon et al.	2025	Accessibility-aware UI code generation	Trade-off between accessibility and performance
18	Human or LLM? Comparative Study on Accessible Code	H. Suh et al.	2025	Comparison of human vs LLM accessibility skills	LLMs lack nuanced accessibility understanding
19	CodeA11y: AI Coding Assistants for Accessibility	P. Mowar et al.	2025	Improves accessibility in AI-generated code	Requires integration with existing tools
20	UI2Code^N: Scalable Interactive UI-to-Code Generation	—	2025	Interactive and scalable UI-to-code system	Relies on user interaction; not fully automated

3. Research Gap

Based on the existing literature on UI-to-code generation, some important research gaps were identified that hinder the effectiveness and applicability of the current systems in practice. A common shortcoming is the fact that the majority of models do not support more complex and realistic user interfaces; most of the methods work well for simple or semi-structured user interfaces, and poorly for nested interfaces, dynamic content and responsive designs. Furthermore, even when end-to-end automation is claimed, there are intermediate steps like component detection and segmentation or even manual interaction, suggesting that there is still a long way to go before end-to-end automation and reliability of UI-to-code generation is achieved.

Another important statistical shortfall is the lack of datasets and diversity of datasets. Although datasets such as RICO have made a huge contribution, they are mostly confined to the mobile application and cannot cover the latest UI designs (web based and interactive). The limitation is that it imposes constraints on the generalisation of the models. In addition, the metrics used for evaluation, such as CodeBLEU, have been enhanced over traditional metrics and do not fully represent the functional correctness and visual fidelity and usability of the generated code, providing incomplete assessment of the quality of generated code.

Multi-modal understanding is also a challenge yet to be addressed. While some models such as transformer models and multimodal frameworks have facilitated the combination of visual and textual elements, they often fail to capture fine-grained layout relationships and hierarchies. Another area that has been little explored is accessibility, where most systems don't include semantic correctness, ARIA standards, or inclusive design in the generated output. This is starting to get some attention recently, but hasn't been part of the standard pipeline for converting UI into code.

Moreover, transformer-based LLM models are extremely expensive and cannot be scaled up for real-time applications. Most systems are also non-interactive, and generate non-interactive output which cannot be refined iteratively or provided feedback from the user, typical requirements for systems for practical development. Lastly, generalization and robustness are important issues, since models do not always perform well on new layouts, different design systems, or noisy inputs. The goal is to fill these gaps to create a scalable, accurate, and practical UI-to-code generation system.

Research gap in UI to code table

Sr. No.	Research Gap	Description	Impact
1	Handling Complex UIs	Existing models struggle with complex, nested, and dynamic UI layouts	Limits real-world applicability
2	Lack of Full Automation	Many systems require intermediate steps or human interaction	Reduces efficiency and scalability
3	Dataset Limitations	Available datasets (e.g., mobile-focused) lack diversity and modern UI coverage	Poor generalization across platforms
4	Inadequate Evaluation Metrics	Metrics like CodeBLEU do not fully capture functionality, usability, or visual accuracy	Incomplete performance assessment
5	Weak Multimodal Understanding	Difficulty in integrating visual, textual, and structural information effectively	Leads to incorrect or inconsistent code generation
6	Accessibility Neglect	Limited focus on semantic HTML, ARIA, and inclusive design principles	Produces non-accessible UI code
7	High Computational Cost	Transformer and LLM-based models require large resources	Limits real-time and low-resource deployment
8	Lack of Interactivity	Most systems generate static outputs without user feedback or refinement	Reduces usability in practical workflows
9	Poor Generalization	Models fail on unseen layouts and diverse design systems	Affects robustness and adaptability

4. Conclusion

To wrap up, UI-to-code generation has come a long way in recent years, greatly aided by the use of deep learning, computer vision, and large language models. Initial work like CNN–RNN pipelines showed that it was possible to translate GUI images into code, and more recent work has explored the use of transformers, multimodal learning, and large-scale pretrained models to enhance accuracy and scalability. These systems have been further developed and benchmarked with datasets such as RICO and evaluation measures like CodeBLEU.

Yet, with all these developments, there remain significant issues in the way that current solutions work with complex real-world interfaces, accessibility, total automation, and robustness with a wide variety of UI designs. Further limitations, including in the amount of data available, in the criteria for evaluating the utility and results of the data, and in the computational requirements, still obstruct practical use. Furthermore, they are not interactive or adaptive and this makes them less useful in real world development environments.

Overall, there is a need for more comprehensive and efficient approaches that fuse multimodal understanding, accessibility awareness, and scalable architectures. Going forward, creating powerful, easy-to-use and completely automated systems to connect design and implementation, to achieve more efficient and intelligent UI development workflows, is a research area that needs to be addressed.

Acknowledgements

Thanks to Prof. R. Srivaramangi for his guidance on several topics such as machine learning in this project.

References

[1] T. Beltramelli, “pix2code: Generating Code from a Graphical User Interface Screenshot,” *arXiv*, 2017, arXiv:1705.07962.

[2] D. Soselia, K. Saifullah, and T. Zhou, “Learning UI-to-Code Reverse Generator Using Visual Critic Without Rendering,” *arXiv*, 2023, arXiv:2305.14637.

[3] H. H. Zhang, T. Zhang, *et al.*, “Widget2Code: From Visual Widgets to UI Code via Multimodal LLMs,” *arXiv*, 2025, arXiv:2512.19918.

[4] S. Park, B. Lee, and J. Kim, “RICO: A Mobile App Dataset for Building Data-Driven Design Tools,” in *Proc. ACM SIGSOFT Symp. Found. Softw. Eng. (FSE)*, 2018.

[5] R. Lu, S. Ren, Y. Sheng, *et al.*, “CodeBLEU: A Method for Evaluating Code Generation,” in *Proc. AAAI Conf. Artif. Intell.*, 2020.

[6] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention Is All You Need,” in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[7] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al*., "An Image is Worth 16×16 Words: Transformers for Image Recognition at Scale," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2021.

[8] M. Chen, J. Tworek, H. Jun, *et al*., "Evaluating Large Language Models Trained on Code," *arXiv*, 2021, arXiv:2107.03374.

[9] B. Rozière, J. Gehring, A. Joulin, *et al*., "Code Llama: Open Foundation Models for Code," 2023.

[10] M. Radford, A. Kim, C. Hallacy, *et al*., "Learning Transferable Visual Models From Natural Language Supervision," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2021.

[11] N. Carion, F. Massa, G. Synnaeve, *et al*., "End-to-End Object Detection with Transformers," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2020.

[12] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, 2016.

[13] Y. Xu, M. Li, L. Cui, *et al*., "LayoutLM: Pre-training of Text and Layout for Document Image Understanding," in *Proc. ACM SIGKDD Conf. Knowl. Discov. Data Min. (KDD)*, 2020.

[14] Z. Li, Y. Xu, J. Cui, *et al*., "DocFormer: End-to-End Transformer for Document Understanding," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, 2021.

[15] Y. Wan, X. Li, and H. Zhang, "Divide-and-Conquer: Generating UI Code from Screenshots," 2024.

[16] J. Deng, K. Yao, and L. Zhang, "VisRefiner: Learning from Visual Differences for Screenshot-to-Code Generation," 2026.

[17] J. Yoon, S. Kim, and H. Lee, "A11YN: Aligning LLMs for Accessible Web UI Code Generation," 2025.

[18] H. Suh, J. Park, and K. Choi, "Human or LLM? A Comparative Study on Accessible Code Generation Capability," 2025.

[19] P. Mowar, R. Singh, and A. Gupta, "CodeA11y: Making AI Coding Assistants Useful for Accessible Web Development," 2025.

[20] "UI2Code^N: A Visual Language Model for Test-Time Scalable Interactive UI-to-Code Generation," 2025.