MULTIMODAL GRAPH REPRESENTATION LEARNING FOR WEBSITE GENERATION BASED ON VISUAL SKETCH

Anonymous authors

ABSTRACT

The Design2Code problem, which involves converting digital designs into functional source code, is a significant challenge in software development due to its complexity and time-consuming nature. Traditional approaches often struggle with accurately interpreting the intricate visual details and structural relationships inherent in webpage designs, leading to limitations in automation and efficiency. In this paper, we propose a novel method that leverages multimodal graph representation learning to address these challenges. By integrating both visual and structural information from design sketches, our approach enhances the accuracy and robustness of code generation, particularly in producing semantically correct and structurally sound HTML code. We present a comprehensive evaluation of our method, demonstrating significant improvements in both accuracy and efficiency compared to existing techniques. Extensive evaluation demonstrates significant improvements of multimodal graph learning over existing techniques, highlighting the potential of our method to revolutionize design-to-code automation.

1 INTRODUCTION

The Design2Code problem, which involves converting digital designs into functional source code, is a pivotal challenge in software development that lies at the intersection of computer vision, natural language processing, and programming. This task is particularly demanding when generating HTML code from visual designs, as it requires not only the interpretation of visual elements but also an understanding of their spatial arrangements and hierarchical relationships. While significant advances have been made in both vision-language models and code generation models, combining the strengths of these two fields to solve Design2Code remains an unsolved challenge. One of the fundamental difficulties is that current vision-language models, which excel at generating descriptive text from images (Hakimov & Schlangen, 2023; Iscen et al., 2024; Lin et al., 2024), often fall short when tasked with understanding the intricate structural layout of web designs. These models, which typically focus on high-level image features, struggle with capturing the precise relationships between nested elements, such as divs or dynamic components. On the other hand, code generation models (Zan et al., 2023; Sun et al., 2024; Lyu et al., 2024), despite their proficiency in generating syntactically correct code from textual descriptions, often lack the visual comprehension needed to map these descriptions to complex layouts or design sketches. This results in code that may be syntactically correct but fails to accurately represent the intended design.

Furthermore, a core issue complicating the Design2Code process is the challenge of capturing and reproducing the complex hierarchical structures inherent in modern webpages. Webpages are often built using deeply nested blocks, grid layouts, and diverse visual components, each of which requires precise spatial alignment and accurate rendering in code. This structural complexity is compounded by the need to accurately reflect the visual relationships between components—such as positioning, layering, and grouping—which are often difficult for automated models to capture. It is further compounded when considering real-world design practices, which often involve incomplete or ambiguous specifications that make automated code generation even more prone to errors. The key to overcoming these challenges lies in developing a model architecture that not only captures the global features of an image but also pays close attention to these critical attributes, ensuring both breadth and depth in its analysis.

Given these motivations, we introduce a novel framework for the Design2Code problem that integrates multimodal information from webpage screenshots and component-level relationships. Using

Optical Character Recognition (OCR) and segmentation models, our approach extracts textual and visual elements, organizing them into a multimodal graph that captures the webpage's structure and serves as a blueprint for generating HTML code. By isolating text components early through OCR, we enable the segmentation model to more accurately focus on visual elements without interference from overlapping text. The multimodal graph maps both textual and visual components, capturing their spatial and semantic relationships. Textual elements are linked semantically, while visual components are connected based on spatial proximity, forming the foundation for code generation. Our model integrates the visual layout and structural relationships within a graph representation, using a Vision-Language Model (VLM) to encode the webpage's appearance. This combined multimodal data ensures both visual and structural details contribute to generating accurate HTML code. The contributions of this paper are threefold:

- We introduce a novel OCR-Segmentation pipeline for extracting webpage components, which enhances the segmentation of non-textual elements by masking noisy textual content.
- We propose a graph-enhanced Vision-Language Model that integrates graph-based representations of webpage structure with vision and language features to generate accurate HTML code.
- We perform extensive experiments to demonstrate that our approach significantly improves both content accuracy and layout fidelity compared to baseline models, offering a robust solution to the Design2Code problem.

In the following sections, we first explore the existing research on the problem of converting sketches into website source code in Section 2. Section 3 then details the methodology employed in this study. The experimental results are discussed in Section 4, followed by conclusions, limitations, and future research in Section 5 and Section 6, respectively.

2 Related work

2.1 Design2Code

Approaches to Automating Design2Code Generation. The Design2Code (or UI2Code) problem has gained significant traction as researchers work to automate the conversion of user interface (UI) designs into source code. Early methods primarily relied on rule-based systems that assigned UI components to predefined code templates (Nguyen & Csallner, 2015), offering limited flexibility. As the complexity of web design increased, newer approaches adopted deep learning techniques for enhanced automation and accuracy. Notable models like Pix2Code (Beltramelli, 2018) and Sketch2Code (Robinson, 2019) utilize convolutional neural networks (CNNs) to transform UI screenshots into single-file code, with Pix2Code demonstrating cross-platform versatility and Sketch2Code focusing on wireframe sketches. Recent efforts, such as those by Soselia et al. (2023), treat image-to-code generation as a Reinforcement Learning problem, using the Intersection over Union (IoU) score as a reward signal. Evaluations on benchmarks like Design2Code Si et al. (2024) have employed large language models (LLMs) such as GPT-4V and Gemini Pro Vision, achieving impressive results without fine-tuning. However, the high computational costs of these models limit their practical use in real-time development. While other models like Web2Code baselines Yun et al. (2024) integrate LLMs and CLIP for local inference, they often lack precision due to omitted structural features. This work enhances Visual-Language Models (VLMs) with suitable size by incorporating graph learning to better capture structural features, thereby improving precision in the Design2Code task.

Datasets for Design2Code. A crucial factor in advancing Design2Code methods is the availability of suitable datasets and benchmarks. One foundational dataset in this field is Pix2Code, an open-source, synthesized collection of 1,750 UI screenshots paired with corresponding source code. The Sketch2Code dataset (ShantamVijayputra, 2022) builds on this by converting Pix2Code screenshots into hand-drawn wireframe representations, adding an additional layer of abstraction for model training. Additionally, the Websight dataset from Hugging Face (Laurençon et al., 2024) offers a larger scale, comprising 2 million triplets of HTML code, screenshots, and generated descriptions. While these synthesized datasets are critical for enabling large-scale training, they often lack the diversity and complexity found in real-world web pages. Similarly, Vision2UI Gui et al. (2024)

provides 20,000 samples extracted from real-world scenarios through a meticulous process of data collection, cleaning, and filtering. Although these datasets advance the field by introducing more complex and diverse data, the WebUI dataset (Wu et al., 2023), although comprehensive and rich in metadata, lacks the corresponding source code and is therefore not used in Design2Code tasks. To bridge the gap between synthesized datasets and real-world applications, the Design2Code benchmark (Si et al., 2024) was introduced as the first dataset of real-world web pages specifically for evaluating design-to-code models. Our work leverages the available WebSight datasets to train and benchmark the effectiveness of graph learning in the Design2Code task.

Existing Benchmarks. htmlBLEU (Soselia et al., 2023) uses DOM-tree matching between text inputs and enhances this method by incorporating attribute matching and assigning additional weights to HTML keywords in the BLEU calculation. Similarly, (Gui et al., 2024) introduces TreeBLEU, an improvement over BLEU, which evaluates the match between generated HTML by comparing 1-height subtrees of the DOM from both the hypothesis and reference structures, enabling a more accurate comparison. Web2Code (Yun et al., 2024) proposes metrics based on large language models (LLMs) to evaluate webpage understanding and code generation; however, resource constraints limit the real-time application of this method in development. Other vision-related metrics in Soselia et al. (2023) have used simpler metrics such as Mean Squared Error (MSE) or Structural Similarity Index (SSIM) based on pixel values. Our work utilizes various existing evaluation metrics, excluding VLM prompting methods, to provide a comprehensive assessment of model performance in the Design2Code domain, building on previous approaches that integrated both VLMs and LLMs for evaluation.

2.2 From Mutlimodal Input To Text With Vision Language Models

Vision-Language Models (VLMs) have transformed how machines interpret and generate language from visual inputs, enabling a range of multimodal tasks. Early models like VisualBERT (Li et al., 2019) and LXMERT (Tan & Bansal, 2019) laid the groundwork by combining BERT (Devlin et al., 2019) with visual encoders, using a dual-stream approach that limited deep fusion of visual and textual information. Later models, such as UNITER (Chen et al., 2020) and VilBERT (Li et al., 2019), achieved tighter integration and better cross-modal interactions but struggled with tasks requiring complex reasoning, like code generation. CLIP (Radford et al., 2021) marked a shift by aligning visual and textual embeddings through contrastive learning, while GPT-4V (OpenAI, 2023) and LLaVA (Liu et al., 2023) extended multimodal understanding by generating unimodal (text) outputs from multimodal inputs. PALM-E (Driess et al., 2023) further pushed VLMs into real-world tasks like robotic manipulation. DeepMind's Flamingo (Alayrac et al., 2022) excels at processing interleaved visual and textual data, performing well in few-shot learning tasks. Its open-source counterpart, OpenFlamingo (Awadalla et al., 2023), and models like BLIP (Li et al., 2022), ALBEF (Li et al., 2021), Idefics (Laurençon et al., 2023), and CogVLM (Wang et al., 2024) enhance feature alignment, improving text generation from visual inputs. Our work builds on Flamingo and OpenFlamingo by incorporating Graph-Enhanced Learning to better understand the structural aspects of language (code) and images (design).

2.3 Graph Neural Networks

Graph Neural Networks (GNNs) have gained significant attention in recent years due to their ability to extend deep learning to non-Euclidean domains, specifically graphs. By effectively modeling relationships between entities in structured data, GNNs have become a crucial tool for solving problems in fields such as social network analysis (Perozzi et al., 2014; Fan et al., 2019; Bonifazi et al., 2024), knowledge graphs (Kipf & Welling, 2017; Barceló et al., 2020; Cucala et al., 2022), molecular modeling (Reiser et al., 2022; Xia et al., 2023; An et al., 2024), and more. GNNs are uniquely designed to operate on graph-structured data, where nodes represent entities and edges define relationships between them. This makes GNNs highly adaptable, as they can capture both local node information and the structural dependencies within the graph. Among the most prominent variants of GNNs (Kipf & Welling, 2017; Liu et al., 2019; Ruiz et al., 2020; Rampášek et al., 2022), Graph Convolutional Networks (GCNs) can still stand out due to their ability to generalize convolution to graph-structured data, leveraging the connectivity of nodes to aggregate information from their neighbors. By utilizing a recursive neighborhood aggregation process, often referred to as "message

passing," GCNs enable each node to update its representation by integrating its own features with those of its neighbors. This allows the model to capture both local node information and the broader structural context in a scalable and efficient manner. Their simplicity, efficiency, and strong theoretical foundation make GCNs highly adaptable to a wide range of graph-based tasks. Formally, in GCNs, the forward pass for each layer can be describe as:

$$H^{(l+1)} = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}).$$

Here, $\tilde{A} = A + I_N$ denotes the adjacency matrix of the undirected graph $\mathcal{G}$ with inserted self-loops, I_N is the identity matrix, and $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ is its diagonal degree matrix. $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes a non-linear activation function such as $\mathbf{ReLU}(\cdot)$. $H^{(l)} \in \mathbb{R}^{N\times D}$ is the matrix of activation in the l^{th} layer, with $H^{(0)} = X$ is the feature matrix. In GCNs, the "convolution" is applied to nodes in a graph, where the neighbors of each node are analogous to the local region in a CNN. The key idea is that each node aggregates information from its neighbors based on the graph's connectivity. Instead of spatially local filters, GCNs use the graph adjacency matrix to propagate and aggregate features from a node's neighborhood. This process updates each node's representation by combining its features with those of its neighbors in a way similar to how convolution combines information from nearby pixels in CNNs. Its node-wise formulation is given by:

$$x_i^{'} = \theta^T \sum_{j\in\mathcal{N}(i)\bigcup\{i\}} \frac{e_{j,i}}{\sqrt{\hat{d}_j\hat{d}_i}} x_j,$$

in which $\hat{d}_i = 1 + \sum_{j\in\mathcal{N}(i)} e_{j,i}$ denotes in-degree of node i and $e_{j,i}$ denotes the edge weight from source node j to target node i. By drawing on these related works, our approach integrates the strengths of vision-language models, code generation techniques, and graph representation learning.

3 METHODOLOGY

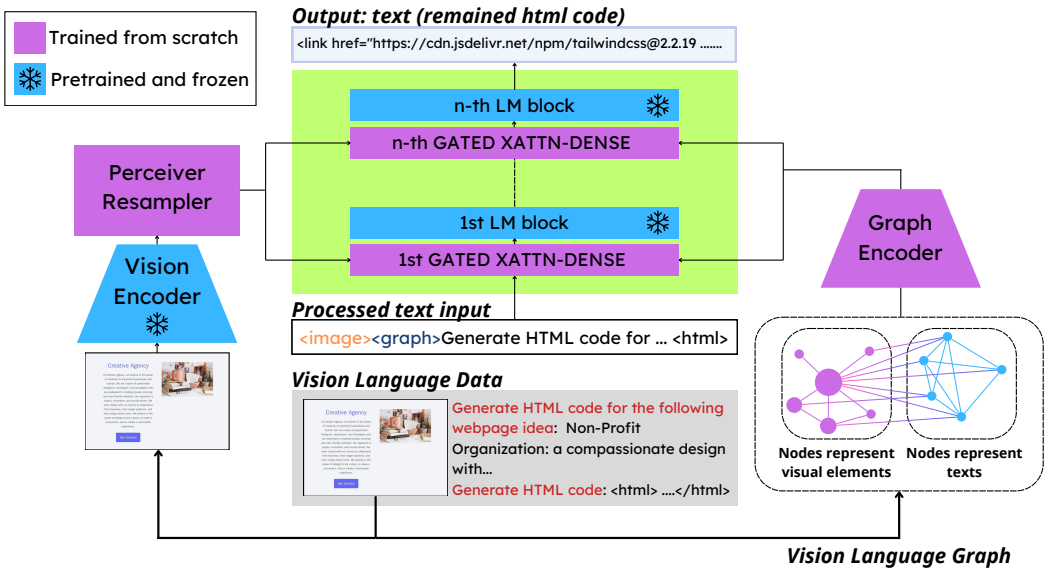


Figure 1: Overview of the Graph-enhanced Multimodal Architecture for Generating HTML Code from Visual Sketches.

In this work, we propose a novel framework to address the Design2Code problem, which aims to automatically generate webpage's code from visual designs. Our approach leverages both an Optical Character Recognition (OCR) model and a Segmentation model to identify and extract the individual components from a given webpage screenshot. These extracted components, along with their spatial positions, are then used to construct a graph representing the webpage structure. To further enhance

the quality of code generation, we introduce a graph-enhanced vision language model that integrates visual and structural information to produce both the content and the corresponding HTML code for the webpage.

(a) The original webpage screenshot

(b) Text-masked image by PaddleOCR

(c) Element-segmented image by SAM

Figure 2: Illustration of Component Extraction process.

Webpage screenshots often present significant challenges due to the complexity and variety of their content. These images typically contain noisy visual elements, such as overlapping text, dense layouts, and intricate designs, which make it difficult for traditional segmentation models to accurately identify and isolate individual components. To address this issue, we utilize a two-step approach combining Optical Character Recognition (OCR) and the Segment Anything Model (SAM) [1].

Text Extraction Using OCR. We begin by applying an OCR model to detect and extract all textual content from the webpage. The OCR model effectively identifies text components, such as headers, paragraphs, and embedded labels, which are often misinterpreted or missed by a segmentation model working directly on the raw screenshot. Once the OCR has successfully extracted these text elements, we remove them from the image and replace them with black blocks. This step reduces the visual complexity of the image, allowing the segmentation model to focus on non-textual elements.

Segmentation of Non-Textual Components. With the text removed and masked by black blocks, we apply the SAM to the modified screenshot to extract the remaining components, such as images, buttons, and containers. The black blocks act as placeholders, guiding the segmentation model to recognize distinct visual elements without being confused by text. This process ensures that the non-text components are segmented more accurately and that their relationships with the surrounding elements are better preserved.

Final Components Combination. After segmentation, we merge the OCR-extracted textual content with the corresponding visual components from the original webpage. The final set of components includes both the textual content identified by the PaddleOCR [2] model and the rectangular blocks representing the elements extracted by the segmentation model. These components are then used to construct a multimodal graph that represents the content, structure and layout of the webpage. By separating the handling of text and non-text elements, our approach significantly improves the precision of component extraction and ensures that both types of content are correctly interpreted for the subsequent stages of the framework.

3.1 Graph Construction

To capture the structural and semantic relationships between components, we construct a multimodal graph in which each node corresponds to a component extracted in the previous step. These components include both textual contents (or textual components) and visual blocks (or visual components). The graph structure is designed to encode both content-based and spatial relationships between elements, allowing the model to efficiently process and generate the webpage's layout and content.

[1] https://github.com/facebookresearch/segment-anything

[2] https://github.com/PaddlePaddle/PaddleOCR

Node Representation. Each node in the graph represents a specific component extracted from the webpage screenshot. Textual components, such as headers, paragraphs, and labels, are derived from the OCR process, while visual components, including images, buttons, and containers, are extracted via the segmentation model. By distinguishing between textual and visual nodes, the graph captures both the content and the spatial characteristics of the webpage.

Edge Construction. Edges in the graph are designed to encode meaningful relationships between components, ensuring the graph reflects both the logical content structure and the spatial layout of the webpage. We define edges as follows:

- **Textual Component Edges**: All nodes representing textual components are fully connected. This means that each pair of textual nodes has an edge between them, forming a complete subgraph. The rationale for this design is that textual components often contribute to the overall coherence of the webpage content, making it essential for these nodes to share information. By fully connecting textual components, we allow the model to better understand the relationships between different text elements, enhancing the overall representation.
- **Visual Component Edges**: Edges between visual components are constructed based on their spatial relationships. Specifically, if the intersection area between two visual components exceeds 80% of their combined coverage area, an edge is created between them. This threshold ensures that components that are physically close or overlapping in the layout are connected, facilitating the exchange of spatial information between these nodes. The goal is to capture the spatial dependencies between visual elements, such as the relationships between images, buttons, and other design elements.
- **Textual-Visual Edges**: Edges between textual and visual components are constructed using the same spatial criterion. If the intersection area between a textual component (represented by the rectangular block detected by the OCR model) and a visual component exceeds the 80% threshold, an edge is added between these two nodes. This design enables the model to capture interactions between textual content and nearby visual elements, such as text labels associated with buttons or images, ensuring that both content and layout are appropriately represented.

3.2 GRAPH-ENHANCE VISION LANGUAGE MODEL

Our framework leverages a graph-enhanced vision-language model to generate accurate webpage content and HTML code by conditioning a language model on both visual and graph-based information. This architecture combines the strengths of graph neural networks (GNNs), vision encoders, and pretrained language models, creating a multimodal system that captures the structure, layout, and content of the webpage.

Graph Convolutional Network for Graph Embeddings. We employ a graph convolutional network (GCN) to encode the multimodal graph constructed from the webpage components. The GNN serves as the graph encoder, learning informative embeddings for each node in the graph. At the Layer 0 of the GNN, we initialize the node embeddings using the features extracted from each textual and visual component. For the feature extraction, we utilize CLIP, a powerful multimodal vision-language model. CLIP allows us to obtain rich, contextualized embeddings for each component, combining both visual features (for visual components) and textual embeddings (for textual components). These embeddings are passed into the GNN, where subsequent layers refine the node embeddings by considering the graph's structure and relationships between components. The GCN effectively learns higher-level representations that capture the relationships between different webpage components, ensuring that both semantic relevance (from textual components) and spatial relationships (from visual components) are encoded.

Vision Encoder with Perceiver Resampler. Vision Encoder with Perceiver Resampler To encode the visual information from the entire webpage screenshot, we draw inspiration from the Flamingo model and incorporate a Vision Encoder alongside Perceiver Resampler layers. The Vision Encoder processes the full webpage image and extracts high-level visual features that provide contextual understanding of the page layout, color schemes, and design elements. The Perceiver Resampler layers

are applied to reduce the dimensionality of the visual features, condensing the extensive information present in a high-resolution webpage screenshot into a more compact and computationally efficient form. This allows our model to focus on the most relevant visual aspects of the webpage, which is critical for generating the HTML code in alignment with the design's visual structure.

Cross-Attention for Mutimodal Conditioning. The crux of our model lies in how we combine the information from the graph and vision encoders to condition the language model for text and code generation. We use cross-attention layers, which are interleaved between the pretrained layers of a standard language model, to fuse the multimodal inputs into the token prediction process. Specifically, freshly initialized cross-attention layers are introduced between the existing layers of the pretrained language model. These layers take the embeddings from both the vision encoder and the graph encoder, allowing the language model to incorporate multimodal information at every step of the token generation process. This approach is more aggressive than simple concatenation or late fusion, as it enables a deeper integration of visual and graph information throughout the entire language modeling process. The process is formalized as follows:

$$\mathbf{E}_{\mathrm{L}}^{i} = \mathrm{GCA}(\mathbf{X}, \mathbf{E}_{\mathrm{L}}^{i-1}) + \mathrm{GCA}(\mathbf{Z}, \mathbf{E}_{\mathrm{L}}^{i-1}) + \mathbf{E}_{\mathrm{L}}^{i-1},$$

where $\mathbf{X}$ represents the vision embedding, $\mathbf{Z}$ represents the graph embedding, and $\mathbf{E}_{\mathrm{L}}^{i}$ is the output at the i^{th} model layers, which integrates both graph and visual information into the textual embedding. The operator $\mathrm{GCA}(\mathbf{A}, \mathbf{B})$ denotes the Gated-Cross Attention mechanism, which performs attention between two embeddings $\mathbf{A}$ and $\mathbf{B}$ from different modalities.

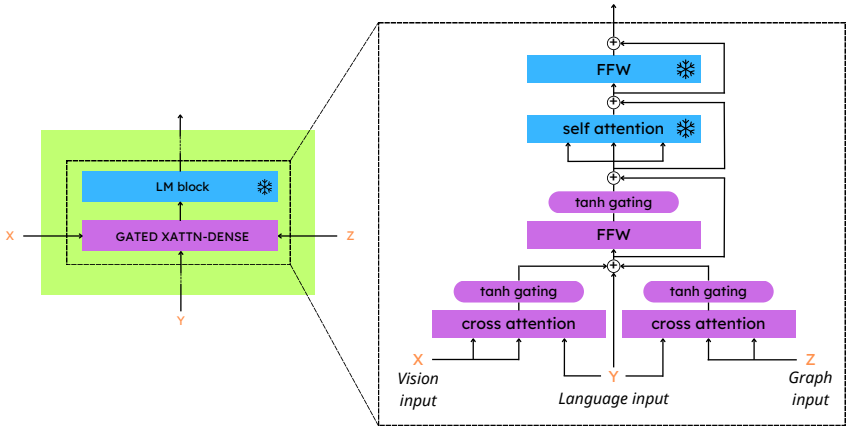


Figure 3: Gated Cross Attention Block. The Gated Cross Attention Block integrates vision (X), language (Y), and graph (Z) inputs through cross-attention layers, followed by tanh gating to control information flow. Each input is processed through its respective cross-attention mechanism, and the outputs are combined

The cross-attention mechanism ensures that the model can attend to both the layout and the relationships between webpage components when generating the next token, whether that token is part of the webpage content or the HTML code. By doing so, the model not only generates accurate content but also respects the spatial and semantic coherence of the original webpage design.

4 Experiment

4.1 Experimental Setup

For datasets, we used both real-world and synthetic data sources. Real-world datasets like Design2Code and Vision2UI provide authentic UI examples but pose challenges such as instability during training and complex structures. Due to these issues and limited computing resources, we focused on synthetic data. We trained our model on a curated subset of the Websight Hugging Face dataset, which contains 2 million HTML code and screenshot pairs. This subset was chosen for its

	WebSight v0.1	WebSight v0.2	Design2Code	WebSight benchmark
Size	823K	1.92M	484	500
Purpose	Training	Training	Testing	Testing
Avg Length (tokens)	647±216	708±265	31216±23902	723±271
Avg Tag Count	19±8	19±7	158±100	20±7
Avg DOM Depth	5±1	6±1	13±5	6±1
Avg Unique Tags	10±3	11±3	22±6	11±3

Table 1: Comparison of dataset statistics between WebSight v0.1, WebSight v0.2, Design2Code benchmark, and collected samples from WebSight v0.2 as benchmark.

diversity and manageable size, avoiding the noise and instability of real-world data. Another subset of Websight Hugging Face was used as a benchmark for comparison.

Table 1 presents a comparison of dataset statistics between WebSight v0.1, WebSight v0.2, Design2Code, and a WebSight benchmark subset. The data highlights key metrics such as average length, tag count, DOM depth, and unique tags. The WebSight benchmark subset (500 samples) shows average values for length (723±271 tokens), DOM depth (6±1), and tag count (20±7), which closely resemble the overall characteristics of WebSight v0.2. This alignment suggests that the selected WebSight benchmark subset appropriately represents the complexity of the full WebSight dataset, making it a suitable choice for evaluating models alongside datasets like Design2Code.

In terms of evaluation, we employed several metrics to assess performance. These include Block-match, which evaluates the spatial alignment of design blocks; Text, focusing on textual accuracy; and Position, Color, and CLIP, which assess visual and layout fidelity. Additionally, we considered traditional metrics such as MSE (Mean Squared Error) and SSIM (Structural Similarity Index) for image quality, as well as more specialized metrics like TreeBLEU and htmlBLEU for assessing code similarity and structure.

4.2 Quantitative Results

	Block-Match ↑	Text ↑	Position ↑	Color ↑	CLIP ↑
Websight HF benchmark					
OURS-graph	24.94	79.33	70.52	75.41	**91.36**
OURS-no-graph	21.60	76.06	66.21	69.70	89.63
Gemini-prompting	**98.21**	**99.47**	**78.29**	**83.83**	89.85
Design2Code benchmark					
OURS-graph	2.63	47.15	37.97	40.07	82.63
OURS-no-graph	2.51	50.15	37.37	43.30	82.90
Gemini-prompting	**90.64**	**95.92**	**78.06**	**73.24**	**88.37**

Table 2: Performance Comparison on Websight HF and Design2Code Benchmarks. This table presents the performance of OURS-graph, OURS-no-graph, and Gemini prompting on various metrics for Design2Code tasks across two benchmarks: Websight HF and Design2Code. The metrics evaluated include Block-Match, Text, Position, Color, and CLIP. Gemini prompting demonstrates superior performance across most metrics. OURS-graph shows better performance than OURS-no-graph, especially in visual metrics like Block-Match and Position, reflecting the advantage of incorporating graph-based modeling for structural alignment and visual fidelity.

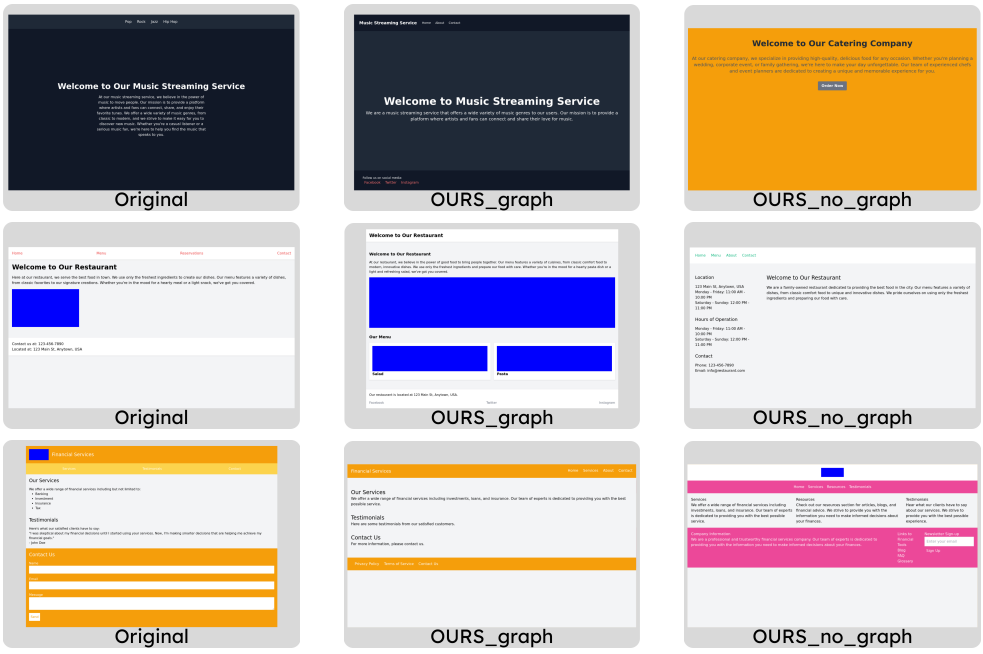
4.3 Qualitative Results

The performance comparison between OURS-graph, OURS-no-graph, and Gemini prompting demonstrates the clear advantage of using graph-based representations. OURS-graph consistently achieves better structural harmony across webpage components, capturing spatial relationships more

	BLEU ↑	HTML-BLEU ↑	MSE ↓	SSIM ↑	TreeBLEU ↑
Websight HF benchmark					
OURS-graph	**43.83**	**52.55**	**35.91**	**81.16**	**44.29**
OURS-no-graph	39.69	48.19	43.20	77.57	37.21
Gemini-prompting	41.37	34.87	63.60	76.58	15.52
LLAVA-CrystalChat-7B	12.81	21.24	N/A	N/A	7.69
Design2Code benchmark					
OURS-graph	2.49	7.15	67.67	65.82	16.59
OURS-no-graph	2.03	**7.27**	75.69	65.14	16.71
Gemini-prompting	**6.45**	7.17	**48.55**	**69.72**	**24.83**
LLAVA-CrystalChat-7B	N/A	N/A	N/A	N/A	N/A

Table 3: Performance on Traditional Metrics for Websight HF and Design2Code Benchmarks. This table compares OURS-graph, OURS-no-graph, Gemini prompting, and LLAVA-CrystalChat-7B using traditional metrics such as BLEU, HTML-BLEU, MSE, SSIM, and TreeBLEU. On the Websight HF benchmark, OURS-graph outperforms both baselines across all metrics, indicating its effectiveness in generating accurate and visually coherent HTML code. On the Design2Code benchmark, OURS-graph maintains superiority over OURS-no-graph, further demonstrating the value of graph-based representation in improving structural and visual fidelity, although Gemini prompting performs better on some metrics like BLEU and MSE.

Table 4: Qualitative comparison between graph and w/o graph methods



accurately and delivering modern, well-aligned designs. It also excels in visual aspects, such as coloring and style. While OURS-no-graph generates functional HTML, it lacks the precision in layout and visual consistency. Gemini prompting performs well in capturing both content and structure but occasionally falls short in maintaining accurate coloring and style. Overall, OURS-graph proves superior in balancing both structure and visual fidelity.

5 CONCLUSION

In conclusion, we present a novel approach to solving the Design2Code problem by integrating multimodal information through a graph-based model that captures the spatial and semantic relationships within webpage designs. Our OCR-Segmentation pipeline effectively isolates textual content, allowing for more accurate segmentation of visual elements. The proposed graph-enhanced vision-language model bridges the gap between visual comprehension and code generation, resulting in improved HTML generation in terms of both layout fidelity and content accuracy. Our extensive experiments validate the effectiveness of this approach, offering a significant advancement toward automated code generation from web designs.

6 LIMITATION AND FUTURE RESEARCH

One key limitation of our approach is the high computational cost of fine-tuning multimodal models, particularly in aligning graph representations with visual and textual modalities. Additionally, the model struggles with dynamic and interactive webpage elements, such as animations and scripts, which are not well-represented in our current graph structure.

Future research could delve into extending the graph-based representation to model interactive elements. Graphs are well-suited for representing relationships between components, and this can be expanded to capture the temporal and behavioral aspects of dynamic content. For instance, nodes and edges could represent not only static components and their spatial relationships but also interactions, such as hover effects, clicks, and animations. By incorporating state transitions or event-driven behaviors into the graph, the model could generate more comprehensive representations of how a webpage functions, making the framework applicable not only to static HTML but also to dynamic web environments that involve JavaScript or other client-side scripting languages.

REFERENCES

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andrew Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=EbMuimAbPbs.

Junyi An, Chao Qu, Zhipeng Zhou, Fenglei Cao, Xu Yinghui, Yuan Qi, and Furao Shen. Hybrid directional graph neural network for molecules. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=BBD6KXIGJL.

Anas Awadalla, Irena Gao, Josh Gardner, Jack Hessel, Yusuf Hanafy, Wanrong Zhu, Kalyani Marathe, Yonatan Bitton, Samir Gadre, Shiori Sagawa, Jenia Jitsev, Simon Kornblith, Pang Wei Koh, Gabriel Ilharco, Mitchell Wortsman, and Ludwig Schmidt. Openflamingo: An open-source framework for training large autoregressive vision-language models, 2023. URL https://arxiv.org/abs/2308.01390.

Pablo Barceló, Egor V. Kostylev, Mikael Monet, Jorge Pérez, Juan Reutter, and Juan Pablo Silva. The logical expressiveness of graph neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=r1lZ7AEKvB.

Tony Beltramelli. pix2code: Generating code from a graphical user interface screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '18, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450358972. doi: 10.1145/3220134.3220135. URL https://doi.org/10.1145/3220134.3220135.

Gianluca Bonifazi, Francesco Cauteruccio, Enrico Corradini, Michele Marchetti, Domenico Ursino, and Luca Virgili. A network analysis-based framework to understand the representation dynamics of graph neural networks. *Neural Computing and Applications*, 36(4):1875–1897, Feb 2024.

ISSN 1433-3058. doi: 10.1007/s00521-023-09181-w. URL https://doi.org/10.1007/s00521-023-09181-w.

Yen-Chun Chen, Linjie Li, Licheng Yu, Ahmed El Kholy, Faisal Ahmed, Zhe Gan, Yu Cheng, and Jingjing Liu. Uniter: Universal image-text representation learning. In *Computer Vision – ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXX*, pp. 104–120, Berlin, Heidelberg, 2020. Springer-Verlag. ISBN 978-3-030-58576-1. doi: 10.1007/978-3-030-58577-8_7. URL https://doi.org/10.1007/978-3-030-58577-8_7.

David Jaime Tena Cucala, Bernardo Cuenca Grau, Egor V. Kostylev, and Boris Motik. Explainable GNN-based models over knowledge graphs. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=CrCvGNHAIrz.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019. URL https://arxiv.org/abs/1810.04805.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. Palm-e: an embodied multimodal language model. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Wenqi Fan, Yao Ma, Qing Li, Yuan He, Eric Zhao, Jiliang Tang, and Dawei Yin. Graph neural networks for social recommendation. In *The World Wide Web Conference*, WWW '19, pp. 417–426, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366748. doi: 10.1145/3308558.3313488. URL https://doi.org/10.1145/3308558.3313488.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Shaoling Dong, Xing Zhou, and Wenbin Jiang. Vision2ui: A real-world dataset with layout for code generation from ui designs. *ArXiv*, abs/2404.06369, 2024. URL https://api.semanticscholar.org/CorpusID:269010048.

Sherzod Hakimov and David Schlangen. Images in language space: Exploring the suitability of large language models for vision & language tasks. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 14196–14210, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.894. URL https://aclanthology.org/2023.findings-acl.894.

Ahmet Iscen, Mathilde Caron, Alireza Fathi, and Cordelia Schmid. Retrieval-enhanced contrastive vision-text models, 2024. URL https://arxiv.org/abs/2306.07196.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Hugo Laurençon, Lucile Saulnier, Leo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander Rush, Douwe Kiela, Matthieu Cord, and Victor Sanh. Obelics: An open web-scale filtered dataset of interleaved image-text documents. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 71683–71702. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/e2cfb719f58585f779d0a4f9f07bd618-Paper-Datasets_and_Benchmarks.pdf.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. Unlocking the conversion of web screenshots into html code with the websight dataset, 2024. URL https://arxiv.org/abs/2403.09029.

Junnan Li, Ramprasaath Selvaraju, Akhilesh Gotmare, Shafiq Joty, Caiming Xiong, and Steven Chu Hong Hoi. Align before fuse: Vision and language representation learning with momentum

distillation. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 9694–9705. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/505259756244493872b7709a8a01b536-Paper.pdf.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 12888–12900. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/li22n.html.

Liunian Harold Li, Mark Yatskar, Da Yin, Cho-Jui Hsieh, and Kai-Wei Chang. Visualbert: A simple and performant baseline for vision and language, 2019. URL https://arxiv.org/abs/1908.03557.

Ji Lin, Hongxu Yin, Wei Ping, Pavlo Molchanov, Mohammad Shoeybi, and Song Han. Vila: On pretraining for visual language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 26689–26699, June 2024.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34892–34916. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6dcf277ea32ce3288914faf369fe6de0-Paper-Conference.pdf.

Qi Liu, Maximilian Nickel, and Douwe Kiela. Hyperbolic graph neural networks. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/103303dd56a731e377d01f6a37badae3-Paper.pdf.

Michael R. Lyu, Baishakhi Ray, Abhik Roychoudhury, Shin Hwei Tan, and Patanamon Thongtanunam. Automatic programming: Large language models and beyond, 2024. URL https://arxiv.org/abs/2405.02213.

Tuan Anh Nguyen and Christoph Csallner. Reverse engineering mobile application user interfaces with remaui (t). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 248–259, 2015. URL https://api.semanticscholar.org/CorpusID:7499368.

OpenAI, 3 2023. URL https://openai.com/index/gpt-4-research/.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pp. 701–710, New York, NY, USA, 2014. Association for Computing Machinery. ISBN 9781450329569. doi: 10.1145/2623330.2623732. URL https://doi.org/10.1145/2623330.2623732.

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 Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 8748–8763. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/radford21a.html.

Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 14501–14515. Curran Associates, Inc., 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/5d4834a159f1547b267a05a4e2b7cf5e-Paper-Conference.pdf.

Patrick Reiser, Marlen Neubert, André Eberhard, Luca Torresi, Chen Zhou, Chen Shao, Houssam Metni, Clint Van Hoesel, Henrik Schopmans, Timo Sommer, and Pascal Friederich. Graph neural networks for materials science and chemistry. *Communications Materials*, 3 (1), 11 2022. doi: 10.1038/s43246-022-00315-6. URL https://doi.org/10.1038/s43246-022-00315-6.

Alex Robinson. Sketch2code: Generating a website from a paper mockup, 2019. URL https://arxiv.org/abs/1905.13750.

Luana Ruiz, Fernando Gama, and Alejandro Ribeiro. Gated graph recurrent neural networks. *IEEE Transactions on Signal Processing*, 68:6303–6318, 2020. doi: 10.1109/TSP.2020.3033962.

ShantamVijayputra. Sketch2Code, 12 2022. URL https://www.kaggle.com/datasets/vshantam/sketch2code.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. Design2code: How far are we from automating front-end engineering?, 2024. URL https://arxiv.org/abs/2403.03163.

Davit Soselia, Khalid Saifullah, and Tianyi Zhou. Learning ui-to-code reverse generator using visual critic without rendering, 2023. URL https://arxiv.org/abs/2305.14637.

Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, Qipeng Guo, Xipeng Qiu, Pengcheng Yin, Xiaoli Li, Fei Yuan, Lingpeng Kong, Xiang Li, and Zhiyong Wu. A survey of neural code intelligence: Paradigms, advances and beyond, 2024. URL https://arxiv.org/abs/2403.14734.

Hao Tan and Mohit Bansal. LXMERT: Learning cross-modality encoder representations from transformers. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan (eds.), *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 5100–5111, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1514. URL https://aclanthology.org/D19-1514.

Weihan Wang, Qingsong Lv, Wenmeng Yu, Wenyi Hong, Ji Qi, Yan Wang, Junhui Ji, Zhuoyi Yang, Lei Zhao, Xixuan Song, Jiazheng Xu, Bin Xu, Juanzi Li, Yuxiao Dong, Ming Ding, and Jie Tang. Cogvlm: Visual expert for pretrained language models, 2024. URL https://arxiv.org/abs/2311.03079.

Jason Wu, Siyan Wang, Siman Shen, Yi-Hao Peng, Jeffrey Nichols, and Jeffrey P Bigham. Webui: A dataset for enhancing visual ui understanding with web semantics. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3581158. URL https://doi.org/10.1145/3544548.3581158.

Jun Xia, Chengshuai Zhao, Bozhen Hu, Zhangyang Gao, Cheng Tan, Yue Liu, Siyuan Li, and Stan Z. Li. Mole-BERT: Rethinking pre-training graph neural networks for molecules. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=jevY-DtiZTR.

Sukmin Yun, Haokun Lin, Rusiru Thushara, Mohammad Qazim Bhat, Yongxin Wang, Zutao Jiang, Mingkai Deng, Jinhong Wang, Tianhua Tao, Junbo Li, Haonan Li, Preslav Nakov, Timothy Baldwin, Zhengzhong Liu, Eric P. Xing, Xiaodan Liang, and Zhiqiang Shen. Web2code: A large-scale webpage-to-code dataset and evaluation framework for multimodal llms. *ArXiv*, abs/2406.20098, 2024. URL https://api.semanticscholar.org/CorpusID:270845897.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Yongji Wang, and Jian-Guang Lou. Large language models meet nl2code: A survey, 2023. URL https://arxiv.org/abs/2212.09420.