WAFFLE: Fine-tuning Multi-Modal Model for Automated Front-End Development

Shanchao Liang
Purdue University
liang422@purdue.edu

Nan Jiang
Purdue University
jiang719@purdue.edu

Shangshu Qian
Purdue University
qian151@purdue.edu

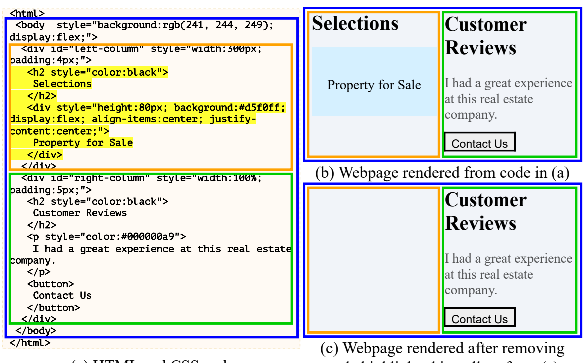
Lin Tan
Purdue University
lintan@purdue.edu

Abstract

Web development involves turning UI designs into functional webpages, which can be difficult for both beginners and experienced developers due to the complexity of HTML's hierarchical structures and styles. While Large Language Models (LLMs) have shown promise in generating source code, two major challenges persist in UI-to-HTML code generation: (1) effectively representing HTML's hierarchical structure for LLMs, and (2) bridging the gap between the visual nature of UI designs and the text-based format of HTML code. To tackle these challenges, we introduce WAFFLE, a new fine-tuning strategy that uses a structure-aware attention mechanism to improve LLMs' understanding of HTML's structure and a contrastive fine-tuning approach to align LLMs' understanding of UI images and HTML code. Models fine-tuned with WAFFLE show up to 9.00 pp (absolute percentage point) higher HTML match, 0.0982 higher CW-SSIM, 32.99 higher CLIP, and 27.12 pp higher LLEM on our new benchmark WebSight-Test and an existing benchmark Design2Code, outperforming current fine-tuning methods.

1 Introduction

While Large Language Models have significantly advanced the automation of code generation in popular programming languages such as Python or Java (Jiang et al., 2023; Touvron et al., 2023; Fried et al., 2023; Nijkamp et al., 2022; Rozière et al., 2024; Guo et al., 2024; Li et al., 2023c; Lozhkov et al., 2024), the automation of HTML code generation from UI design remains under-explored and challenging. As the core of front-end development, this task requires the model to understand not only the transformation from natural languages (NL) to programming languages (PL) but also from visual designs to PL. Recently, Multi-modal Large Language Models (MLLMs) have brought much progress in generating text from image descriptions (Radford et al., 2021; Zhai et al., 2023; Li et al., 2023a; Liu et al., 2023b,a; Li et al., 2022, 2023b; Dai et al., 2023; Blecher et al., 2023; Chen et al., 2023; Wei et al., 2023; vikhyat, 2024). On top of this, a few MLLMs have been fine-tuned using UI image-to-code datasets (e.g., WebSight (Laurençon et al., 2024), Design2Code (Si et al., 2024)). Nonetheless, these approaches mainly apply standard fine-tuning and fail to address specific HTML code generation challenges.



```
<html>
 <body  style="background:rgb(241, 244, 249);
display:flex;">
  <div id="left-column" style="width:300px;
padding:4px;">
   <h2 style="color:black">
    Selections
   </h2>
   <div style="height:80px; background:#d5f0ff;
display:flex; align-items:center; justify-
content:center;">
    Property for Sale
   </div>
  </div>
  <div id="right-column" style="width:100%;
padding:5px;">
   <h2 style="color:black">
    Customer Reviews
   </h2>
   <p style="color:#000000a9">
    I had a great experience at this real estate
company.
   </p>
   <button>
    Contact Us
   </button>
  </div>
 </body>
</html>
```

(a) HTML and CSS code

(b) Webpage rendered from code in (a)

(c) Webpage rendered after removing code highlighted in yellow from (a)

Figure 1: Removing the children of the element `<div id = "left-column">` highlighted in yellow does not affect the structure of the visual layout of itself or its sibling element `<div id="right-column">`.

Two key challenges exist in translating UI design images to HTML code: (1) how to teach models to learn effectively the **domain knowledge of HTML structures**, which significantly influences the rendered UI design, and (2) how to teach the models to learn the **subtle differences in the visual understanding of UI images and text understanding of HTML code**.

Regarding the first challenge, there are three basic structural aspects of HTML code. Firstly, all the styles of the parent element are directly passed to the children unless specifically overridden. Secondly, the layout of the siblings affects each other. Thirdly, nodes are not affected by the subtrees of their siblings. The last principle might be less obvious compared to the first two, and we explain it with an example. Figure 1 shows (a) an HTML code file and (b) its rendered webpage. We use blue, orange, and green blocks to map the code chunks and their corresponding visual rendering on the webpage. The top-level `<body>` element refers to the whole webpage, the child element `div id="left-column"` refers to the left part of the webpage, and another child element `div id="right-column"` refers to the right part. Modifications to the child of `div id="left-column"` do not change how the `div id="right-column"` looks on the webpage (even if we remove all the content inside `div id="left-column"` as (c) shows).

To learn such domain knowledge of HTML code structure, we propose a novel **structure-aware attention** mechanism. The structure-aware attention captures the structure information of the HTML code by explicitly allowing tokens to focus on three types of previous code segments that are the most relevant (details in Section 2.2). With such structural information in the HTML code, WAFFLE can focus on parts of the code that have the most influence on the resulting UI design, thus benefiting the overall performance.

For the second challenge, minor variations in layout structure can still impact the content of the generated code. However, MLLMs often fail to capture these subtle differences and generate identical code for visually distinct inputs. We illustrate this issue in Appendix A.2. To enable MLLMs to recognize subtle differences in UI images due to minor changes in the code, we adopt **contrastive learning** (Zhai et al., 2023; Radford et al., 2021; Gao et al., 2021) to the current task to teach MLLMs to focus on important visual differences.

Combining the two approaches, this paper introduces WAFFLE, a fine-tuning pipeline specifically designed for UI images to HTML code generation, with the following contributions:

1. We design *structure-aware attention* for HTML code generation, which enables MLLMs to learn the structure knowledge of HTML code.
2. We apply *contrastive learning* algorithms to boost MLLMs' understanding of the details in the rendered UI images and the HTML code.
3. We create a new dataset of 231,940 pairs of webpages and their HTML code, which could facilitate future research on web MLLMs.
4. We conduct comprehensive experiments on two backbone MLLMs. WAFFLE improves the backbone MLLMs by achieving up to 9.00 pp higher HTML Match, 0.0982 higher CW-SSIM, 32.99 higher CLIP, and 27.12 pp higher LLEM.
5. We highlight that WAFFLE as a fine-tuning approach, is model-independent and can be applied to improve any MLLMs for UI-to-HTML code generation.

Availability: https://github.com/lt-asset/Waffle

2 Approach

Figure 2 represents the overview of WAFFLE. We create a new mutated HTML dataset (Section 2.1) for training and fine-tuning. In addition, we design structure-aware attention (Section 2.2) during model fine-tuning to teach models to focus on important HTML segments. Finally, we use contrastive learning training to teach models to learn visual differences and align the models' visual and HMTL/CSS code understanding (Section 2.3). *We note that* WAFFLE *is a generalizable fine-tuning pipeline that can benefit any pre-trained MLLMs.*

Specifically, we construct the training dataset by applying mutation rules for HTML code on a subset of a popular dataset, WebSight-v0.1, to generate the corresponding source code and UI images.

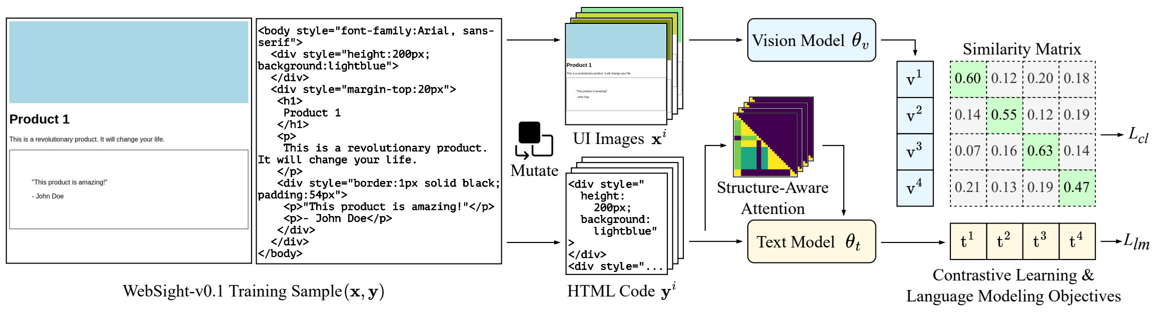
CSS						HTML	Total
Color	Size	Margin	Font	Display	Position		
12	11	19	10	1	2	8	63

Table 1: Most frequent causes of failures.

2.1 Training Data Mutation

To teach MLLMs the important visual differences, we create WAFFLE's contrastive training data from WebSight-v0.1, which is a fine-tuning dataset built by HuggingfaceM4 and contains 822,987 pairs of HTML code and its corresponding screenshots (Laurençon et al., 2024). However, to make contrastive learning effective, the model must learn to recognize subtle yet realistic variations in UI elements.

To achieve this, we analyze common mistakes in VLM-WebSight by conducting a failure analysis on 50 validation samples, identifying seven common error categories (Table 1). Using insights from these failures, we design realistic mutation rules to modify HTML/CSS in WebSight-v0.1, ensuring that the contrastive training data reflects real-world errors. The mutation process follows the observed error frequencies; for example, since color mismatches account for 19.05% of errors (12/63), mu-


Product 1
This is a revolutionary product. It will change your life.
"This product is amazing!"
- John Doe
Mutate
UI Images $\mathbf{x}^i$
Vision Model θ_v
Structure-Aware Attention
Text Model θ_t
Similarity Matrix
L_{cl}
L_{lm}
WebSight-v0.1 Training Sample $(\mathbf{x}, \mathbf{y})$
HTML Code $\mathbf{y}^i$
Contrastive Learning & Language Modeling Objectives

```
<body style="font-family:Arial, sans-
serif">
  <div style="height:200px;
background:lightblue">
  </div>
  <div style="margin-top:20px">
   <h1>
    Product 1
   </h1>
   <p>
    This is a revolutionary product.
It will change your life.
   </p>
   <div style="border:1px solid black;
padding:54px">
    <p>"This product is amazing!"</p>
    <p>- John Doe</p>
   </div>
  </div>
</body>
```

```
<div style="
  height:
    200px;
  background:
    lightblue"
>
</div>
<div style="...
```

v^1	0.60	0.12	0.20	0.18
v^2	0.14	0.55	0.12	0.19
v^3	0.07	0.16	0.63	0.14
v^4	0.21	0.13	0.19	0.47
	t^1	t^2	t^3	t^4

Figure 2: Overview of WAFFLE, including training data mutation, structure-aware attention, and contrastive learning.

tations on color properties are applied at the same rate (Table 1). The full set of mutation rules is detailed in Appendix A.1.

Based on the mutation rule, we randomly sample 100,000 instances from the WebSight-v0.1 dataset, creating four distinct mutants with each sample. The mutation rules are applied based on the frequency of failures computed from the validation set. The final mutated dataset (after removing rendering failures, identical mutants, and blank images) has 57,985 groups, each group containing four pairs of HTML code and the corresponding rendered webpage images.

2.2 Structure-Aware Attention

HTML code has clear structures, and certain structural properties can be directly reflected in the rendered UI design. Such domain knowledge can benefit the generation process of MLLMs. There are three most important elements for rendering an element's layout: its parent element, sibling elements, and the element itself. WAFFLE implements a novel attention mask that provides each element with a pruned view of all previous tokens, including *parent-attention*, *sibling-attention*, and *self-attention*. These attention masks allow the tokens to pay specialized attention to their parent elements, sibling elements, and themselves.

Figure 3 shows a simple example of WAFFLE's structure-aware attention. Figure 3 (a) shows an HTML code snippet and (b) shows the DOM tree of HTML code in (a), where the root node is the `<body>` element. `<div id="leftCol">` and `<div id="rightCol">` are two children of node `<body>`, and they are siblings to each other. `<div id="leftCol">` has one child, the text `Selections`. `<div id="rightCol">` has one child, which is a `<h2>` element with the text `Customer Reviews` inside.

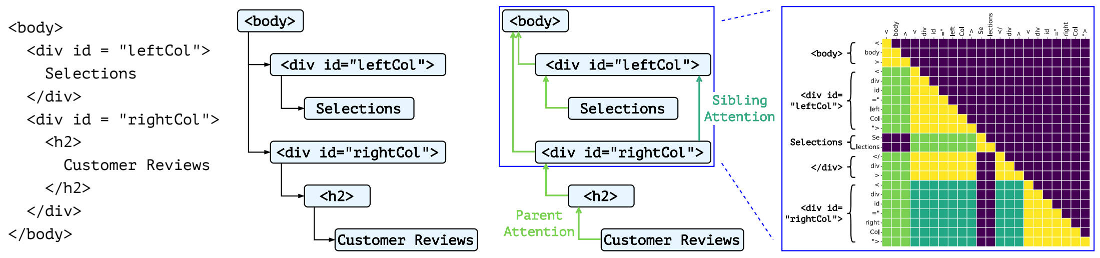
According to the domain knowledge that an element is mostly affected by its parent and sibling elements, WAFFLE builds the structure-aware attentionas shown in (c).

Parent-Attention. The parent-attention is from each element's tokens to its parent element's tokens. WAFFLE utilizes the fact that all the children elements inherit the parent element's styles and structure. For instance, the tokens of the element `<div id="leftCol">` pay parent-attention to tokens of the element `<body>`, and the tokens of the element `Selections` pay parent-attention to tokens of the element `<div id="leftCol">`.

Sibling-Attention. The sibling attention is from each element's tokens to its preceding sibling elements' tokens. WAFFLE utilizes the fact that sibling elements under the same parent can affect the arrangement and style of each other, so each element needs to pay attention to its preceding siblings. For instance, the tokens of the element `<div id="rightCol">` pay sibling-attention to tokens of the element `<div id="leftCol">`.

Self-Attention. This is the standard self-attention mechanism, it allows each token to focus on all previous tokens within the same element, excluding the children elements. To illustrate, all tokens in a specific element have self-attention (yellow cells Figure 3) to all the tokens belonging to the element itself.

WAFFLE applies the structure-aware attention mechanism exclusively to the language model decoder while keeping the vision encoder unchanged. Specifically, WAFFLE applies the structure-aware attention mask—formed by the union of parent-attention, sibling-attention, and self-attention masks—to only one-fourth of the attention heads in the decoder. This enables these heads to explicitly capture structural domain knowledge, while the remaining three-fourths retain full self-attention and leverage pre-trained knowledge. The number of attention heads using structure-aware attention is a tunable hyperparameter, which can be adjusted as described in Section 4.3.



```
<body>
  <div id = "leftCol">
    Selections
  </div>
  <div id = "rightCol">
    <h2>
      Customer Reviews
    </h2>
  </div>
</body>
```

(a) HTML Code (b) DOM Tree (c) Structure-Aware Attention

Figure 3: Example of structure-aware attention. Yellow : self-attention. Green : parent-attention. Teal : sibling-attention. Purple : masked-out Attention.

2.3 Contrastive-Learning

To address current models' limited ability to effectively map variations in HTML code to the corresponding UI, WAFFLE utilizes contrastive learning, allowing models to learn from comparisons and contrasts between similar examples.

More concretely, the training dataset consists of groups of HTML codes and UI Image pairs, where each group has k such pairs. During training, in any group, for image $\mathbf{x}^i$ and code $\mathbf{y}^i$ ($i \in \{1, ..., k\}$, image $\mathbf{x}^i$ is the rendered webpage image from code $\mathbf{y}^i$), the webpage image is split into a list of pixel patches. Each patch is encoded by the vision model θ_v and fused to the text model's latent space using an adapter to result in a list of patch embeddings $\mathbf{v}^i = \{v_1^i, v_2^i, ..., v_M^i\}$ (M is a hyper-parameter of the backbone MLLM). The HTML code is tokenized into tokens $\mathbf{y}^i = \{y_1^i, y_2^i, ..., y_{N^i}^i\}$, where N^i is the number of the tokens of the HTML code $\mathbf{y}^i$. These tokens are encoded to embeddings $\mathbf{t}^i = \{t_1^i, t_2^i, ..., t_{N^i}^i\}$ by the language model θ_t (using the structure-aware attention). We use the average over all the patch embeddings to represent the embeddings of the webpage image, and the average overall all the tokens' embeddings to represent the embedding of the whole HTML code, denoted by:

$$\overline{v^i} = \frac{1}{P}\sum_{j=1}^{P} v_j^i \quad , \quad \overline{t^i} = \frac{1}{N^i}\sum_{j=1}^{N^i} t_j^i \tag{1}$$

Then we calculate the cosine similarity score between the code and image embeddings, $\overline{t^i}$ and $\overline{v^i}$, bipartitely. The contrastive learning objective is to maximize the similarity between the embeddings of the corresponding code and UI, by minimizing the contrastive learning loss L_{cl}, along with the language modeling loss L_{lm} as follows:

$$L_{cl} = -\sum_{i=1}^{k}\left(\frac{exp\left(sim\left(\overline{t^i}, \overline{v^i}\right)\right)}{\sum_{j=1}^{k} exp\left(sim\left(\overline{t^i}, \overline{v^j}\right)\right)}\right) \tag{2}$$

$$L_{lm} = -\sum_{i=1}^{k}\sum_{j=1}^{N_i} \log P(y_j^i \mid \mathbf{y}_{<j}^i, \mathbf{x}^i, \theta_t, \theta_v) \tag{3}$$

The contrastive learning loss aims to maximize the similarity scores at the diagonal of the similarity matrix (as the green cells of the similarity matrix shown in Figure 2). It trains the MLLM's vision model, θ_v, which encodes a webpage, $\mathbf{x}^i$, to closely match the encoded embeddings of the text model, θ_t, on the corresponding HTML code, $\mathbf{y}^i$.

The language modeling loss, on the other hand, aims to maximize the probability of generating the correct token y_j^i given all previous token $\mathbf{y}_{<j}^i$ and the input webpage image $\mathbf{x}^i$. This is the standard objective of training an MLLM to generate text from images. WAFFLE jointly optimizes the two objectives, $L_{\text{WAFFLE}} = L_{lm} + \lambda L_{cl}$, where λ is a hyper-parameter constant controlling the effect of contrastive learning on the overall optimization.

3 Experimental Setup

3.1 Model Training

We implement WAFFLE on two backbones, VLM-WebSight, and Moondream2 (Laurençon et al., 2024; vikhyat, 2024). Each backbone is first fine-tuned on the WebSight-v0.1 dataset using the standard language modeling objective. For VLM-Websight, we use the released fine-tuned checkpoint with details in (Laurençon et al., 2024). This checkpoint is fine-tuned using DoRA (Liu et al., 2024) (a variant of parameter-efficient training, LoRA (Hu et al., 2022), with `rank` set to 64). For Moondream2, we also fine-tune the model using DoRA (Liu et al., 2024) (with `rank` set to 64, and `lora_alpha` set to 128). The model's weights are updated using the AdamW (Loshchilov and Hutter, 2019) optimizer, with the learning rate set to $3e^{-5}$. The batch size is 64.

On top of the fine-tuned MLLMs from the first step, we apply the structure-aware attention and contrastive learning approach. Structure-aware attention is applied to $\frac{1}{4}$ of the attention heads in each attention layer in the LLM decoder. Each model is trained with DoRA using the combined learning objective on the contrastive learning dataset (λ set to 0.1). The model's weights are updated using the AdamW optimizer, with the learning rate set to $2e^{-5}$. The batch size is 32.

3.2 Test Data

We evaluate WAFFLE using two test datasets: WebSight-Test, which consists of synthetic webpages, and Design2Code, which consists of real-world webpages. Since WebSight-v0.1 was already used for training, we created WebSight-Test following the same process as WebSight-v0.1 (Laurençon et al., 2024). In total, WebSight-Test contains 500 test samples, each has a webpage image and the respective ground-truth HTML source code.

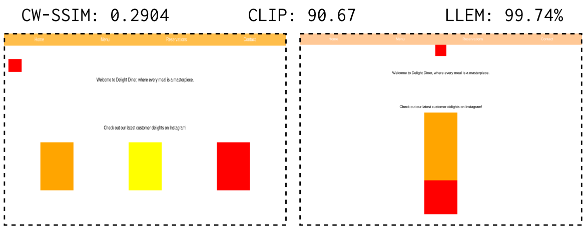
Design2Code is an open-source benchmark of 484 manually processed real-world website screenshots, which are more complex than Websight-v0.1. Evaluation on Design2Code indicates the generalization ability of models fine-tuned on WAFFLE's training dataset to real-world scenarios.

3.3 Evaluation Metrics

We note that existing methods struggle to effectively capture subtle or structural differences in images. To address this, we adopt CLIP and Low-Level Element Matching (LLEM) metrics from previous work (Si et al., 2024). Additionally, we introduce two new metrics: HTML-Match, designed to evaluate structural similarity in rendered HTML, and complex wavelet structural similarity index (CW-SSIM) to detect minor differences in images, respectively.

HTML-Match. HTML-Match is the percentage of generated images that match the ground truth images perfectly at the pixel level. For this comparison, styles and attributes are removed from both the ground truth and generated HTML. This process emphasizes the model's ability to accurately recognize the text content and the DOM tree structure of the HTML code, and assess MLLM's ability to recover the HTML structure from the given UI.

CW-SSIM. CW-SSIM computes the structural similarity between images (Sampat et al., 2009). In certain cases, LLEM and CLIP scores might yield high scores as they failed to capture the structural difference. For example, in Figure 4 the rendered


CW-SSIM: 0.2904
CLIP: 90.67
LLEM: 99.74%
Home
Menu
Reservations
Contact
Welcome to Delight Diner, where every meal is a masterpiece.
Check out our latest customer delights on Instagram!
Home
Menu
Reservations
Contact
Welcome to Delight Diner, where every meal is a masterpiece.
Check out our latest customer delights on Instagram!

(a) Ground Truth Image (b) Inferred Image

Figure 4: Different metrics on a pair of images.

image is structurally different from the ground truth image, resulting in low CW-SSIM scores, 0.2904. However, the LLEM gives a high score of 99.74%, and the CLIP score is 90.67, which does not align with human evaluation as well as CW-SSIM.

CLIP. CLIP score (Radford et al., 2021; Si et al., 2024) measures the similarity between the rendered webpage of the inferred HTML code and the ground truth webpage based on the `CLIP-ViT-B/32`'s embeddings of webpages.

Low-Level Element Matching (LLEM). Previous work (Si et al., 2024) proposes LLEM to measure the percentage of matched (1) text blocks, (2) text content, (3) position of each matched text block, and (4) font color within each text block.

In this work, we prioritize CW-SSIM as a more robust evaluation metric. Yet, we still report CLIP and LLEM as they are used in existing work.

Human Evaluation. Two human annotators are asked to compare the rendered webpages generated by different techniques from a subset of test data with the ground-truth webpage and rank them.

4 Results

4.1 Effectiveness of WAFFLE

Table 2 show the performance of various fine-tuning strategies on the two testing datasets, WebSight-Test (500 samples) and Design2Code (484 samples).

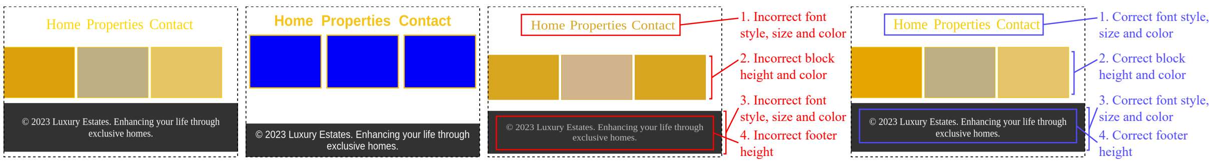
Comparison against standard fine-tuning. We compare WAFFLE with the baseline, the standard fine-tuning (FT) method. In both tables, WAFFLE achieves significant improvements over the standard FT on ***all metrics*** with Moondream2 and VLM-WebSight as the backbone.

On WebSight-Test, WAFFLE achieves 6.00 pp higher HTML-Match (27.60% vs. 21.60%) and 0.0253 higher CW-SSIM (0.4486 vs. 0.4233) than Standard FT with Moondream2 as the backbone. On the VLM-WebSight backbone, WAFFLE outperforms Standard FT, i.e., original VLM-WebSight,

Backbones	Techniques	WebSight-Test				Design2Code		
		HTML-Match (%) ↑	CW-SSIM ↑	CLIP ↑	LLEM (%) ↑	CW-SSIM ↑	CLIP ↑	LLEM (%) ↑
Gemini 1.5 Pro	Prompting	9.40	0.3385	88.55	90.16	0.2652	87.76	87.17
GPT-4o mini	Prompting	10.20	0.3055	87.72	87.54	0.2304	86.06	78.84
GPT-4o	Prompting	11.40	0.3666	89.03	92.18	89.03	83.67	75.98
Moondream2	Standard FT	21.60	0.4233	89.92	90.59	0.1348	46.63	40.71
	WAFFLE	**27.60**	**0.4486**	**89.98**	**91.72**	**0.2142**	**79.62**	**67.83**
VLM-WebSight	Standard FT	28.00	0.5023	93.30	92.73	0.2518	82.35	73.00
	WAFFLE	**37.00**	**0.6005**	**94.57**	**95.16**	**0.2815**	**85.98**	**77.81**

*Gemini 1.5 Pro's results are on 384 test instances as it generates no answers for the remaining 100 instances.

Table 2: Main results on the WebSight-Test and Design2Code dataset.


Home Properties Contact
© 2023 Luxury Estates. Enhancing your life through exclusive homes.
Home Properties Contact
© 2023 Luxury Estates. Enhancing your life through exclusive homes.
Home Properties Contact
© 2023 Luxury Estates. Enhancing your life through exclusive homes.
1. Incorrect font style, size and color
2. Incorrect block height and color
3. Incorrect font style, size and color
4. Incorrect footer height
Home Properties Contact
© 2023 Luxury Estates. Enhancing your life through exclusive homes.
1. Correct font style, size and color
2. Correct block height and color
3. Correct font style, size and color
4. Correct footer height

(a) Expected Webpage (b) GPT-4o (CW-SSIM: 0.1353) (c) Standard FT (CW-SSIM: 0.3760) (d) WAFFLE (CW-SSIM: 0.9995)

Figure 5: Example from WebSight-Test dataset, with the generated images by GPT-4o, Standard FT (VLM-WebSight), and WAFFLE (VLM-WebSight).

by 9.00 pp HTML-Match, 0.0982 CW-SSIM, improving the model's ability to generate structurally similar images and align the two modalities. On Design2Code, WAFFLE achieves greater improvements compared to the Standard FT technique with both backbones.

Summary: Overall, WAFFLE significantly improves all metrics for both backbones in both test datasets over standard fine-tuning with up to 9.00 pp for HTML Match, 0.0982 for CW-SSIM, 32.99 for CLIP, and 27.12 pp for LLEM.

Comparison against SOTA commercial models. Due to the lack of comparable baselines, we compare WAFFLE with top commercial models, which include GPT-4o mini, GPT-4o, and Gemini 1.5 Pro. We apply direct prompting following prior work (Si et al., 2024). The result is shown in Table 2.

On the WebSight-Test dataset, models fine-tuned with WAFFLE perform better than the SOTA commercial models. VLM-WebSight overperforms GPT-4o by 25.60 pp on HTML-Match (37.00% vs. 11.40%) and 0.2339 on CW-SSIM (0.6005 vs. 0.3666), showing WAFFLE's benefit in addressing the two challenges, i.e., generate structurally correct HTML and closing the gap between the two modalities. Similarly, for the smaller backbone, Moondream2 exceeds GPT-4o on CW-SSIM and HTML-Match.

As shown in Table 2, VLM-WebSight fine-tuned by WAFFLE is better than GPT-4o on CW-SSIM by 0.039, and better than GPT-4o mini by 0.511 on the Design2Code dataset. However, GPT-4o is better in the other two metrics versus VLM-WebSight. Moondream2 fine-tuned by WAFFLE has a lower performance compared to GPT-4o and GPT-4o mini on all metrics. This is likely due to its smaller size, which could influence its generalizability to more complex, out-of-distribution data.

Summary: On simpler data, WAFFLE achieves better or comparable results than SOTA commercial models, with 16.20–25.60 pp improvement on HTML-Match and 0.0820–0.2339 improvement on CW-SSIM. On more complex data, WAFFLE enables VLM-WebSight to outperform commercial models on CW-SSIM.

4.2 Case Study

Figure 5 shows the generation results for one instance from WebSight-Test. The generated webpage of GPT-4o has a CW-SSIM of 0.1353, significantly lower than that of 0.3760 from VLM-WebSight under standard fine-tuning. On the other hand, the webpage generated by VLM-WebSight fine-tuned by WAFFLE reaches an almost perfect CW-SSIM score. This example shows the effectiveness of using WAFFLE on UI-to-HTML generation.

4.3 Ablation Studies

Comparison against ablation models. We compare WAFFLE with two ablation models:

- WAFFLE-`attn`: This is WAFFLE with only contrastive learning and without the use of structure-aware attention.

Backbones	Techniques	WebSight-Test				Design2Code		
		HTML-Match (%) ↑	CW-SSIM ↑	CLIP ↑	LLEM (%) ↑	CW-SSIM ↑	CLIP ↑	LLEM (%) ↑
Moondream2	Standard FT	21.60	0.4233	89.92	90.59	0.1348	46.63	40.71
	WAFFLE-attn	23.60	0.4311	**90.47**	91.34	0.1821	67.73	56.49
	WAFFLE-contra	26.00	0.4296	89.55	91.21	0.2100	76.63	65.82
	WAFFLE	**27.60**	**0.4486**	89.98	**91.72**	**0.2142**	**79.62**	**67.83**
VLM-WebSight	Standard FT	28.00	0.5023	93.30	92.73	0.2518	82.35	73.00
	WAFFLE-attn	30.80	0.5411	94.29	94.20	0.2480	85.64	75.34
	WAFFLE-contra	35.80	0.5677	**95.08**	**95.30**	0.2653	85.16	76.48
	WAFFLE	**37.00**	**0.6005**	94.57	95.16	**0.2815**	**85.98**	**77.81**

Table 3: Ablation studies on the two test datasets. LLEM refers to the averaged Low-Level Element Matching.

Techniques	Rank 1 ↑	Rank 2 ↑	Rank 3 ↑	Avg Rankings ↓
Standard FT	9 | 34 (43)	18 | 13 (46)	34 | 25 (54)	2.88 | 2.37 (2.62)
WAFFLE-attn	22 | 23 (45)	11 | 30 (41)	22 | 26 (71)	2.67 | 2.45 (2.56)
WAFFLE-contra	60 | 33 (93)	13 | 17 (30)	11 | 26 (42)	1.78 | 2.41 (2.10)
WAFFLE	46 | 54 (**100**)	33 | 22 (55)	10 | 15 (26)	1.85 | 1.79 (**1.82**)

Table 4: Human evaluation on two datasets using VLM-WebSight as the backbone. The numbers are shown as "x|y (x+y)", where x is the result on WebSight-Test and y is the result on Design2Code.

- WAFFLE-contra: This is WAFFLE with only structure-aware attention.

Shown in Table 3, WAFFLE-attn brings improvements compared to the Standard FT on all metrics, and WAFFLE-contra brings a 4.40 pp improvement on HTML-Match. On the Design2Code dataset, WAFFLE has dominating performance on ***all metrics*** for models fine-tuned with both backbones. Across the two backbones, models fine-tuned with WAFFLE are higher than those fine-tuned with WAFFLE-attn by up to 0.0335 on CW-SSIM.

Summary: Contrastive learning and structure-aware attention significantly improve performance over standard fine-tuning. On the simpler WebSight-Test data, models trained with WAFFLE achieve the highest HTML-Match and CW-SSIM scores. On the more complex Design2Code data, WAFFLE still delivers the best results across all metrics for both backbones.

Human evaluation results. We select 50 test samples from WebSight-Test and Design2Code (100 total). Each sample has four generated HTML codes and rendered webpages from Standard FT, WAFFLE-attn, WAFFLE-contra, and WAFFLE. Human raters rank the generated webpages based on similarity to the ground-truth webpage without knowing which model produced each one. Multiple results can share the same rank if they are deemed equally similar to the ground truth. Table 4 shows the human evaluation results for VLM-WebSight fine-tuned by Standard FT, WAFFLE-attn, WAFFLE-contra, and WAFFLE.

Techniques	Prior	Current	Drop (%)
WAFFLE-attn	0.8002	0.5797	27.55
WAFFLE	0.8291	0.7932	4.34

Table 5: CW-SSIM on 20 samples using the VLM-WebSight backbone. "Prior" refers to "without intermediate mistakes", and "Current" to "with intermediate mistakes".

Across both datasets, WAFFLE has the best averaged rankings, 1.82, outperforming both ablations and the baseline. Specifically, WAFFLE reaches 54 times rank 1 placement on Design2Code, showing great generalizability on more complex datasets. WAFFLE-contra is the second best technique on the two testsets, reaching 93 times rank 1, and an average ranking of 2.10. On the other hand, WAFFLE-attn is the third-best technique but still outperforms standard FT.

Summary: Human evaluation shows that (1) both structure-aware attention and contrastive learning contribute to the code generation quality, and (2) WAFFLE-generated HTML/CSS code is consistently rated higher than code generated with standard fine-tuning.

4.4 Structure-Aware Attention's Effect

To demonstrate how structure-aware attention helps MLLMs focus on the correct structural elements (such as parent and sibling elements) during generation, we introduce controlled errors in the generation process. Specifically, we select 20 high-performing samples—those with the highest CW-SSIM scores—generated by VLM-WebSight fine-tuned with WAFFLE and WAFFLE-attn. These samples indicate cases where the models originally performed well on UI-to-code generation. Our goal is to evaluate whether models fine-tuned with structure-aware attention exhibit greater robustness in handling intermediate errors compared to those without it.

For each selected sample, we manually modify the partially generated HTML code by editing the sibling elements of the correct structure, following the example in Figure 1. These modifications alter the rendered output, simulating realistic errors that could occur during generation. The models are then tasked with re-completing the HTML code while being provided the same UI image with the partially generated, incorrect HTML as input. Since the modifications affect sibling elements rather than the primary structure, an ideal model should recover from these errors without cascading failures in subsequent generations.

Table 5 shows the results of re-completion following the intermediate mistakes. With WAFFLE-attn, the averaged CW-SSIM across the 20 samples drops by 0.2205 (from 0.8002 to 0.5797) if the model makes intermediate mistakes. By contrast, with WAFFLE, the averaged CW-SSIM only drops by 0.0359, from 0.8291 to 0.7932.

Summary: Integrating structure-aware attention brings stability to model generation, making models more robust against intermediate mistakes, and reducing the performance drop by 23.31%, from 27.55% to 4.24%, ensuring more consistent generation quality.

4.5 Contrastive Learning's Effect

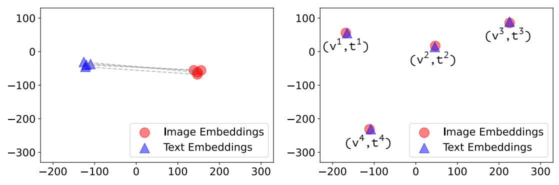
To show contrastive learning's effect on MLLMs' visual and textual understanding, we design two experiments. The first experiment analyzes whether MLLM's understanding of the image and code is aligned through the integration of contrastive learning, and the second experiment analyzes whether contrastive learning can teach the model to capture the subtle difference in the images.

Aligning models' two modalities. Specifically, under the same procedure, we compute the averaged text embeddings and image embeddings for a subset (12 samples from WebSight-Test dataset) of the test samples in Section 4.4 using the Moondream2 model fine-tuned by WAFFLE-attn and Standard FT. Then for each pair of the averaged image embeddings and text embeddings, $(\overline{v^i}, \overline{t^i})$, we normalize them and compute the Euclidean distance and the cosine similarity between them.

Table 6 shows the results of the measurements for both techniques. The Euclidean distance between the embeddings of the two modalities is 0.8447 for WAFFLE-attn, which is lower than 1.3395, the distance of the embeddings from Standard FT, by 0.4798. Similarly, the cosine similarity

Techniques	$d(\overline{v^i}, \overline{t^i}) \downarrow$	$sim(\overline{v^i}, \overline{t^i}) \uparrow$
Standard FT	1.3395	0.1027
WAFFLE-attn	0.8447	0.6244

Table 6: Distance (d) and similarity (sim) between averaged image embeddings $\overline{v^i}$ and text embeddings $\overline{t^i}$, using Moondream2 as the backbone.



(a) Standard FT's Embeddings (b) WAFFLE-attn's Embeddings

Figure 6: t-SNE plots of the text and image embeddings, computed by Moondream2 fine-tuned with Standard FT and WAFFLE-attn.

between the embeddings encoded by WAFFLE-attn is higher than the Standard FT by 0.5017 (0.6244 vs. 0.1027).

In addition, Figure 6 demonstrates that contrastive learning teaches the model to align the text and image understandings. The vision embeddings (red circles) are far away from their corresponding text embeddings (blue triangles) when calculated by Standard FT. By contrast, the vision embeddings are grouped with their corresponding text embeddings by WAFFLE-attn.

Summary: Fine-tuning with contrastive learning (WAFFLE-attn) enhances MLLM visual-textual alignment, making image and code embeddings more cohesive, reducing Euclidean distance while elevating the cosine similarity between them.

5 Related Work

5.1 Multi-Modal Large Language Models

Recent advances in vision and language models have greatly improved MLLMs' capabilities in tasks like image captioning (Zhai et al., 2023; Li et al., 2022, 2023b; Laurençon et al., 2023), text-to-image generation (Ramesh et al., 2022; Rombach et al., 2021), visual question answering (Liu et al., 2023b,a; Bai et al., 2023), and document editing (Suri et al., 2024). While popular models like Llava (Liu et al., 2023b,a), Qwen-VL (Bai et al., 2023), and Vary (Wei et al., 2023) perform well in general image tasks, they don't focus on converting UI images to HTML code. To address this, we propose WAFFLE, a fine-tuning method that equips MLLMs with domain-specific knowledge needed for UI-to-HTML generation.

5.2 Attention Mechanism

The attention mechanism is the key part of modern Transformer (Vaswani et al., 2017) architectures, as it effectively captures the hidden features of input data. To handle the challenges of certain domains, specialized attention mechanisms have been explored, such as pyramid attention (Chai and Li, 2022) and hierarchical attention (Guo et al., 2023; Shi et al., 2021; Nguyen et al., 2023; Yang et al., 2016) designed for long-range, cross-file code understanding and generation, as well as regularized attention for assembly code (Su et al., 2024). Different from existing work, WAFFLE targets HTML code, with the new challenge of its restricted structure. WAFFLE designs a novel structure-aware attention to learn such structure knowledge.

5.3 UI to HTML Generation

Early direction for UI code generation utilizes sketch webpage figures, e.g., hand-drawn website sketches, to generate UI code that can be rendered into similar images as sketch images (Robinson, 2019). Yet, this direction is not practical, as not all front-end developers want to draw out a sketch website when they need help from an automated tool. In contrast, leveraging advancements in MLLMs, Huggingface has recently released WebSight, which is trained on the WebSight-v0.1 dataset (Laurençon et al., 2024). Although specific details of the model are not disclosed, it represents a significant shift towards end-to-end UI to code generation. Similarly, Design2Code-18B is a model using CogAgent as the backbone using a subset of WebSight-v0.1 (Si et al., 2024; Hong et al., 2023). However, neither WebSight nor Design2Code tries to adapt domain knowledge of HTML for this task. In contrast, we provide structure-aware attention and apply contrastive learning with the mutations to teach the model the fine-grained difference of HTML images.

6 Conclusion

This work presents WAFFLE, a fine-tuning pipeline for UI-to-HTML code generation, that is generalizable to any transformer-based MLLMs. WAFFLE introduces a structure-aware attention mechanism to capture HTML structure and employs contrastive learning to align visual and textual understanding, aiding MLLMs in distinguishing subtle webpage differences. WAFFLE outperforms standard fine-tuning on two backbone MLLMs, with improvements of up to 9 pp in HTML Match, 0.0982 in CW-SSIM, 32.99 in CLIP, and 27.12 pp in LLEM. Ablation studies confirm that both key components of WAFFLE contribute to better cross-modality understanding and more robust code generation. Notably, WAFFLE is model-independent and can enhance any MLLMs for UI-to-HTML code generation.

7 Limitation

One limitation of WAFFLE is that it has only been implemented on two models: VLM-WebSight and Moondream2. While WAFFLE could potentially be applied to any MLLM, like Design2Code, further exploration is limited by computing resources. Our experiments show that WAFFLE brings significant improvements over standard fine-tuning on these two models, indicating some level of generalizability. Another limitation is that the metrics we use do not fully capture human evaluation. HTML-Match overlooks CSS styling, and metrics like CW-SSIM, CLIP, and LLEM are similarity-based, which can lead to unreliable scores. Evaluating HTML code automatically is challenging, so we include CLIP and LLEM, as used in previous work (Si et al., 2024; Gui et al., 2024), along with CW-SSIM and HTML-Match to ensure fair evaluation.

8 Acknowledgements

We thank the anonymous reviewers for their feedback on this work. This research was supported in part by NSF 1901242 and 2006688 and a CFI fund. This work also used Anvil at Purdue University through allocation CIS240304 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program (Boerner et al., 2023), which is supported by U.S. National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

References

Jinze Bai, Shuai Bai, Shusheng Yang, Shijie Wang, Sinan Tan, Peng Wang, Junyang Lin, Chang Zhou, and Jingren Zhou. 2023. Qwen-vl: A frontier large vision-language model with versatile abilities. *arXiv preprint arXiv:2308.12966*.

Lukas Blecher, Guillem Cucurull, Thomas Scialom, and Robert Stojnic. 2023. Nougat: Neural optical understanding for academic documents. *Preprint*, arXiv:2308.13418.

Timothy J. Boerner, Stephen Deems, Thomas R. Furlani, Shelley L. Knuth, and John Towns. 2023. Access: Advancing innovation: Nsf's advanced cyberinfrastructure coordination ecosystem: Services & support. PEARC '23, page 173–176, New York, NY, USA. Association for Computing Machinery.

Lei Chai and Ming Li. 2022. Pyramid attention for source code summarization. *Advances in Neural Information Processing Systems*, 35:20421–20433.

Lin Chen, Jisong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. 2023. Sharegpt4v: Improving large multi-modal models with better captions. *arXiv preprint arXiv:2311.12793*.

Wenliang Dai, Junnan Li, Dongxu Li, Anthony Meng Huat Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale Fung, and Steven Hoi. 2023. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Preprint*, arXiv:2305.06500.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen tau Yih, Luke Zettlemoyer, and Mike Lewis. 2023. Incoder: A generative model for code infilling and synthesis. *Preprint*, arXiv:2204.05999.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Yi Gui, Zhen Li, Yao Wan, Yemin Shi, Hongyu Zhang, Yi Su, Shaoling Dong, Xing Zhou, and Wenbin Jiang. 2024. Vision2ui: A real-world dataset with layout for code generation from ui designs. *Preprint*, arXiv:2404.06369.

Daya Guo, Canwen Xu, Nan Duan, Jian Yin, and Julian McAuley. 2023. Longcoder: a long-range pre-trained language model for code completion. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. *Preprint*, arXiv:2401.14196.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxuan Zhang, Juanzi Li, Bin Xu, Yuxiao Dong, Ming Ding, and Jie Tang. 2023. Cogagent: A visual language model for gui agents. *Preprint*, arXiv:2312.08914.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *Preprint*, arXiv:2310.06825.

Hugo Laurençon, Lucile Saulnier, Léo Tronchon, Stas Bekman, Amanpreet Singh, Anton Lozhkov, Thomas Wang, Siddharth Karamcheti, Alexander M. Rush, Douwe Kiela, Matthieu Cord, and Victor Sanh. 2023. Obelics: An open web-scale filtered dataset of interleaved image-text documents. *Preprint*, arXiv:2306.16527.

Hugo Laurençon, Léo Tronchon, and Victor Sanh. 2024. Unlocking the conversion of web screenshots into html code with the websight dataset. *Preprint*, arXiv:2403.09029.

Dongxu Li, Junnan Li, Hung Le, Guangsen Wang, Silvio Savarese, and Steven C.H. Hoi. 2023a. LAVIS: A one-stop library for language-vision intelligence. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 31–41, Toronto, Canada. Association for Computational Linguistics.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023b. Blip-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. 2022. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 12888–12900. PMLR.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri

Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023c. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. 2023a. Improved baselines with visual instruction tuning.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. 2023b. Visual instruction tuning. *Preprint*, arXiv:2304.08485.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. DoRA: Weight-decomposed low-rank adaptation.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. *Preprint*, arXiv:1711.05101.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii, Nii Osae Osae Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov, Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2024. Starcoder 2 and the stack v2: The next generation. *Preprint*, arXiv:2402.19173.

Minh Huynh Nguyen, Nghi D. Q. Bui, Truong Son Hy, Long Tran-Thanh, and Tien N. Nguyen. 2023. Hierarchynet: Learning to summarize source code with heterogeneous representations. *Preprint*, arXiv:2205.15479.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. A conversational paradigm for program synthesis. *arXiv preprint*.

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. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*.

Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *Preprint*, arXiv:2204.06125.

Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup. *Preprint*, arXiv:1905.13750.

Robin Rombach, A. Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. 2021. High-resolution image synthesis with latent diffusion models. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 10674–10685.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Mehul P. Sampat, Zhou Wang, Shalini Gupta, Alan Conrad Bovik, and Mia K. Markey. 2009. Complex wavelet structural similarity: A new image similarity index. *IEEE Transactions on Image Processing*, 18(11):2385–2401.

Ensheng Shi, Yanlin Wang, Lun Du, Hongyu Zhang, Shi Han, Dongmei Zhang, and Hongbin Sun. 2021. CAST: Enhancing code summarization with hierarchical splitting and reconstruction of abstract syntax trees. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4053–4062, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Chenglei Si, Yanzhe Zhang, Zhengyuan Yang, Ruibo Liu, and Diyi Yang. 2024. Design2code: How far are we from automating front-end engineering? *Preprint*, arXiv:2403.03163.

Zian Su, Xiangzhe Xu, Ziyang Huang, Zhuo Zhang, Yapeng Ye, Jianjun Huang, and Xiangyu Zhang. 2024. Codeart: Better code models by attention regularization when symbols are lacking. *Proc. ACM Softw. Eng.*, 1(FSE).

Manan Suri, Puneet Mathur, Franck Dernoncourt, Rajiv Jain, Vlad I Morariu, Ramit Sawhney, Preslav Nakov, and Dinesh Manocha. 2024. DocEdit-v2: Document structure editing via multimodal LLM grounding. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 15485–15505, Miami, Florida, USA. Association for Computational Linguistics.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *Preprint*, arXiv:2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.

vikhyat. 2024. Moondream: tiny vision language model.

Haoran Wei, Lingyu Kong, Jinyue Chen, Liang Zhao, Zheng Ge, Jinrong Yang, Jianjian Sun, Chunrui Han, and Xiangyu Zhang. 2023. Vary: Scaling up the vision vocabulary for large vision-language models. *arXiv preprint arXiv:2312.06109*.

Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1480–1489, San Diego, California. Association for Computational Linguistics.

X. Zhai, B. Mustafa, A. Kolesnikov, and L. Beyer. 2023. Sigmoid loss for language image pre-training. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 11941–11952, Los Alamitos, CA, USA. IEEE Computer Society.

A Appendix

A.1 Mutation Rules

Table 7 shows the mutation rules we used to mutate the HTML code and create the contrastive learning dataset. Both HTML code and the CSS styles for each element are mutated according to the failure types in our manual analysis

For CSS styles, we mutate the properties of 1) color, 2) size, 3) margin, 4) font size, 5) type of the element bounding box (display), and 6) positioning of each element. Column "Specification" in Table 7 shows the details of the valid values for each property. For HTML codes, we randomly duplicate one HTML element excluding the ones that will cause render failures (i.e., `<head>`, `<header>`, `<html>`, `<body>`).

A.2 Illustrating Example

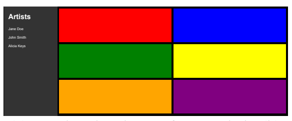
Figure 7 illustrates the second challenge, i.e., learning the fine differences and details of the visual input. Figure 7(a) and Figure 7(c) are two highly similar but different UI images of rendered webpages, i.e., the colors and text are identical, but the widths of the columns are slightly different. VLM-WebSight (Laurençon et al., 2024), a state-of-the-art MLLM for webpage image to HTML code generation, fails to capture such small differences; thus, it generates identical HTML and CSS code (Figure 7(b)) for the different UI images: Figure 7(a) and Figure 7(c). The model fails to generate `1fr` `2fr` (highlighted in code segments (d) with red background) for screenshot (c). In this case, VLM-WebSight's vision model fails to recognize the visual difference, and its text model is unable to use the encoded visual information to produce accurate textual output.

A.3 Tuning the Integration of WAFFLE's Structure-Aware Attention.

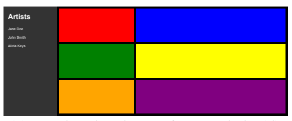
WAFFLE applies the structure-aware attention on the attention layer in MLLM's decoder. To study the portion of attention heads that use structure-aware attention, we fine-tuned VLM-WebSight on a subset of our training dataset (40,000 pairs of

Class	Failure Type	Specification
CSS	Color	Random Color in Range [`#000000`, `#FFFFFF`]
	Size	Random Size in [0, 500] pixels
	Margin	Random Size in [0, 100] pixels
	Font	Random Size in [0, 40] pixels
	Display	Random Keyword for `text-align`, `display`, `flex-direction`, and `justify-content`
	Position	Random Keyword for `border-radius`, `position`, `top`, and `right`
HTML	Structure	Duplication of a Random HTML Element, excluding `<head>`, `<header>`, `<html>`, `<body>`

Table 7: Specification for Mutation Rules to construct the Contrastive dataset.



(a) Rendered webpage from code in (b)



(c) Rendered webpage from code in (d)

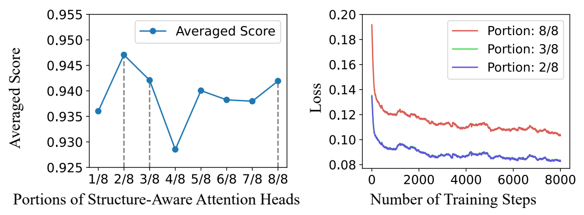
```
#grid {
  display: grid;
  grid-template-columns: 1fr 1fr;
}
...
<div id="grid">
  <div style="background-color: red;">
  </div>
  <div style="background-color: blue;">
  </div>
  <div style="background-color: green;">
  </div>
  <div style="background-color: yellow;">
  </div>
  <div style="background-color: orange;">
  </div>
  <div style="background-color: purple;">
  </div>
</div>
...
```

(b) Snippet of HTML and CSS code

```
#grid {
  display: grid;
  grid-template-columns: 1fr 2fr;
}
...
<div id="grid">
  <div style="background-color: red;">
  </div>
  <div style="background-color: blue;">
  </div>
  <div style="background-color: green;">
  </div>
  <div style="background-color: yellow;">
  </div>
  <div style="background-color: orange;">
  </div>
  <div style="background-color: purple;">
  </div>
</div>
...
```

(d) Small modification on CSS in (b)

Figure 7: Existing MLLM generates identical HTML and CSS code in (b) given the different webpage screenshots in (a) and (c). The MLLM fails to generate the correct `2fr` highlighted in red in (d).



(a) Validation LLEM Score (b) Training Loss

Figure 8: Illustration of the tuning process of the parameter that controls the effect of structure-aware attention. In (b), the green line almost overlaps with the blue line.

Techniques	$d(\overline{v^i}, c) \uparrow$	$sim(\overline{v^i}, c_g) \downarrow$
Standard FT	0.1224	0.9910
WAFFLE-attn	0.7590	0.6202

Table 8: Distance (d) and similarity (sim) between each averaged image embeddings $\overline{v^i}$ with the corresponding centroid c of the group of mutants, with Moondream2 backbone.

HTML code and webpage screenshots). We set the portion of attention heads using structure-aware attention from $\frac{1}{8}$ to all, incrementing each setting by $\frac{1}{8}$. All models are trained with a batch size of 4, and a learning rate of $2e^{-5}$. The models are then evaluated on the validation dataset. We use two metrics to decide the final hyper-parameter: averaged LLEM score and training loss.

Figure 8 (a) shows the averaged LLEM score. Applying structure-aware attention on $\frac{2}{8}$, $\frac{3}{8}$, and $\frac{8}{8}$ of the attention heads results in the top three validation scores. We also consider their training loss in Figure 8 (b). Although applying structure-aware attention on all (i.e., $\frac{8}{8}$) of the attention heads yields a high LLEM score, it also results in a high training loss, likely due to the regular attention heads retaining some prior knowledge during pre-training. In contrast, $\frac{2}{8}$, $\frac{3}{8}$ show similar and lower training losses. Combining these results, we select $\frac{2}{8}$ (i.e., $\frac{1}{4}$) as the final hyper-parameter controlling the portion of attention heads that use structure-aware attention.

A.4 Additional Ablation of Contrastive Learning's effect

Capturing subtle visual differences. Using the same computed embeddings, we compute the averaged distances and similarities between each image embedding the centroid of its corresponding group of mutants. Formally, for each group of mutants, G, consisting of image embedding $\{\overline{v^i}\}$, $v^i \in G$, the centroid of the image embeddings is computed as: $c = \sum_{i=1}^{|G|} \overline{v^i}$. Table 8 shows the distance and cosine similarities between the image embeddings. The average distance between each image embedding with its respective centroid computed by the WAFFLE-attn is 0.7590, greatly surpassing the average distance of 0.1224 computed by Standard FT. Likewise, the cosine similarity between the image embeddings is much lower for WAFFLE-attn (0.6202 vs. 0.9910), showing WAFFLE-attn's better ability to distinguish between the images.

Figure 6 also shows that Standard FT encodes the four different images almost the same in the latent space (i.e., the four red circles are overlapped in (a)), while WAFFLE-attn is able to encode them differently.

A.5 Attention Visualization

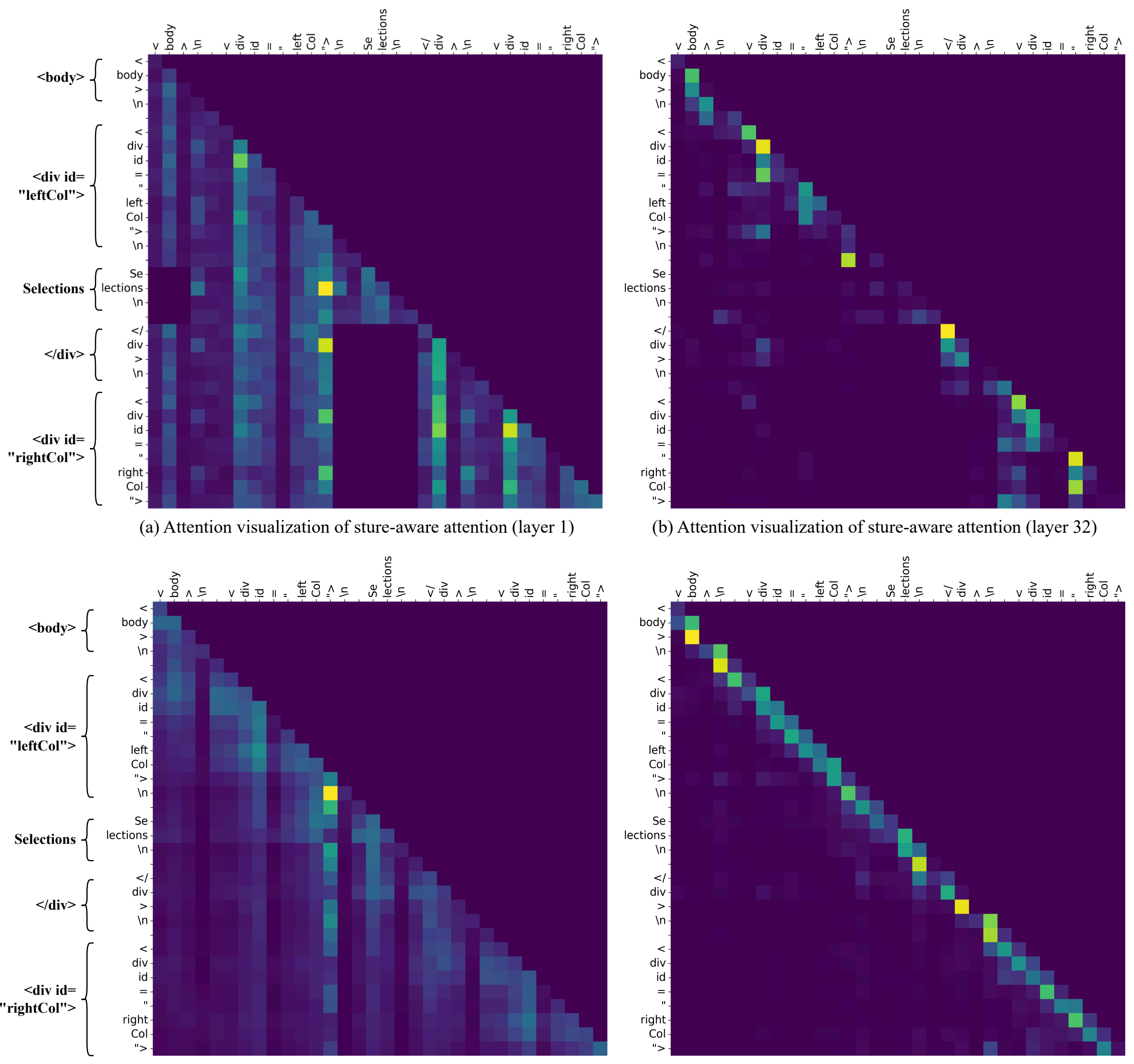
The visualization of attention weights in Figure 9 provides insight into the behavior of structure-aware attention and standard attention mechanisms. Four representative examples are selected from different attention heads, each demonstrating distinct characteristics.

The attention in (a) exhibits a sparse and discrete distribution while effectively leveraging HTML domain knowledge. Notably, elements such as `<div id = "rightCol">` do not attend to the children of their siblings, allowing for a more efficient allocation of attention resources. This pattern suggests that structure-aware attention successfully encodes hierarchical relationships, prioritizing relevant dependencies.

The attention in (b) demonstrates a diagonalized pattern, aligning with the common attention sink phenomenon observed in deeper layers. The attention weights primarily concentrate along the diagonals, indicating that elements predominantly attend to themselves or closely positioned tokens. Despite this, the model retains structural awareness, as elements do not indiscriminately attend to siblings' children. This behavior suggests that structure-aware attention maintains domain knowledge while refining local relationships in later layers.

The attention in (c) presents a more scattered distribution, lacking clear structural constraints. The model attends to multiple elements, including those that do not directly influence the rendered appearance of the current node. A notable observation is that HTML elements tend to ignore tokens such as `\n` and =, suggesting a preference for more informative tokens.

The attention in (d) also exhibits a diagonalized



(a) Attention visualization of sture-aware attention (layer 1)

(b) Attention visualization of sture-aware attention (layer 32)

(c) Attention visualization of normal attention (layer 1)

(d) Attention visualization of normal attention (layer 32)

Figure 9: Selected Attention Weight Visualization of WAFFLE on a simplified example.

pattern, indicative of attention sink behavior, where tokens predominantly attend to closely positioned elements. This pattern resembles the behavior observed in (b), yet lacks the structured constraints of structure-aware attention. The similarity between the two suggests that deeper layers across both models tend to favor local attention.

A.6 WAFFLE's Computational Overhead

We discuss the computational overhead of WAFFLE in preprocessing, training, and inference separately.

Preprocessing The only additional one-time preprocessing cost incurred for each sample and tokenizer is the construction cost of the structure-aware attention masks. The construction process for these structure-aware attention masks exhibits a complexity of $O(B \cdot L)$, where B denotes the buffer length in our specialized HTML parser used for tracking HTML tag tokens, and L represents the overall token length. This compares to the $O(L)$ complexity for constructing a standard causal mask. Notably, the buffer B solely contains tokens of currently incomplete HTML tags, rendering its size small relative to the full token length L. Furthermore, this preprocessing can be executed on a CPU and is exceptionally lightweight, as it does not involve matrix multiplication operations. Once these attention masks are generated, they are reused throughout the entire training process without incurring further overhead, making this initial computational investment negligible when amortized across the full training cycle.

Training The computational demands during the training phase of WAFFLE are influenced by two primary components: Structure-Aware Attention and Contrastive Learning.

The introduction of Structure-Aware Attention results in **no additional computational cost** during training. This is because the only modification pertains to the values within the attention mask, while the total number of operations remains the same to that of standard attention mechanisms.

Our implementation of Contrastive Learning, the image embeddings are computed once and subsequently reused for both standard fine-tuning tasks and the contrastive learning objectives. The main additional computation arises from generating text embeddings, which requires an extra forward pass through the decoder. Other minor computational elements, such as similarity calculations and the computation of the contrastive loss, are negligible in comparison to the overall training expenditure.

In our experimental configuration, utilizing 4x NVIDIA A100 GPUs with WebSight-VLM, standard fine-tuning necessitates approximately **26 hours**. In contrast, training with WAFFLE requires about **34 hours**, which constitutes a **31% increase**. It is important to emphasize that this is a **one-time training cost** and **does not adversely affect inference speed**. We consider this increase a worthwhile investment, given the significant performance improvements achieved by WAFFLE.

Inference During inference, the additional computational cost introduced by WAFFLE stems from parsing the generated HTML and constructing the structure-aware attention mask for newly generated tokens. This construction process is identical to the one employed during preprocessing and, similarly, remains lightweight in comparison to the model's other operational demands.

In conclusion, WAFFLE introduces a limited computational overhead during training and imposes minimal computation overhead during inference and preprocessing.

A.7 Analysis of Failure Examples

To provide a balanced evaluation of WAFFLE's capabilities, this section presents an analysis of selected failure cases. We discuss two sets of illustrative examples, one from each test set (Design2Code and WebSight-Test), where the VLM-WebSight model fine-tuned by WAFFLE exhibited deviations from the ground truth.

Design2Code-42 The first case is from the Design2Code test set, example Design2Code-42, shown in Figure 10. The ground truth for this example is depicted in Figure 10 (a), while the output generated by our WAFFLE-enhanced VLM-WebSight model is shown in Figure 10 (b). For a comparative perspective, the result from VLM-WebSight under standard fine-tuning (SFT) is provided in Figure 10 (c).

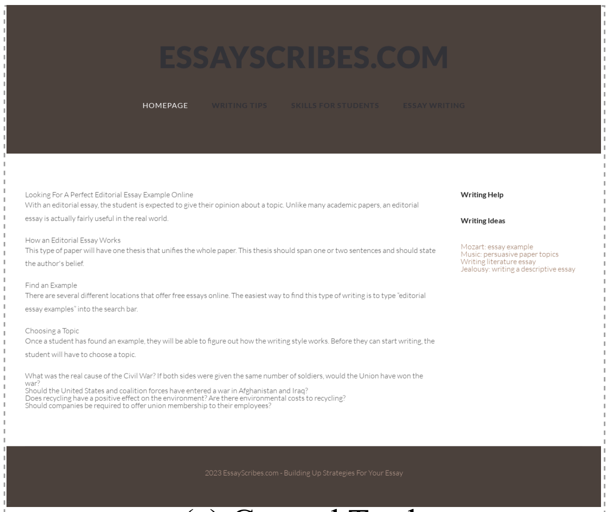
Upon examining the output from WAFFLE, several discrepancies were identified when compared to the ground truth. A primary challenge appears to be the accurate recognition and reproduction of all textual elements. For instance, there was a noticeable difference in the domain name; the ground truth specifies "ESSAYSCRIBES.COM", whereas the generated version incorrectly rendered it as "EASYSCRIBES.COM". Further textual inaccuracies included differences in the primary heading or description on the first line, where "Essential Example" was expected but variations occurred, and section headings were altered, such as "How an Editorial Essay Works" in the ground truth becoming "How The Editor Writes an Essay" in the generated HTML.

Styling inconsistencies also emerged. The navigation menu in the WAFFLE-generated version exhibited different spacing and font treatment compared to the original design. Additionally, the navigation items themselves were simplified and featured slightly different labeling. Other minor errors were observed, such as in the footer area, where, although the copyright text was consistent, its formatting and color scheme diverged from the ground truth.

In summary, while VLM-WebSight fine-tuned by WAFFLE successfully captured the main structure of this real-world UI from the Design2Code test set, several minor errors in text rendering and styling persist. These indicate areas for further improvement in UI-to-HTML generation. It is pertinent to note, however, that these errors are considerably less severe than those observed in the output from the VLM-WebSight model under standard fine-tuning. The SFT version exhibited more drastic failures, with significant mistakes in color fidelity, overall structure, and text extraction.

WebSight-Test-45 The second failure case analyzed is WebSight-Test-45. The ground truth image for this example is available as Figure 11 (a), and the corresponding page generated by VLM-WebSight fine-tuned with WAFFLE is shown in Figure 11 (b).

In this instance, the generated output aligns well with the ground truth in terms of most visual elements. However, a key error was identified in the



ESSAYSCRIBES.COM

HOMEPAGE WRITING TIPS SKILLS FOR STUDENTS ESSAY WRITING

Looking For A Perfect Editorial Essay Example Online
With an editorial essay, the student is expected to give their opinion about a topic. Unlike many academic papers, an editorial essay is actually fairly useful in the real world.

How an Editorial Essay Works
This type of paper will have one thesis that unifies the whole paper. This thesis should span one or two sentences and should state the author's belief.

Find an Example
There are several different locations that offer free essays online. The easiest way to find this type of writing is to type "editorial essay examples" into the search bar.

Choosing a Topic
Once a student has found an example, they will be able to figure out how the writing style works. Before they can start writing, the student will have to choose a topic.

What was the real cause of the Civil War? If both sides were given the same number of soldiers, would the Union have won the war?
Should the United States and coalition forces have entered a war in Afghanistan and Iraq?
Does recycling have a positive effect on the environment? Are there environmental costs to recycling?
Should companies be required to offer union membership to their employees?

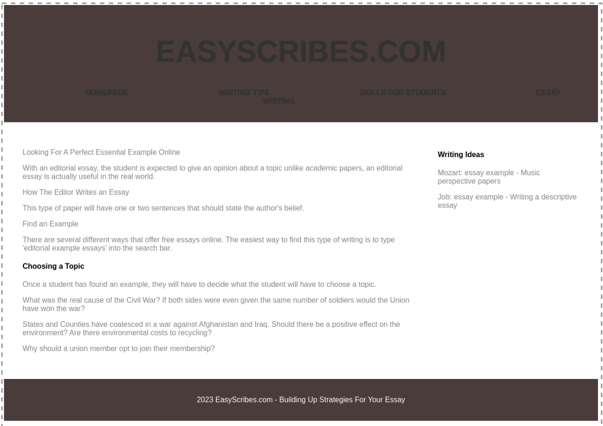
Writing Help

Writing Ideas

Mozart: essay example
Music: persuasive paper topics
Writing literature essay
Jealousy: writing a descriptive essay

2023 EssayScribes.com - Building Up Strategies For Your Essay

(a) Ground Truth



EASYSCRIBES.COM

HOMEPAGE WRITING TIPS SKILLS FOR STUDENTS ESSAY WRITING

Looking For A Perfect Essential Example Online

With an editorial essay, the student is expected to give an opinion about a topic unlike academic papers, an editorial essay is actually useful in the real world.

How The Editor Writes an Essay

This type of paper will have one or two sentences that should state the author's belief.

Find an Example

There are several different ways that offer free essays online. The easiest way to find this type of writing is to type 'editorial example essays' into the search bar.

Choosing a Topic

Once a student has found an example, they will have to decide what the student will have to choose a topic.

What was the real cause of the Civil War? If both sides were even given the same number of soldiers would the Union have won the war?

States and Counties have coalesced in a war against Afghanistan and Iraq. Should there be a positive effect on the environment? Are there environmental costs to recycling?

Why should a union member opt to join their membership?

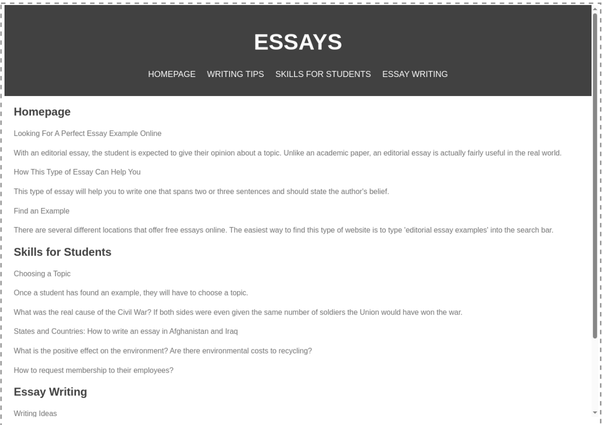
Writing Ideas

Mozart: essay example - Music perspective papers

Job: essay example - Writing a descriptive essay

2023 EasyScribes.com - Building Up Strategies For Your Essay

(b) Generation-Waffle



ESSAYS

HOMEPAGE WRITING TIPS SKILLS FOR STUDENTS ESSAY WRITING

Homepage

Looking For A Perfect Essay Example Online

With an editorial essay, the student is expected to give their opinion about a topic. Unlike an academic paper, an editorial essay is actually fairly useful in the real world.

How This Type of Essay Can Help You

This type of essay will help you to write one that spans two or three sentences and should state the author's belief.

Find an Example

There are several different locations that offer free essays online. The easiest way to find this type of website is to type 'editorial essay examples' into the search bar.

Skills for Students

Choosing a Topic

Once a student has found an example, they will have to choose a topic.

What was the real cause of the Civil War? If both sides were even given the same number of soldiers the Union would have won the war.

States and Countries: How to write an essay in Afghanistan and Iraq

What is the positive effect on the environment? Are there environmental costs to recycling?

How to request membership to their employees?

Essay Writing

Writing Ideas

(c) Generation-SFT

Figure 10: Figures of Design2Code task 42.

positioning of the search bar. In the generated version, the search bar is incorrectly placed at the bottom of the property cards. In contrast, the ground truth image clearly shows the search bar embedded within the middle property card. Although our WAFFLE-enhanced model correctly identified that the search bar should be horizontally centered within the UI, it failed to accurately replicate its embedded placement within the designated property card.

This specific layout mistake could potentially be rectified by applying CSS rules such as `position:absolute; top:50%; left:50%; transform:translate(-50%, -50%)` to the search button's styling. The occurrence of such an error suggests that incorporating a refinement step for the generated results could be a promising avenue for future research, potentially allowing for corrections of these types of spatial misplacements.

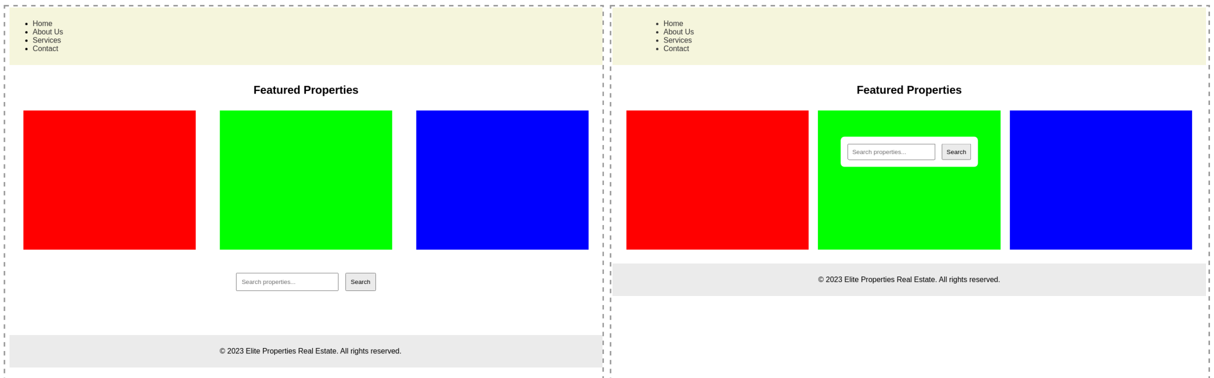
The analysis of these failure cases offers insights into the current limitations of WAFFLE, particularly in areas such as precise text recognition, complex styling fidelity, and exact component placement. These observations highlight potential directions for future improvements.

A.8 Infrastructure

Our approach is implemented with the following packages: `transformers 4.41.1`, `pytorch 2.3.0`, `selenium`, `deepspeed 0.14.1`, `accelerate 0.30.1`, and `datasets 2.19.1`. The experiments are conducted on a shared computing cluster with four NVIDIA A100 GPUs.

A.9 Potential Risk and Impact

This work aims to develop a multi-modality model to help front-end developers write and understand HTML and CSS code. We assume no risk of this approach being misused.



(a) Ground Truth (b) Generation-Waffle

Figure 11: Figures of WebSight-Test task 45.