Learning to Generate Code from Images with Actor-Critic

Josh H Davis and **Zlatko-Salko Lagumdzija** and **Shelvin Pauly** and
Benjamin Wu and **Michel Temgoua** and **Davit Soselia**
University of Maryland, College Park

Abstract

This project presents a novel approach for generating HTML code from images using actor-critic fine-tuning, along with an adapted metric based on BLEU called htmlBLEU for comparing HTML code samples. The objective is to address the challenge of accurately reproducing visual elements and structure from input images to facilitate web development and design automation. We evaluate our proposed approach against transformer and CNN-based baselines and show improved performance against all, with additional results showing robustness of our advantage over varying sample complexity. The findings advance automated code generation towards potential applications in web development and design automation, enabling efficient conversion of images into functional HTML code. We conclude with several observations of remaining limitations and directions for future work.

1 Introduction

In recent years, Language Models (LMs) have demonstrated remarkable ability in generating coherent text and text samples. Models such as Codex and CodeRL have shown promise in aiding software engineers in their daily work, while attention-based models have achieved success in various downstream tasks, such as image classification, segmentation, and annotation. Among them, some landmark works are Vision Transformer (ViT), SWIN, and other architectures, with some models specifically targeting document-related tasks, such as the Document Image Transformer (DiT).

This project explores the application of actor-critic fine-tuning to train a model that generates front-end code capable of reproducing an input image. The research builds upon existing literature on RL-based code generation from text specifications and extends the idea to image or multimodal contexts. The model's performance is evaluated using the modified CodeBLEU metric to attempt to better approximate the visual similarity of the output that would result in from running the generated code.

2 Related Work

In recent years, the intersection of language models and code generation has garnered significant attention in the research community. Several studies have explored the application of language models for code generation tasks, ranging from software engineering support to document-related tasks. In this section, we discuss relevant works that have paved the way for our research on generating HTML code from images using an actor-critic fine-tuning approach.

Language models, particularly transformer-based models, have demonstrated impressive capabilities in generating coherent text and code. Notably, the (Chen et al., 2021), developed by OpenAI, has shown promise in assisting software engineers with code generation. Codex leverages a transformer architecture trained on a large corpus of code, allowing it to provide accurate code completions and suggestions. The success of Codex and similar models has inspired further exploration into leveraging language models for code-related tasks. Code generation has been addressed by Fried et al. (2022), which trains on publicly available repositories where regions of code have been randomly masked and moved to the end of each file, allowing bidirectional context for code infilling. The model is capable to perform zero-shot code infilling. Another method for code generation is introduced by Chen et al. (2021), in which GPT-3 is fine-tuned on publicly available code from GitHub. This approach achieves a 28.8% solving rate on the HumanEval dataset, which aims to measure the functional correctness of generated code. Robinson (2019) attempts to generate code from website wireframes using both traditional computer vision algorithms (e.g. denoising, edge detection, segmentation and OCR) and machine learning tech-

niques (e.g. ANNs, MLPs, CNNs). The study finds that deep learning approaches outperform classical computer vision approaches in visual and structural comparisons.

Reinforcement learning (RL) has been widely applied in code generation tasks, enabling models to learn from feedback and improve their performance iteratively. The (Le et al., 2022) framework proposed by Zettlemoyer et al. explores the use of RL to generate code from text specifications. By framing code generation as a sequential decision-making problem, CodeRL utilizes an actor-critic architecture to optimize code generation policies. This work serves as an inspiration for our actor-critic fine-tuning approach in the context of image-to-code generation.

(Dosovitskiy et al., 2020) revolutionized image classification by adapting the transformer architecture to process visual data. ViT showed remarkable performance in image recognition tasks, demonstrating the potential of transformer models for visual understanding. Inspired by ViT's success, our research incorporates visual processing capabilities into the actor-critic fine-tuning approach, aiming to generate HTML code that reproduces input images accurately.(Li et al., 2022) focuses specifically on document-related tasks and addresses the challenge of understanding and processing diverse document layouts. DiT employs a transformer-based architecture and leverages self-attention mechanisms to extract hierarchical representations of document images. Given the similarities between document layout understanding and image-to-code generation, we draw inspiration from DiT's architecture and adapt it to fine-tune the DiT-GPT2 model for generating HTML code from images.

Assessing the quality and similarity of generated code compared to ground truth is crucial in code generation tasks. Traditional evaluation metrics such as BLEU (Bilingual Evaluation Understudy) have been widely used for evaluating the quality of generated text. However, code generation requires specialized metrics that consider both code functionality and structure.

In our research, we propose the HTMLBLEU metric, which combines matching of HTML DOM trees, CSS attributes, and keywords, as well as a weighted BLEU score. This metric provides a comprehensive evaluation of the similarity between generated and ground truth HTML code. These related works collectively contribute to the development of our research on generating HTML code from images using actor-critic fine-tuning. By leveraging insights from language models, RL-based code generation, vision transformers, document image understanding, and specialized evaluation metrics, we aim to advance the state-of-the-art in image-to-code generation and provide valuable tools for software engineers and web developers.

3 Methodology

In this section, we outline the methodology employed in our research on generating HTML code from images using actor-critic fine-tuning. We begin by defining the problem of generating HTML code from images. The objective is to develop a model that can accurately reproduce the structure and visual elements of an input image by generating corresponding HTML code. The generated code should reflect the positioning, size, color, and other attributes present in the image, while ensuring functional similarity with the original image.

3.1 Datasets

The first step of this project, preparing data, has already been carried out prior to this class. For the project, we generated examples of HTML code to generate combinations of various simple shapes. The dataset has 25,000 samples for training and 3,000 samples for testing.

The collected data undergoes preprocessing to ensure consistency and compatibility with our model. We parse the HTML code, extract the relevant visual and structural information, and transform the images into a suitable format for training the model.

3.2 CodeBLEU-based validation metric

(Ren et al., 2020) provides a new automatic evaluation metric for code synthesis that addresses the limitations of commonly used metrics like BLEU and perfect accuracy. It incorporates the strengths of BLEU in n-gram matching and further injects code syntax via abstract syntax trees (AST) and code semantics via data-flow. CodeBLEU has been tested in three code synthesis tasks, including text-to-code synthesis, code translation, and code refinement, and has shown to significantly differentiate system performance and achieve better correlation with quality scores given by programmers than BLEU. To carry out validation throughout the project, a similarity metric between the ground

truth code and generated code. CodeBLEU has already defined an evaluation metric for comparing Python code. Therefore, we adapted CodeBLEU to work for HTML, which we call HTMLBLEU. Specifically, we redefined the $Match_{ast}$ and $Match_{df}$ terms in CodeBLEU to apply to HTML code. The $Match_{ast}$ term is straightforwardly adapted to describe the alignment between HTML Document Object Models (DOMs), which are also tree structures like ASTs. $Match_{df}$, which describes the match between the flow of data through Python code, is less straightforward to adapt.

Given the number of different CSS commands we've made our metric only work with the elements that appear in our dataset. We've adatapted the idea of $Match_{df}$ to our code by creating a CSSBLEU metric for measurement of CSS similarity. CSSBLEU takes each element in the source and matches its position and size to the element which it shares the highest overlap in area and color. Namely we take their area of intersection and divide by the average of their individual areas. We then multiply that average by the distance between the two element colors (1 being the same color and 0 being completely different color RGB values). This gives us an estimate of the actual mean squared error and IOU between the two divs given it calculates both the overlapped area and the multiplies the differences in color as well. Note that this method will give us a 1 when the two shapes overlap in size and color and a 0 when there is no overlap.

While HTMLBLEU provides a strong metric of similarity between HTML codes for the purposes of evaluating our model, for the training process, we had to use something that is differentiable, which HTMLBLEU won't be. This is the motivation for using an actor-critic method, as used by CodeRL, in which the critic learns a mean squared error (MSE) estimation.

3.3 Baseline model

In addition to the language model approach, we also explored baselines using convolutional neural networks (CNNs) inspired by the Sketch2code model proposed by Robinson (2019). The CNN component of our baseline models plays a crucial role in extracting effective and meaningful features from the input images. By leveraging the inherent capability of CNNs in image processing and feature extraction, we aim to capture relevant visual information that can aid in the generation of accurate and contextually appropriate HTML code.

To ensure the generated code is coherent and syntactically correct for the CNN baseline, we utilize an autoregressive model called BART (Bidirectional and AutoRegressive Transformers). The BART model is employed in conjunction with the CNN component to generate code sequences by predicting the next token based on the previously generated tokens. This autoregressive approach allows us to maintain the logical structure and integrity of the generated HTML code.

3.4 Actor-Critic RL Tuning

We finetune our pretrained code generator baseline model we expand CodeRL (Le et al., 2022) and shown in Figure 1. In this setup a baseline CE finetuned model serves

Pairs of generated samples and reference samples are input into the model as a joint string in the format "generated code\nGround:ground code". The IoU bucket label is subsequently employed. Once the critic model has been readied, we can proceed to fine-tune the actor baseline using the critic model.

We then use the critic model to generate intermediate outputs for each prediction, and ground code samples in the training set. We also create a mask to only use the values related to predicted code tokens in the tuning loss calculation. We then apply softmax to each token's corresponding output and select values of the IoU ground truth bucket. The resulting vector is then multiplied by the corresponding assigned reward. We assign negative rewards of -1, -0.7, and -0.3 to classes 0,1,2 and positive reward of 1 for class 3. The resulting vector is used to scale the original loss during the finetuning training run.

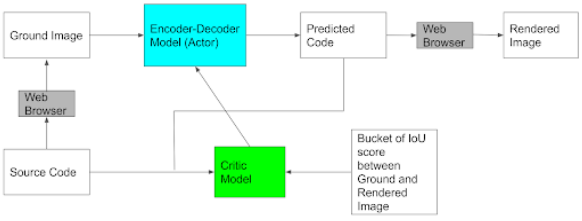

Figure 1: Actor-Critic Finetuning System

4 Experimental Setup

We carry out a series of experiments comparing the various baseline model configurations to our actor-critic fine-tuned model. We primarily use mean IoU, MSE and colorbind MSE to comapare these approaches. Additionally, for the ViT and DiT baselines, we performed additional experiments considering the BLEU and htmlBLEU metrics as well as element counts. To evaluate the critic model, we measure confusion scores between true and predicted class among four ranges of IoU scores: very low (0-23), low (23-42), high (42-77), and very high (77+). For the transformer models we trained until hitting a loss plateau, about ten epochs. For CNN, this came later, at forty epochs. We performed actor-critic fine-tuning for one epoch. We also perform an experiment varying the number of `<div>` elements and recording changes in IoU score to examine the relationship between sample complexity and model performance for the ViT, DiT, and DiT-AC (actor-critic fine-tuned) models. The numbers of samples trained on for each element count were equal.

5 Results

The results of our primary experiment comparing the various baseline models to our actor-critic fine-tuned model are recorded in Table 1. As hypothesized, the actor-critic fine-tuned model outperforms all of the baselines examined to a significant degree. It improves over the baseline version of DiT-GPT2 (the model we chose to fine-tune), by 21 to 94% depending on the metric. There are some interesting results to note relating to the CNN baseline specifically. Overall, the CNN model performs quite poorly, as confirmed by visual spot-checking of the outputs and very low mean IoU. However, the mean MSE and colorblind MSE for the CNN models are actually better than ViT, despite IoU being much better for ViT and visual spot checking favoring ViT. This result suggests that MSE is not capturing the complete picture of model performance for this task, further motivating the use of specialized measures like our htmlBLEU. Additionally, we observe that with additional epochs (2 to 40), CNN mean IoU worsens but mean colorblind MSE improves.

Figure 2 shows the change in IoU score as we increase the number of `<div>` elements in the input samples. Increasing the number of elements, and therefore the complexity of the sample, causes IoU score to drop for all the models. The relationship between models, however, remains constant, with our actor-critic approach consistently winning out. The gap between our approach and the baselines is wider at higher element counts, indicating greater robustness to sample complexity.

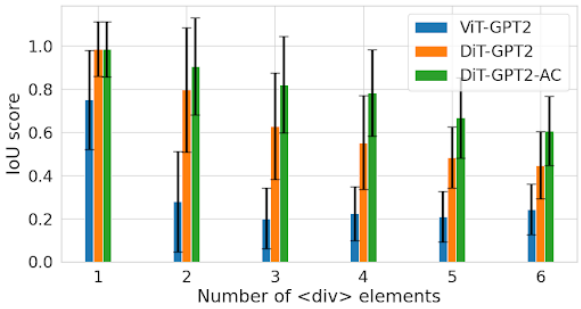

Figure 2: Performance of the number of elements increases

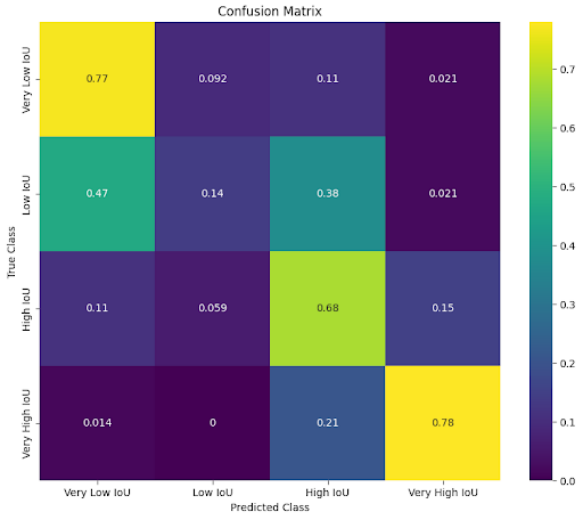

Figure 3: Confusion Matrix of the Critic Model

Figure 3 shows the confusion matrix for the critic model's IoU prediction. As expected, the critic model predicts more extreme cases, very high and very low IoU, with relative ease, but struggles with more borderline cases. In particular, it most commonly mistook low samples for very low samples, and other low samples as high samples. Some very high samples were also mistaken for high samples. This generally suggests some difficulty with "slightly-off" cases that calls for further examination to remedy.

Figure 4 shows outputs of the ViT, DiT, and actor-critic tuned (RL) models for three sample inputs (the top row). These outputs are rendered from the HTML code generated by each model. From a visual comparison of these representative

	ViT+GPT2	DiT+GPT2	CNN-BART-2	CNN-BART-40	DiT+GPT2-AC
mean IoU ↑	0.31	0.64	0.0540	0.0330	**0.78**
mean MSE ↓	19.63	12.25	14.5	14.5	**9.0**
mean MSE (CB) ↓	0.15	0.07	0.127	0.102	**0.036**

Table 1: Visual metrics of code generation from screenshot. We show results for the CNN model at two epochs and forty. The MSE (CB) entry is colorblind MSE. CNN-BART-2 is trained for two epochs and chosen due to the lowest validation loss, while CNN-BART-40 is trained for forty.

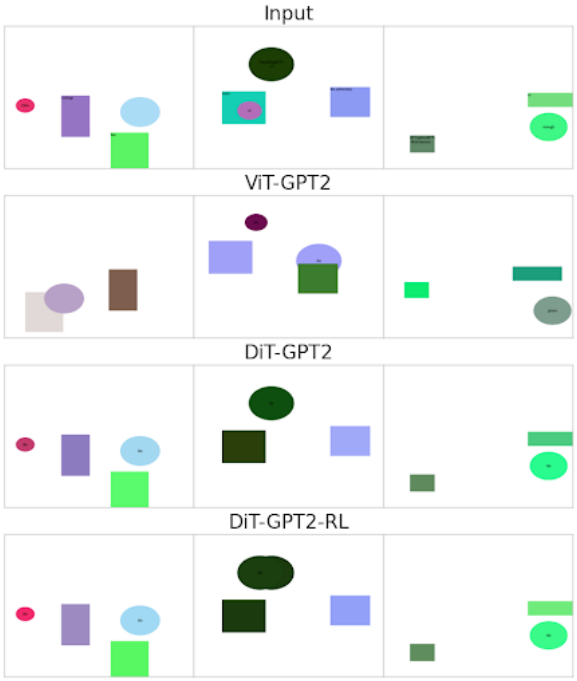

Figure 4: Sample Visual Comparison

samples we can make three observations. First, the gap between ViT and DiT is made more clear, with DiT as the obvious winner. Second, the gap between actor-critic and DiT becomes less clear, with the benefit of actor-critic looking relatively minor. Qualitatively, we observe that our approach tends to capture the colors of elements slightly better, while sometimes making mistakes with shape layout as with the ellipse rendered as two smaller ellipses in the middle column. Finally, we observe that no model is able to handle overlapping shapes well, with the purple ellipse on teal square not correctly generated by any model compared.

Finally, we conclude our results overview with a more detailed comparison of ViT and DiT which includes the BLEU and htmlBLEU metrics, as shown in Table 2. As predicted by Figure 4, DiT wins by a large margin for most measures. Interestingly, the difference between the models by htmlBLEU is much smaller. This could be because this metric emphasizes more complex features of samples like overlapping elements that all approaches struggled with. Taken with Figure 4, the results from Table 2 strongly motivate our choice to use DiT over ViT as the baseline to fine-tune with actor-critic.

6 Limitations & Future Work

Despite the promising results, it is important to acknowledge certain limitations of the study. Firstly, the evaluation was focused on a specific set of web page designs and may not generalize to all possible design variations. Future research could explore a more diverse range of input images to further validate the performance of the fine-tuned model.

Secondly, the training and evaluation datasets primarily consisted of synthetic examples and scraped login forms, which may not fully capture the complexity and variability of real-world web page designs. Incorporating larger and more diverse datasets from different sources could enhance the model's generalization capabilities.

Lastly, the study primarily focused on the visual aspects of the generated code and did not extensively explore its functional correctness. Further investigations could consider incorporating additional validation techniques to ensure the generated code produces the expected functionality and adheres to web standards.

Despite these limitations, the results obtained in this study provide valuable insights into the potential of actor-critic fine-tuning for improving the generation of HTML code from input images. The findings pave the way for further advancements in leveraging reinforcement learning techniques to enhance code generation in the context of web development.

7 Conclusion

We have proposed and evaluated a novel method for generating HTML code to replicate a visual specification in the form of an input image. We have also proposed and evaluated a custom performance met-

	ViT+GPT2	DiT+GPT2
BLEU ↑	0.65 ± 0.08	**0.74 ± 0.09**
htmlBLEU ↑	0.62 ± 0.13	**0.69 ± 0.14**
IoU ↑	0.31 ± 0.25	**0.64 ± 0.27**
MSE ↓	19.63 ± 11.59	**12.25 ± 8.83**
MSE (Single Channel) ↓	0.15 ± 0.09	**0.07 ± 0.06**
Element Counts ↑	**0.97 ± 0.16**	0.97 ± 0.18

Table 2: Evaluation of code generation from screenshot. The error bounds are standard deviations from the means listed.

ric for HTML generation tasks, htmlBLEU, which can accurately compare a ground truth and generated HTML code sample. Our evaluation shows the improved performance of our method over transformer and CNN-based baselines, even with varying sample complexities. However, visual evaluation suggests there are still aspects of image to code generation our model fails to capture, particularly overlapping elements. We end with suggestions for future directions to extend the robustness and applicability of our proposed method.

References

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2020. An image is worth 16x16 words: Transformers for image recognition at scale.

Daniel Fried, Armen Aghajanyan, Jessy Lin, Sida Wang, Eric Wallace, Freda Shi, Ruiqi Zhong, Wen-tau Yih, Luke Zettlemoyer, and Mike Lewis. 2022. Incoder: A generative model for code infilling and synthesis. *arXiv preprint arXiv:2204.05999*.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven CH Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *arXiv preprint arXiv:2207.01780*.

Junlong Li, Yiheng Xu, Tengchao Lv, Lei Cui, Cha Zhang, and Furu Wei. 2022. Dit: Self-supervised pre-training for document image transformer.

Shuo Ren, Daya Guo, Shuai Lu, Long Zhou, Shujie Liu, Duyu Tang, Neel Sundaresan, Ming Zhou, Ambrosio Blanco, and Shuai Ma. 2020. Codebleu: a method for automatic evaluation of code synthesis. *arXiv preprint arXiv:2009.10297*.

Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup.