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

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 Research Question(s)/Hypothesis

The project aims to address the following question: *"How can reinforcement learning, and specifically the actor-critic method, be used to train a model which generates code (HTML, CSS) that reproduces an input image?"* Existing literature has shown how to use RL in an actor-critic setting to learn to generate Python code from a text specification. We plan to extend this idea to an image and potentially multi-modal context, in which the text specification becomes an image or screenshot that the output code must try to reproduce, and the image input may potentially be coupled with a text description.

3 Research Methods

In this project, we want to use reinforcement learning to reproduce an input image by training a model that will generate code (HTML, CSS) that reproduces that image.

3.1 Datasets

The first step of this project, preparing data, has already been carried out prior to this class. We have three datasets ready to use for this project:

1. Examples of simple website login forms scraped from open-source projects on GitHub.
2. Simple HTML and CSS examples scraped for the website W3Schools.
3. Synthetically generated examples of HTML code to generate combinations of various simple shapes.

Each example contains both a screenshot which the model will be trying to reproduce, and ground truth code that generates the image which will be used for validation.

3.2 CodeBLEU-based validation metric

To carry out validation throughout the project, we need to define a similarity metric between the ground truth code and generated code. CodeBLEU (Ren et al., 2020) has already defined an evaluation metric for comparing Python code. We will need to adapt CodeBLEU to work for HTML, which we will call HTMLBLEU. Specifically, we will need to redefine the $Match_{ast}$ and $Match_{df}$ terms in CodeBLEU to apply to HTML code. The $Match_{ast}$ term can be straightforwardly adapted to describe the alignment between HTML Document Object Models (DOMs), which are also tree structures like Abstract Syntax Trees (ASTs). $Match_{df}$, which describes the match between the flow of data through Python code, is less straightforward

to adapt, and we still need to do some preliminary work to determine how to adapt this to HTML, possibly through some kind of attribute matching. While HTMLBLEU should provide a strong metric of similarity between HTML codes for the purposes of evaluating our model, for the training process, we need 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 (Le et al., 2022), in which the critic learns a mean squared error (MSE) estimation.

3.3 Baseline model

Our actor-critic reinforcement learning (RL) approach will be used to tune a baseline model that can take images as input, and generate code to reproduce those images. The next step of the project, then, is to set up this baseline model. The core of the baseline model will be a pretrained large language model (LLM), like GPT-2 (Radford et al., 2019). However, GPT-2 is a text-based model, and we want to process images. To this end, we will need to employ an encoder, specifically the HuggingFace `EncoderDecoderModel` (von Platen, 2020), between the input and GPT-2. Once this baseline model is set up, we will train it on some combination of our datasets to prepare it for RL fine-tuning.

We also plan to do baseline using CNNs, similar to Robinson (2019). In case the baseline models fail to learn meaningful representations we plan to simplify the task, by instead of using connecting layers in between the vision encoders and text decoders, train them separately on simpler inputs, specifically train vision encoder to output recognized shapes from the input, since the base shapes are predefined this could be simple integer, as well as its core attributes, such as location, height, and width. Then tuning a language model to create HTML code related to the input.

3.4 Actor-Critic RL Tuning

We will finetune our pretrained code generator baseline model using the actor-critic method as demonstrated by CodeRL (Le et al., 2022). The first step of this is to prepare the critic model, since the baseline serves as the actor in this scheme. The critic model will estimate the MSE between a ground truth code and the generated code, and provide the MSE estimation to the actor model to update its weights. In order for the critic model to learn a good MSE estimation, however, it needs to learn from a true MSE calculation. We will need to prepare our datasets we will use for training the critic model by computing true MSE values from some combination of true code samples taken from the dataset and some simulated "generated" examples. The "generated" examples can either be truly generated examples taken directly from the baseline, which is how the model will work in the fine-tuning scheme, or some synthetically generated code or just another unrelated code sample from our dataset. Once the critic model is prepared, we are ready to fine-tune the actor baseline using the critic model. We will run a series of experiments comparing the performance various fine-tuning configurations (length of training, datasets sampled from and their composition) against the untuned baselines using the HTMLBLEU metric we defined in Section 3.2.

4 Related Work

A similar prompt-to-code problem was addressed in Le et al. (2022), where prompts were encoded in text rather than image form. The approach involves pretraining of a language model on public GitHub code, followed by fine-tuning on problem-solution pairs. The language model then serves as an actor which receives feedback from a critic network, following a reinforcement learning approach.

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

For the generation of embeddings given input images, Dosovitskiy et al. (2020) develops the vision transformer (ViT), which pretrains a transformer on a large dataset (14M-300M images) before fine-tuning on smaller downstream tasks. Whereas transformers underperform convolutional neural networks in lower data regimes, ViT shows that supervised pretraining on larger datasets alleviates the difference while using fewer computational resources for training. Alternatively, (Li et al., 2022) develops the document image transformer (DiT), introducing self-supervised pre-training specific to images of business documents. This approach improves performance on tasks such as document image classification, document layout analysis, and text detection for OCR.

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.

5 Timeline/Work Distribution

This section describes our planned timeline for completing major milestones, and distribution of work across members of the group.

5.1 Timeline

We will aim to meet the following milestones over the course of the project.

- 14 April: Code for baseline models written
- 21 April: We will have the baseline models trained as well as the critic model
- 5 May: Begin collecting results using actor-critic method
- 16 May: Presentation due, we will have all major experimental results for actor-critic fine-tuning collected to present
- 18 May: Final report due

5.2 Work Distribution

- Image Encoder Decoder code (see Sec. 3.3) - Set up the HuggingFace Encoder Decoder model code. (Davit)
- CodeBLEU adaptation (see Sec. 3.2) adjusting CodeBLEU to support HTML specific features. (Salko)
- Metrics implementation and testing (see Sec. 3.2) - Implement HTMLBLEU and ensure it is working. (Salko)
- HuggingFace module modification (see Sec. 3.3) - Carry out any needed modifications to the HuggingFace Encoder Decoder. (Davit and Michel)
- Data processing (see Sec. 3.1) - Carry out any data processing needed on our datasets already scraped. (Josh)
- CNN Baseline code (see Sec. 3.3) - Prepare a CNN-based baseline model to compare our approach against and use for finetuning. (Benjamin)
- Critic dataset generation (see Sec. 3.4) - Using sample pairs from model outputs and ground dataset, multiple iterations, data tracking. (Josh)
- Critic implementation (see Sec. 3.4) - Implementing critic model code. (Shelvin and Michel)
- Critic training (see Sec. 3.4) - Train the critic model to evaluate ground/predicted code pairs. (Shelvin and Michel)
- Results collection (see Sec. 3.4) - Execute experiment scripts on target systems (Josh and Michel)
- Results Processing (see Sec. 3.4) - Statistical analysis and comparison of the results. (Salko)
- Visualizations (see Sec. 3.4) - Visualizations for interpreting the results. (Josh and Benjamin)
- Human Evaluation (see Sec. 3.4) - Develop framework for human evaluation. (Davit, with evaluation done by all)

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.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

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.

Patrick von Platen. 2020. Transformer-based encoder-decoder models.