Automatic Front-end Code Generation from image Via Multi-Head Attention

Zhihang Zhang[1], Ye Ding[1*], Chenlin Huang[2]
[1]School of Cyberspace Security, Dongguan University of Technology, Dongguan, China;
[2]Academy of Computer Science, National University of Defense Technology, Changsha, China
*Corresponding author's email: dingye@dgut.edu.cn

2023 4th International Conference on Computer Engineering and Application (ICCEA) | 979-8-3503-4754-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICCEA58433.2023.10135462

Abstract. Code generation from Graphical User Interface (GUI) screenshots is a challenging task in machine learning. Existing methods (e.g., Pix2code) can handle simple datasets well but struggle with complex datasets requiring hundreds of code tokens. This paper proposes a novel method for generating front-end code based on multi-head attention. Our method uses a special technique called multi-head attention to analyze a GUI screenshot's feature vector, generate the code tokens, and link the analysis and generation processes. This architecture gives our method a significant advantage over similar models in terms of effectiveness. We conduct experiments on two types of datasets: Pix2code datasets and our own datasets. The experimental results demonstrate that our method achieves the best performance among existing methods.

Keywords-Neural networks; User interface programming; Graphical user interfaces; Scene understanding; Object recognition

I. Introduction

Developing front-end web pages is a time-consuming process that often occupies a significant portion of a developer's time. In recent years, employing machine learning technology to automatically generate front-end code from Graphical User Interface (GUI) screenshots has become an increasingly popular research topic.

However, the primary challenge in this task is the transformation from GUI screenshots to code. The pioneering work of Pix2code [2] introduced a deep learning approach based on Convolutional and Recurrent Neural Networks (CNNs and RNNs) to address this challenge. Although subsequent studies [16, 3] have improved upon this approach by incorporating attention mechanisms, the design of such mechanisms can compromise the model's portability.

We propose a new model based on the multi-head attention (MHA) [14] technique to address these limitations. Our approach enables the model to attend to information from different representation subspaces at various positions without needing specially designed attention mechanisms. Our front-end code generation model is trained on novel GUI screenshot-code datasets, which contain twice as much data as Pix2code and have an average code length of that is 1.5 times longer.

In addition to our new model, we introduce a novel evaluation methodology that more effectively measures the model's ability to process complex data. Our results demonstrate that our approach outperforms existing methods, indicating the potential of multi-head attention techniques for enhancing front-end code generation from GUI screenshots.

II. Related Work

Front-end code generation is a research field that aims to convert graphical user interfaces (GUIs) into code automatically. This field has two primary research directions: generating code from GUI screenshots and generating code from GUI design drafts.

For the first direction, several works have employed deep learning models with encoder-decoder structures composed of CNN and RNN to generate code from GUI screenshots. Pix2code [2] is an experimental project that pioneered this approach using CNN as the encoder and LSTM as the decoder. [13] improved the LSTM-based decoder by introducing Bi-LSTM to enhance model performance. [16] utilized the attention mechanism to optimize the semantic alignment between the encoder and decoder. [3] introduced GRU to refine the method. [15] proposed an evaluation method called MBLUE, which can more reasonably evaluate the code generation results. For the second direction, more detailed layer and layout information can be obtained from the GUI design drafts, but effectively handling and understanding this information is a crucial challenge. [10] proposes a method to encode layout information in the GUI design draft using Transformer. [11] proposed a method to model multi-modal information on the front page, such as images, structures, and text. [12] proposes a novel position encoding method for the position information of elements in the GUI design draft. [5] focuses on accurately identifying icons in GUI design drafts. [9] concentrates on solving the potential problem of fragmented layers in actual GUI design drafts.

III. Methods

Figure 1 presents an overview of our method. On the left side of the figure is the encoder section, which converts image I into a 1D vector I'. On the right is the decoder section, which utilizes I' and the current token sequence X_{t-1} to predict the next token x_t. The architecture can be expressed mathematically as follows:

$$X_{t-1} = (x_0, \ldots, x_{t-1}), x_i \in \mathbb{R}^K$$

$$x_t = \text{Decoder}(\text{Encoder}(I), X_{t-1})$$

where $x_0 = Token_{start}$ and $x_C = Token_{end}$ denote the special token <START> and <END>. These tokens are used to prefix and suffix the code files, respectively. C is the total number of tokens in the program. K represents the size of the token vocabulary.

In the following, we will introduce $\text{Decoder}(\cdot)$ and $\text{Encoder}(\cdot)$.

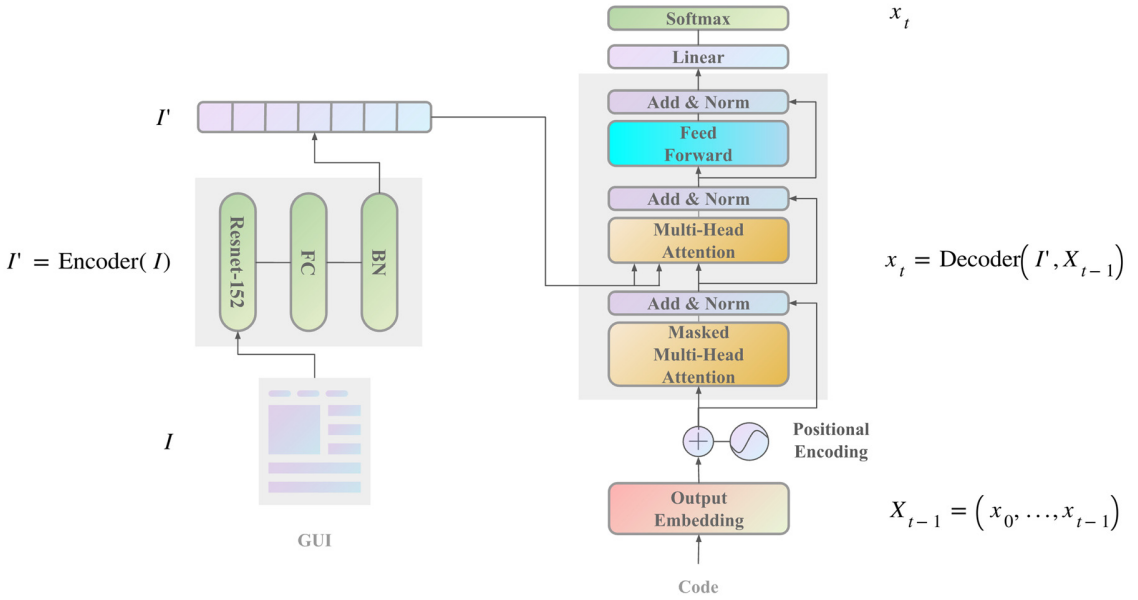

Figure 1. The overview of our method

A. Vision Encoder

The vision encoder comprises three parts: ResNet block [6], fully connected block, and batch normalization block [7]. The encoder takes one input representing a resized 512×512 image. We extract feature representations from input images using an encoder. The representations of I' are part of the input of the decoders. This extraction produces a D-dimensional representation that we refer to image note vectors.

$$I' = (a_1, \ldots, a_D), a_i \in \mathbb{R}$$

Image note vectors are employed later in the multi-head attention mechanism that selectively focuses on specific elements of image note vectors at each time step.

B. Multi-Head Attention-Based Decoder

The multi-head attention-based decoder is similar to the decoder of Transformer [4]. Unlike [4], which uses the output of multi-head attention as input of the decoder, our method employs results of ResNet-152. We chose convolutional neural networks as encoders because they are widely used for computer vision problems. One of the two multi-head attention blocks connects the encoder to the decoder, which we call "encoder-decoder attention" layers, and we call the other one "decoder attention" layer.

In the "encoder-decoder attention" layers, the queries Q_2 come from the previous decoder layers, and the keys I_K and values I_V originate from the output of the encoder. Attention layers enable the decoder to access full image note vectors at each time step. The following equations govern the working flow of the "encoder-decoder attention" layers:

$$\text{MultiHead}(Q_2, I_K, I_V) = \text{Concat}(head_1, \ldots, head_h)W^{O_2}$$

$$\textbf{where } head_i = \text{Attention}(Q_2 W_i^{Q_2}, I_K W_i^{I_K}, I_V W_i^{I_V})$$

where the $W_i^{Q_2} \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^{I_K} \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^{I_V} \in \mathbb{R}^{d_{model} \times d_v}$ and $W^{O_2} \in \mathbb{R}^{hd_v \times d_{model}}$. In this work, we set $h = 8$ attention head, $d_{model} = 512$, and $d_k = d_v = 64$. The Attention(·) is "scaled dot-product attention" [14]. Compared with the most used additive attention [7], dot-product attention is faster and more space-efficient in practice.

In the "decoder attention" layers, the queries Q_1, keys T_K, and values T_V originate from the previous decoder layer. The input is positional encoding and embeddings. We inject positional information into the input embeddings allowing the model use the order of the output token from embedding layers. Mathematically, the layers can be represented as:

$$\text{MultiHead}(Q_1, T_K, T_V) = \text{Concat}(head_1, \ldots, head_h)W^{O_1}$$

$$\textbf{where } head_i = \text{Attention}(Q_1 W_i^{Q_1}, T_K W_i^{T_K}, T_V W_i^{T_V})$$

where the $W_i^{Q_1} \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^{T_K} \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^{T_V} \in \mathbb{R}^{d_{model} \times d_v}$, and $W^{O_1} \in \mathbb{R}^{hd_v \times d_{model}}$.

The working flow of the decoder block can be summarized as follows. The process has three main stages, beginning with the "decoder attention" layers handling embeddings. Then the "encoder-decoder attention" layers connect the encoder and decoder. Finally, the position-wise fully connected feed-forward network processes the previous output. A residual connection [6] and a layer normalization [1] process the output of each stage. Similar Transformer, we use a stack of $N = 6$ identical layers.

C. Training

We train the model in an end-to-end manner. The dataset contains GUI screenshots and Domain Specific Languages (DSL), which reduce search space and token size. Unlike [2, 13], which employed a fixed-size sliding window to obtain slices of code, our method can capture whole code tokens.

Unlike [16, 3], which use attention mechanism in RNN, our method only utilizes attention mechanism in the decoder.

We train the model using minimizing penalized cross-entropy loss to train the model. Learning rate is set to 0.0001.

D. Structural Cross Entropy

Although most works [2, 13, 16, 3] use error rate as an evaluation indicator, it does not reflect the front-end code's structural characteristics. MBLUE [15] proposes a new model evaluation method, but in practice, it does not fully reflect these characteristics either. Therefore, we designed a structure cross entropy (SCE) to evaluate different models. The equation can be represented as the following:

$$\mathrm{SCE} = \frac{\sum_{j=0}^{L} \mathrm{H}(\hat{y}_j, y_j) + \mathrm{H}\left(\widehat{y^{\mathrm{pre}}}_j, y_j^{\mathrm{pre}}\right) + \mathrm{H}\left(\widehat{y^{\mathrm{post}}}_j, y_j^{\mathrm{post}}\right)}{3\mathrm{L}}$$

where L is the number of sentences in the test dataset, and $\mathrm{H}(\cdot)$ represents a cross-entropy function. y denotes the original DSL code sequence. y^{pre} represents the DSL code sequence processed by preorder traversal. y^{post} indicates the DSL code sequence processed by postorder traversal. y_j is the ground truth for the j-th image. $\hat{y}_j$ is the prediction for the j-th image. Fig. 2 displays different traversal methods in each column.

We utilize the results of three traversal methods to evaluate tree structure because they can uniquely determine a tree [4].

Figure 2 displays an example of DSL code traversal sequences using different traversal methods, with the input being the ground truth and generated trees, respectively.

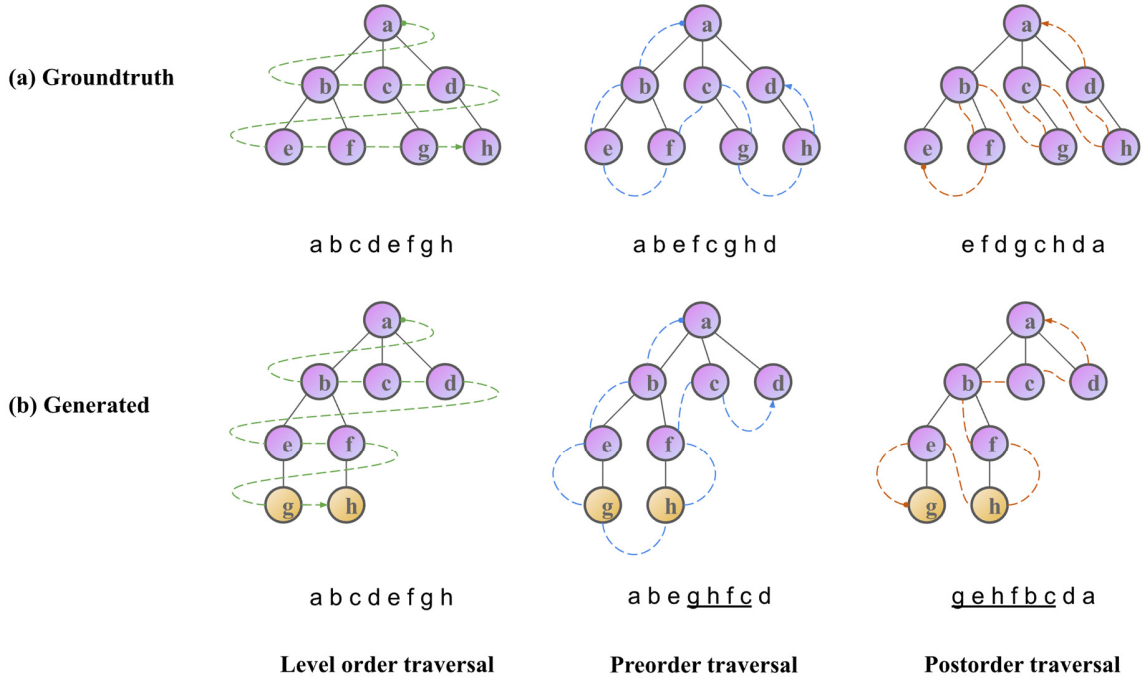

Figure 2. The diagram of structure evaluation

IV. Experiments and results

A. Setup

Data: We implement the proposed front-end code generation method on two datasets: Pix2code and our own dataset.

The first one is the Pix2code dataset provided by [2], which contains 1742 Web GUI-code pairs.

The second one is our dataset. We call the dataset Pix2code++. It consists of 3483 Web GUI-code pairs. The reason for establishing Pix2code++ because the examples in the Pix2code dataset are too simple for existing methods. Half of the data in the Pix2code++ dataset consists of Pix2code, and we create the other half. The code length of half of the dataset we created is, on average, twice as long as Pix2code. As a result, the average length of Pix2code++ is 1.5 times longer than Pix2code.

Implementation Details: First, we normalize the size of the input GUI screenshot to 224x224. For the encoder part, we use a pre-trained ResNet-152-based encoder. We employ an embedding layer for the input DSL code to learn the feature representation. We utilize Adam [8] as an optimizer in various training processes.

Baseline methods: We present the results of two baseline methods to verify the effectiveness of decoders based on multi-head attention. These methods utilize LSTM and GRU, respectively. We employ these two models as decoders because these two models are widely used in code generation from GUI screenshot tasks. For example, LSTM [2, 16], GRU [3]. Therefore, we set the decoder of baseline-1 as LSTM and

baseline-2 as GRU. Both baseline methods are identical except for the decoder.

B. Evaluation Results

Table 1. compares our method with Baseline-1 and Baseline-2 under the Pix2code and Pix2code++ datasets for test cross entropy. When comparing our method with other methods, our method outperforms the baselines for both evaluation methods, and this result is more evident for the Pix2code++ dataset. The above results demonstrate that our proposed method is more advantageous in decoding complex GUI screenshots features. The results using structure cross entropy are shown in Table 2. Compared with cross-entropy, the difference between the results obtained by structural cross-entropy is more pronounced.

Table 1. Comparison of the cross entropy of our method with baseline-1, baseline-2 on two test datasets.

Dataset	**Cross entropy**		
	Baseline-1	Baseline-2	Ours
Pix2code	0.0580	0.1062	**0.0441**
Pix2code++	0.0795	0.1176	**0.0376**

Table 2. Results for structural cross-entropy

Dataset	**Structure cross entropy**		
	Baseline-1	Baseline-2	Ours
Pix2code	0.6014	0.6738	**0.4786**
Pix2code++	1.8335	1.2459	**1.0383**

V. Conclusion and prospect

This paper addresses the problem of generating code from GUI screenshots. We design a decoder method based on the multi-head attention mechanism. Our method achieves the best performance on both public and self-collected datasets. Our experiments demonstrate that the multi-headed attention mechanism in the decoder is highly effective. Moreover, we propose a novel evaluation metric that can comprehensively assess the structure of the generated code. We plan to employ a larger-scale model to handle real-world datasets for future work.

Acknowledgments

This work is supported by National Natural Science Foundation of China (U19A2067, 61976051).

References

[1] BA, J.L., KIROS, J.R., and HINTON, G.E. (2016) *Layer normalization*. arXiv preprint arXiv:1607.06450.

[2] BELTRAMELLI, T. (2018) *pix2code: Generating code from a graphical user interface screenshot*. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, pp. 1-6.

[3] CHEN, W.-Y., PODSTRELENY, P., CHENG, W.-H., CHEN, Y.-Y., and HUA, K.-L. (2022) *Code generation from a graphical user interface via attention-based encoder–decoder model*. Multimedia Systems, 28, pp. 121-130.

[4] CORMEN, T.H., LEISERSON, C.E., RIVEST, R.L., and STEIN, C. (2022) *Introduction to algorithms*. MIT press.

[5] FENG, S., JIANG, M., ZHOU, T., ZHEN, Y., and CHEN, C. (2022) *Auto-Icon+: An Automated End-to-End Code Generation Tool for Icon Designs in UI Development*. ACM Trans. Interact. Intell. Syst.

[6] HE, K., ZHANG, X., REN, S., and SUN, J. (2015) *Deep Residual Learning for Image Recognition*. arXiv preprint arXiv:1512.03385.

[7] IOFFE, S., and SZEGEDY, C. (2015) *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. In: International conference on machine learning, pmlr, pp. 448-456.

[8] KINGMA, D.P., and BA, J. (2014) *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

[9] LI, J., ZHOU, T., CHEN, Y., CHANG, Y., ZHEN, Y., SUN, L., and CHEN, L. (2022) *ULDGNN: A Fragmented UI Layer Detector Based on Graph Neural Networks*. arXiv preprint arXiv:2208.06658.

[10] LI, Y., AMELOT, J., ZHOU, X., BENGIO, S., and SI, S. (2020) *Auto Completion of User Interface Layout Design Using Transformer-Based Tree Decoders*. arXiv preprint arXiv:2001.05308.

[11] LI, Y., LI, G., ZHOU, X., DEHGHANI, M., and GRITSENKO, A. (2021) *VUT: Versatile UI Transformer for Multi-Modal Multi-Task User Interface Modeling.* arXiv preprint arXiv:2112.05692.

[12] LI, Y., SI, S., LI, G., HSIEH, C.-J., and BENGIO, S. (2021) *Learnable fourier features for multi-dimensional spatial positional encoding.* In: Advances in Neural Information Processing Systems, 34, pp. 15816-15829.

[13] LIU, Y., HU, Q., and SHU, K. (2018) *Improving pix2code based Bi-directional LSTM. In: 2018 IEEE International Conference on Automation*, Electronics and Electrical Engineering (AUTEEE), IEEE, pp. 220-223.

[14] VASWANI, A., SHAZEER, N., PARMAR, N., USZKOREIT, J., JONES, L., GOMEZ, A. N., KAISER, Ł., & POLOSUKHIN, I. (2017). *Attention is all you need*. In Advances in neural information processing systems, 30 (pp. 5998-6008).

[15] YAO, X., YAP, M. H., & ZHANG, Y. (2022). *Towards a Deep Learning Approach for Automatic GUI Layout Generation*. Springer Nature Singapore.

[16] ZHU, Z., XUE, Z., & YUAN, Z. (2019). *Automatic Graphics Program Generation Using Attention-Based Hierarchical Decoder.* Springer International Publishing.