Multimedia Systems (2022) 28:121–130
https://doi.org/10.1007/s00530-021-00804-7

REGULAR PAPER

# Code generation from a graphical user interface via attention-based encoder–decoder model

**Wen-Yin Chen[1] · Pavol Podstreleny[2] · Wen-Huang Cheng[3] · Yung-Yao Chen[4] · Kai-Lung Hua[2]**

Received: 9 June 2020 / Accepted: 24 April 2021 / Published online: 18 May 2021


**Abstract**
Code generation from graphical user interface images is a promising area of research. Recent progress on machine learning methods made it possible to transform user interface into the code using several methods. The encoder–decoder framework represents one of the possible ways to tackle code generation tasks. Our model implements the encoder–decoder framework with an attention mechanism that helps the decoder to focus on a subset of salient image features when needed. Our attention mechanism also helps the decoder to generate token sequences with higher accuracy. Experimental results show that our model outperforms previously proposed models on the pix2code benchmark dataset.

**Keywords** Code generation · Encoder–decoder · Attention · Graphical user interface

## 1 Introduction

The development of software applications often starts with a designer who is responsible for creating a graphical user interface (GUI). Software engineers later transform this graphical user interface design into the application code. Applications might target multiple environments, such as web and mobile platforms. Software engineers must dedicate time to implement a GUI for every platform, even though it is often repetitive and time-consuming work that could be automated. Automation could be tackled by leveraging deep learning methods.

The problem of code generation from the GUI image could be approached using the computer vision object detection methods. In that situation, input to the model would be a single image created by a designer, and output would represent a list of predicted elements. Every element would have the predicted category (e.g., button), $x$, $y$ coordinates, and the width and height of the bounding box (element). Finally, predicted categories of elements would be mapped into a corresponding code view for a given platform, and rendered on the screen based on predicted $x$, $y$, width, and height. There are many computer vision methods such as Fast R-CNN [1] to deal with object detection problems. These methods achieve high performance and are used in real-time tasks.

Unfortunately, there are some drawbacks to using these methods in the context of GUI code generation. More specifically, predicted $x$, $y$, width, and height coordinates are sufficient only if the absolute layout is used that draws views based on $x$, $y$, width, and height. The main problem of the absolute layout is its lack of flexibility. In general, a layout should manage to display elements correctly, when

Communicated by B.-K. Bao.

✉ Kai-Lung Hua
hua@mail.ntust.edu.tw

Wen-Yin Chen
g110351002@grad.ntue.edu.tw

Pavol Podstreleny
m10715804@mail.ntust.edu.tw

Wen-Huang Cheng
whcheng@nctu.edu.tw

Yung-Yao Chen
yungyaochen@mail.ntust.edu.tw

[1] Department of Arts and Design, National Taipei University of Education, Taipei, Taiwan

[2] Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan

[3] Institute of Electronics, National Chiao Tung University, Hsinchu, Taiwan

[4] Department of Electronic and Computer Engineering, National Taiwan University of Science and Technology, Taipei, Taiwan

configuration changes such as screen rotation on a mobile phone, or browser resizing happen. Absolute layout lacks this ability, because it draws views based on predefined ($x$, $y$) position that can be off-screen when resizing of browser or rotation of phone occurs. Therefore, more complicated hierarchical layouts are used nowadays.

The graphical user interface is characteristic of its hierarchical structure. That means that graphical elements are nested in the tree-like structure where one element might have multiple children elements. Moreover, graphical elements can be divided into two categories. The first category represents elements that are visible on the screen (text, images, buttons) called views. The second category of elements is called view groups, that act as a container for other views and specify how these views are displayed.

Computer vision object detection methods are great for capturing views but lack the ability to capture view groups. In general, it is hard to create a dataset using bounding boxes that capture view groups. Beltramelli [2] introduced another approach to code generation from GUI. In this approach, the author uses an input image and string representation that describes the image. This string representation consists of individual tokens written in a domain-specific language (DSL) introduced by the Beltramelli [2]. Domain-specific language represents the language that describes views, view groups, and the relationship between them (see Fig. 1). The author introduced DSL to reduce the size of the search space and vocabulary size (i.e., the total number of tokens supported by the DSL).

We can think of this approach as an image captioning task [3]. The model must be powerful enough to solve the computer vision challenges of determining which objects are in an image and must also be capable of capturing and expressing the relationships between views and view groups in DSL language.

String representations that contain DSL tokens might have different lengths based on different input images. Recurrent neural networks (RNNs) represent a class of neural networks that can analyze time-series data and output the sequence of arbitrary length. Therefore, they might be a good fit for code generation problems. In this situation, the model takes image and produces one DSL token every time-step. Production of tokens is finished when special *end* token is predicted at any time-step. Afterward, all tokens are mapped into a corresponding code view, view group for a given platform, and rendered on the screen. The advantage of this method is that generated tokens can represent both view and view groups.

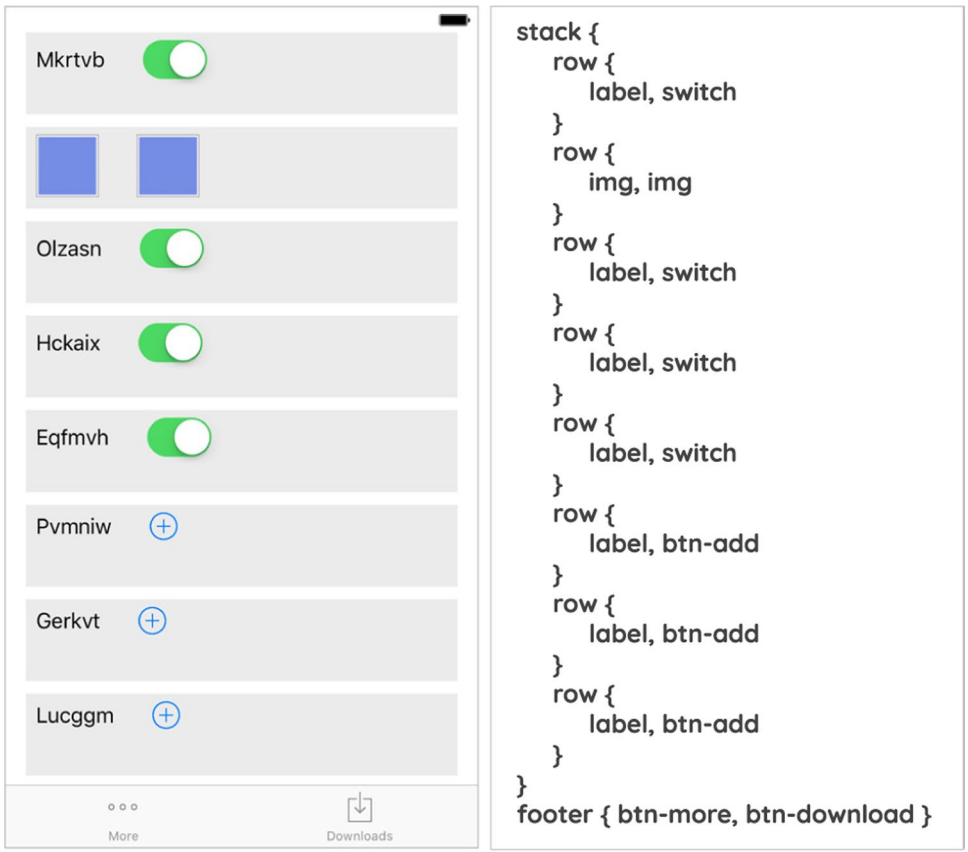



**Fig. 1** Data-point contains the input image and the string representation of image describing the image. This string representation consists of individual tokens (i.e., switch, button) written in DSL. The example of the data-point is used from the pix2code [2] dataset

Therefore, the hierarchical structure of elements is automatically generated.

In this paper, we propose a sequence-to-sequence model based on convolutional and recurrent neural networks with a soft attention mechanism that can learn variable-length strings of tokens from visual input. This model fits into the category of models similar to [2, 4, 5] that are inspired by an image captioning task.

We quantitatively validate the usefulness of the model with the state-of-the-art performance on the benchmark pix-2code [2] dataset.

Moreover, inspired by [3, 6], we introduce the attention mechanism that focuses on the part of the image that contains the GUI element while generating a token representation of that particular GUI element at a given time-step. Our model is also encouraged to pay equal attention to every part of the image during the tokens generation. This is important, because GUI elements appear at different locations in the input image.

## 2 Related work

This section provides a background on previous work related to image captioning and code generation. Generating GUI code using machine learning techniques is a relatively new area of research. In this section, we present methods that fit into the category that is inspired by the image captioning task. Learning-based approaches have shown state-of-the-art performance in many problem domains [7–12]. Likewise, most of the recent approaches in image captioning, discussed here, involve machine learning.

A significant number of researchers have addressed the problem of image captioning with impressive results, showing that deep neural networks can learn latent variables describing objects in an image and their relationships with corresponding variable-length textual descriptions [2].

Chen et al. [13] proposed the bi-directional mapping between images and their sentence-based descriptions using RNN. Mao et al. [14] proposed the multimodal Recurrent Neural Network (m-RNN) that directly models the probability distribution of generating a word given previous words and an image. Their model consists of two sub-networks: RNN for sentences and a CNN for images. The idea of the attention mechanism proposed by Bahdanaou et al. [6] used in machine language translation was widely adopted in the image captioning task [3, 15, 16]. One crucial reason image caption generation is well suited for the encoder–decoder framework of machine translation, because it is analogous to "translating" an image to a sentence [3].

Beltramelli [2] proposed one of the first ideas of generating code from GUI using machine learning techniques. The author's approach was based on a convolutional and recurrent neural network that allowed the generation of computer tokens from a single user interface image. No engineered feature extraction pipeline nor expert heuristics was designed to process the input data. The proposed model used long short-term memory [17] (LSTM) for encoder and decoder parts of the model. The author also introduced a dataset that is divided into three different sub-datasets. Datasets consist of GUI images and string representations of these images written in DSL.

Yanbin Liu et al. [4] achieved an improvement of pix-2code using bi-directional LSTM, which allows the output layer to get complete past and future context information for each point in the input sequence. Both pix2code and solution proposed by Yanbin Liu et al. did not incorporate any form of attention mechanism in their architecture.

The attention-based method was proposed by Zhu et al. [5] using a hierarchical decoder. Their method uses encoder that encodes the input image into the feature vector. This vector is used in "Block" LSTM that computes the number of blocks of code that should be generated. Every time-step "Block" LSTM produces a hidden state that is used in attention mechanism that focuses on the part of the image. The output of attention mechanism is used as input for "Token" LSTM that generates DSL tokens for the given block. Their method generates tokens block by block. The authors were able to improve pix2code [2] paper for each sub-datasets.

There are potential drawbacks using the method proposed by Zhu et al. [5]. "Block" LSTM might mispredict the number of blocks, which leads to automatic skipping of children tokens of that block. Another potential drawback is that output of attention mechanism is used as a static input to "Token" LSTM that generates multiple tokens. Moreover, the attention mechanism has to focus on the larger part of the image that has to be sufficient for generating multiple tokens.

The incorporation of attention into neural networks for the caption generation task has achieved significant performance. In the same spirit, our model uses attention mechanisms to choose salient image features for code generation as needed.

Compare to Zhu et al. [5], our model emphasizes different parts of the image every time-step of token generation. Our model focuses its attention more on a single GUI element rather than on multiple elements. Moreover, the hierarchical structure of the user interface can be produced without using the hierarchical decoder. Hierarchy is already explicitly defined in DSL of the dataset and can be learned by the model.

## 3 Data preprocessing pipeline

Before we mention our proposed model, we would like to introduce data preprocessing pipeline that transforms raw data-points into data-points usable for the model. The model

is evaluated on the pix2code [2] dataset. Each data point in this dataset contains input image and the string representation of the input image written in DSL (see Fig. 1).

First, the images are resized into 224×224 pixels. Then, they are normalized using mean and standard deviation. Image transformations are sufficient for the test dataset. However, for train and validation datasets, additional label preprocessing has to be done.

Prepossessing the string representation of the input image follows multiple steps. First, this raw string is tokenized into a list of DSL tokens. Then, the vocabulary that contains all unique DSL tokens from the dataset is created. We can think of vocabulary as a dictionary, where keys represent DSL tokens, and values represent unique integer identifiers. Next, special *start*, *end*, and *pad* tokens are added to the vocabulary. The list of DSL tokens for a given data-point is mapped into the list of numerical values based on created vocabulary. Finally, this list of unique integer identifiers is used in two ways. First, the integer identifier of the *start* token is prepended into this list. This list is used as input for the RNN model. Second, the integer identifier of the *end* token is appended into the list. This list is used as the label.

The label and input for RNN can be thought of as univariate time-series data. We can think of them as 1D vectors, where the dimension of the vector represented by $c$ represents the number of time-steps

$$\mathbf{y} = \{y_1, \ldots, y_c\}. \tag{1}$$

Therefore, each data-point contains $c$ labels one for every time-step. Each label is represented by one of $k$ integer values from vocabulary, where $k$ represents the size of the vocabulary.

For example, $y_1$ represents the label at time-step 1 for a given data-point.

In summary, these transformations produce resized, and normalized image, $c$ univariate inputs for RNN, and $c$ labels, for each data-point. The teacher forcing method is used for training our RNN.

## 4 Proposed method

A short overview of proposed model is described in the following paragraphs. Detailed information on parts of the model are described in consecutive sections. Visual representation of proposed model is shown in Fig. 2.

The goal of the proposed model is to transform the GUI image into the list of DSL tokens that represent views, view groups, and relationships between them. These predicted tokens are later mapped into views, view groups, and compiled into the final code for a given platform and rendered on the screen.

To achieve this goal, several steps have to be done.

The first step is the extraction of important features from the image. This is done by a pre-trained convolutional neural network (CNN). The CNN extracts a set of feature vectors that we call annotation vectors. This part of the model is called encoder, because it encodes the original image into another representation.

The second step is the start of the token generation process. The recurrent neural network (RNN) is used, because it allows the generation of a variable number of tokens based on the input GUI image. The RNN uses annotation vectors extracted by the encoder as input every time-step. Annotation vectors could be used as a static representation, as shown in [2]. However, several researchers [3, 6] showed that treating this representation as dynamic gives better performance. Therefore, our proposed model contains the attention module that learns to focus on different annotation vectors every time-step.

Model generates token every time-step and generation is stopped as soon as special *end* token is generated.

In the following sections, vectors are represented by bold lowercase letters, and matrices are represented with bold uppercase letters.

### 4.1 Encoder

Convolutional neural networks are used for a wide range of computer vision problems. Their topology is allowing them to learn latent representations from the raw input images. A convolutional neural network is used in our model to extract a set of feature vectors, which we refer to as annotation vectors. The extraction of annotation vectors is done from a lower convolutional layer. This extraction produces $L$ annotation vectors, each of which is a $D$-dimensional representation that corresponds to a part of the image

$$a = \{\mathbf{a}_1, \ldots, \mathbf{a}_L\}, \mathbf{a}_i \in \mathbb{R}^D. \tag{2}$$

Annotation vectors are used later in the attention mechanism that selectively focuses on specific annotation vectors at each time-step.

### 4.2 Decoder

Recurrent neural networks represent a class of neural networks that are commonly used for predicting future based on past data values. They can analyze time-series data and work on a sequence of arbitrary lengths. One of the difficulties of RNNs is unstable gradient, which can be alleviated using layer normalization [18] techniques. Another problem is limited short-term memory. Every time-step, some information is lost, and at the end of the sequence, RNN's state contains almost no trace of the first inputs.

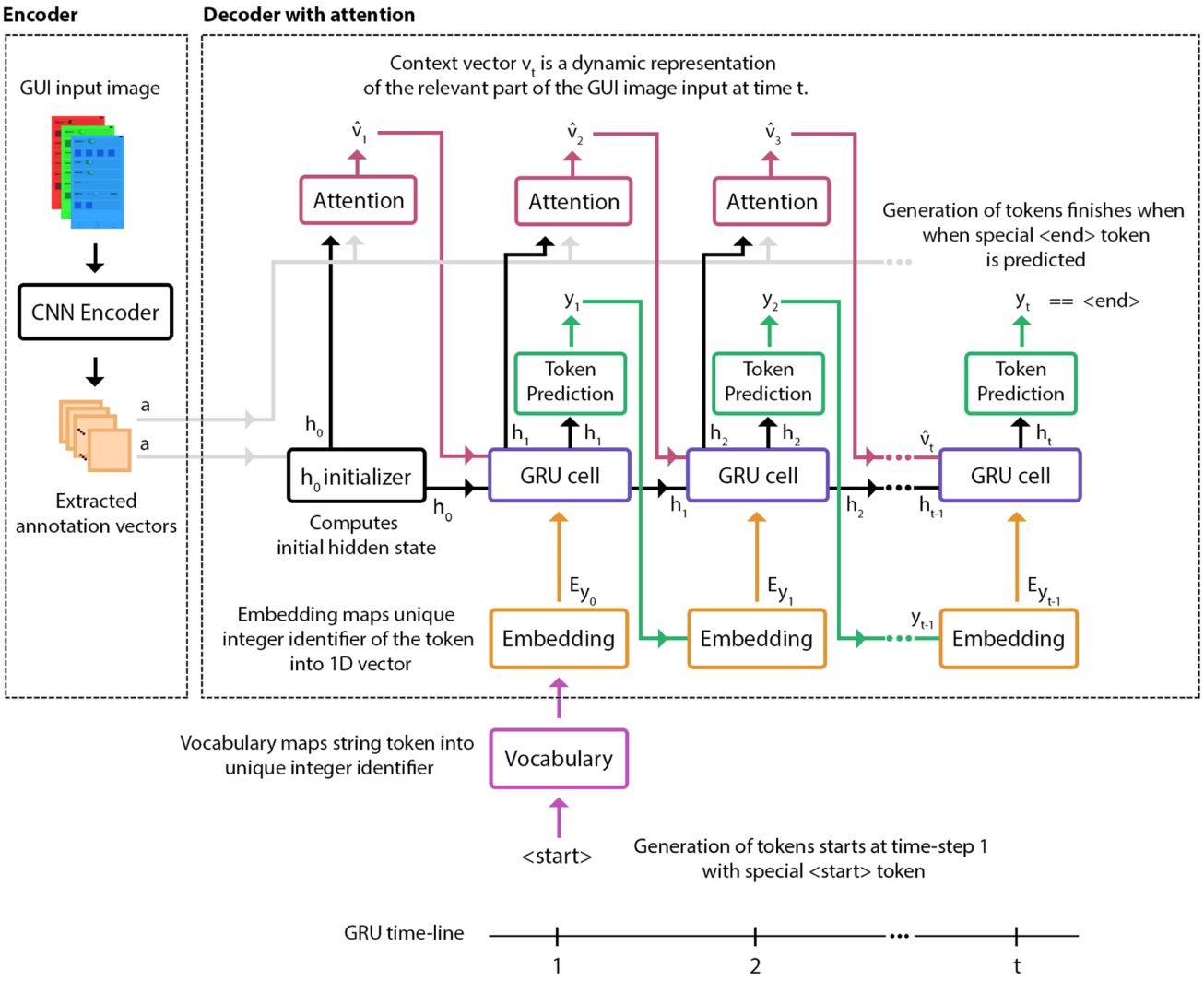



**Fig. 2** Overview of our proposed model for automatic code generation from graphical user input. The input image is passed through the encoder, which extracts annotation vectors. These vectors are used for calculation of the initial hidden state of the decoder and in the attention mechanism. Decoder's cell use context vector is previously generated token and previous hidden state as input. The output of the decoder's cell is used as input to attention mechanism and also for token prediction at given time-step

To tackle short-term memory problems, various long-term memory cells have been proposed. Long short-term memory (LSTM) was proposed by Hochreiter et al. [17]. Gated recurrent unit (GRU) was motivated by the LSTM unit, but is much simpler to compute and implement [19]. Both LSTM and GRU alleviate the problem with an unstable gradient and can detect long-term dependencies in the data.

Our decoder represents a GRU cell. We use GRU because of its simpler structure and computational complexity. The initial hidden state of the GRU is predicted by an average of the annotation vectors fed to one hidden layer neural network as showed in Eqs. 3, 4:

$$\mathbf{u} = \left( \frac{1}{L} \sum_{i}^{L} \mathbf{a}_i \right) \tag{3}$$

$$\mathbf{h}_0 = \mathbf{W}\mathbf{u} + \mathbf{b}. \tag{4}$$

**W** and **b** in Eq. 4 represents weight matrix and bias term, respectively.

The GRU cell produces output every time-step conditioned on the previous hidden state, previously generated token embedding, and context vector. For brevity, we skipped bias terms in Eqs. 5,6,7,8. These equations represent

the implementation details of the GRU cell that is shown in Fig. 3:

$$\mathbf{r}_t = \sigma\left(\mathbf{W}_{er}\mathbf{E}\mathbf{y}_{t-1} + \mathbf{W}_{vr}\hat{\mathbf{v}}_t + \mathbf{W}_{hr}\mathbf{h}_{t-1}\right) \tag{5}$$

$$\mathbf{z}_t = \sigma\left(\mathbf{W}_{ez}\mathbf{E}\mathbf{y}_{t-1} + \mathbf{W}_{vz}\hat{\mathbf{v}}_t + \mathbf{W}_{hz}\mathbf{h}_{t-1}\right) \tag{6}$$

$$\mathbf{g}_t = \phi\left(\mathbf{W}_{eg}\mathbf{E}\mathbf{y}_{t-1} + \mathbf{W}_{vg}\hat{\mathbf{v}}_t + \mathbf{W}_{hg}\left(\mathbf{r}_t \otimes \mathbf{h}_{t-1}\right)\right) \tag{7}$$

$$\mathbf{h}_t = \mathbf{z}_t \otimes \mathbf{h}_{t-1} + \left(1 - \mathbf{z}_t\right) \otimes \mathbf{g}_t. \tag{8}$$

Here, $\mathbf{r}_t$, $\mathbf{z}_t$, $\mathbf{g}_t$, $\mathbf{h}_t$ represent reset gate, updated gate, g gate, and hidden state of GRU, respectively. $\mathbf{E} \in \mathbb{R}^{k \times m}$ represents the embedding matrix, where $k$ is vocabulary size, and $m$ is the embedding dimension. Embedding matrix maps unique integer values (that represent tokens) from vocabulary into $m$-dimensional vectors.

$\mathbf{E}\mathbf{y}_{t-1}$ represents $m$-dimensional embedding representation of token label from the previous time-step. Vector $\hat{\mathbf{v}}_t \in \mathbb{R}^D$ represents a context vector that captures visual information associated with particular input location at time-step $t$. A detailed information about context vector $\hat{\mathbf{v}}_t$ is described in the attention section. Vector $\mathbf{h}_{t-1} \in \mathbb{R}^n$ represents the $n$-dimensional hidden state of GRU cell. The character $\otimes$ represents element-wise multiplication, $\sigma$ logistic sigmoid activation function, and $\phi$ hyperbolic tangent activation function. $\mathbf{W}_{er}$, $\mathbf{W}_{vr}$, $\mathbf{W}_{hr}$, $\mathbf{W}_{ez}$, $\mathbf{W}_{vz}$, $\mathbf{W}_{hz}$, $\mathbf{W}_{eg}$, $\mathbf{W}_{vg}$, and $\mathbf{W}_{hg}$ represent weight matrices.

Each time-step, the output of the GRU cell is used in three ways. First, it is used as one of the inputs for the attention module. Second, it is used for predicting token at a given time-step. In this situation, the output of the GRU cell is linearly transformed, and this transformation produces scores.

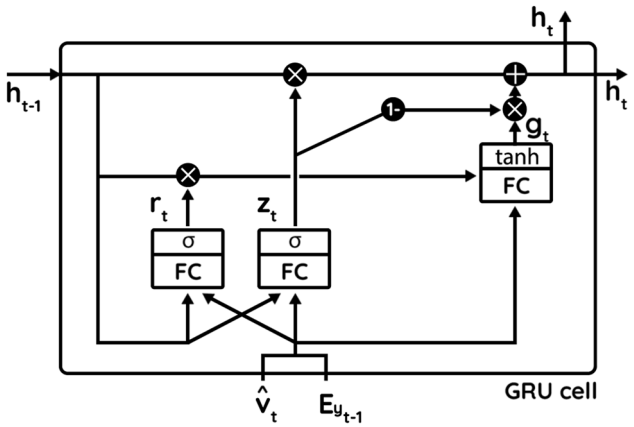



**Fig. 3** GRU cell: Update gate $z_t$ selects whether the hidden state is to be updated with a new hidden state $g_t$. Reset gate $r_t$ decides whether the previous hidden state is ignored. See Eqs. 5, 6, 7, 8 for mode details

Scores are transformed into probabilities using a softmax function. Token with the highest probability is generated at a given time-step. Third, it is used as input to GRU cell next time-step.

### 4.3 Attention

The purpose of attention mechanism in our model is to put more emphasis on annotation vectors that are important for generating correct token at a given time-step. Our attention mechanism is capable of looking at the different part of the image every time-step. This is important, because GUI elements usually appear at different positions in the image.

The attention mechanism takes two inputs. The first input represents all annotation vectors extracted by the encoder. The second input represents GRU's hidden state generated at the previous time-step $\mathbf{h}_{t-1}$. The output of the attention mechanism at given time-step $t$ represents $D$-dimensional context vector $\hat{\mathbf{v}}_t$, where $D$ represents the dimension of an annotation vector.

Several attention mechanisms have been proposed in the past to compute context vector $\hat{\mathbf{v}}_t$. The Bahdanaou attention [6], also called concatenative attention, concatenates the encoder's output with the decoder's previous hidden state. Another mechanism was proposed by Luong et al. [20] using a dot product between encoder's output and decoder's previous hidden state.

Our attention mechanism is inspired by the mechanism proposed by Bahdanaou et al. [6]. Equations 9, 10, 11 show how we calculate $e_{ti}$ that represents the score for annotation vector $\mathbf{a}_i$ at time-step $t$. Score measures how well annotation vector $\mathbf{a}_i$ is aligned with the decoder's previous hidden state $\mathbf{h}_{t-1}$:

$$\mathbf{f}_t = \mathbf{W}\mathbf{h}_{t-1} + \mathbf{b} \tag{9}$$

$$\mathbf{q}_i = \mathbf{W}\mathbf{a}_i + \mathbf{b} \tag{10}$$

$$e_{ti} = \mathbf{w}^T relu\left(\mathbf{f}_t + \mathbf{q}_i\right). \tag{11}$$

$\mathbf{W}$ and $\mathbf{b}$ represent weight matrices and bias terms, respectively. First, the linear transformation of the previous hidden state $\mathbf{h}_{t-1}$ and annotation vector $\mathbf{a}_i$ is applied. The result of these transformations is added together and passes through a rectified linear unit activation function. Dot product with transposed learned vector $\mathbf{w}^T$ is applied in the end:

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}. \tag{12}$$

Finally, all scores go through a softmax layer to get weight $\alpha_{ti}$ for each annotation vector at time $t$ as showed in Eq. 12. Computed weights tell us how much to focus on

the particular annotation vectors at given time-steps, where weight with value 0 represents no focus, and weight with value 1 represents full focus (Table 1).

As mentioned earlier, we want our attention mechanism to focus on the part of the image where the GUI element occurs while generating the corresponding token representation of that GUI element. Kelvin Xu et al. [3] introduced a mechanism that allows the attention module to put more emphasis on the real objects (e.g., hat, ball, car) in the images. Inspired by their work, we implement gating scalar $\beta_t$ that put more emphasis on GUI elements in the image at given time-step. Gating scalar $\beta_t$ is predicted from the previous hidden state $\mathbf{h}_{t-1}$ at each time-step $t$ as shown in Eq. 13:

$$\beta_t = \sigma\left(\mathbf{w}^T \mathbf{h}_{t-1}\right). \tag{13}$$

Symbol $\sigma$ represents sigmoid activation function and $\mathbf{w}^T$ represents transposed learned vector.

Once annotation weights for each annotation vector and gating scalar are computed, attention module computes context vector $\hat{\mathbf{v}}_t$ that represents a dynamic representation of the relevant part of the image input at time $t$:

$$\hat{\mathbf{v}}_t = \beta_t \sum_{i=1}^{L} \alpha_{ti} \mathbf{a}_i. \tag{14}$$

Context vector $\hat{\mathbf{v}}_t$ defined in Eq. 14 is used as input to the decoder at time-step $t$.

### 4.4 Objective function

Our model was trained end-to-end with stochastic gradient descent by minimizing penalized cross-entropy loss function, as shown in Eq. 15. We want our model to focus to every part of the image over the course of the generation, because GUI elements occur in different locations in the picture. Inspired by Kelvin Xu et al. [3], our model use regularization that encourages $\sum_t \alpha_{ti} \approx 1$ which can be interpreted as encouraging the model to pay equal attention to every part of the image:

$$L_d = -\left(\sum_{t}^{C}\sum_{j}^{K} y_{tj} \log\left(p_{tj}\right)\right) + \lambda \sum_{i}^{L}(1 - \sum_{t}^{C} \alpha_{ti})^2. \tag{15}$$

$L_d$ represents cross-entropy loss computed for one data point $d$ in dataset. $C$, $K$, and $L$ represent number of time-steps, vocabulary size (number of classes), and number of annotation vectors, respectively. The $y_{tj}$ represents class label $j$ at time-step $t$ that has value 0 or 1, where 1 represents true label for given time-step $t$. The $p_{tj}$ represents predicted probability of class $j$ at time-step $t$. The $\alpha_{ti}$ represents the weight of annotation vector $i$ at time-step $t$. Symbol $\lambda$ represents the hyper-parameter of the loss function.

## 5 Experiments

In this section, we describe our experimental methodology and quantitative results, which validate the effectiveness of our model for GUI code generation. Moreover, the qualitative analysis of attention mechanism is also included.

### 5.1 Training procedure

To extract annotation vectors, convolutional neural network VGG-16 proposed by Simonyan et al. [21] was used. VGG-16 was pre-trained on the ImageNet [22] dataset. In experiments, we used a 14×14×512 feature map of the fifth convolutional layer before max pooling. This means that we extracted 196 annotation vectors with the dimension of each vector to be equal to 512. The last two layers of the CNN encoder were fine-tuned during the training procedure.

The GRU cell, embedding vector, and context vector were set to 512 dimensions.

The only regularization strategy used for our model was a dropout: 10%, loss regularization $\lambda$: 1.0 (Eq. 15), and early stopping on validation accuracy when the model did not improve over the last 16 consecutive epochs.

Moreover, our model applies learning rate decay on the RMSprop optimizer when no improvement in validation accuracy is achieved over the last four consecutive epochs. RMSprop learning rate was set to 1e-3 at the beginning of training and the decay factor was set to 1e-1.

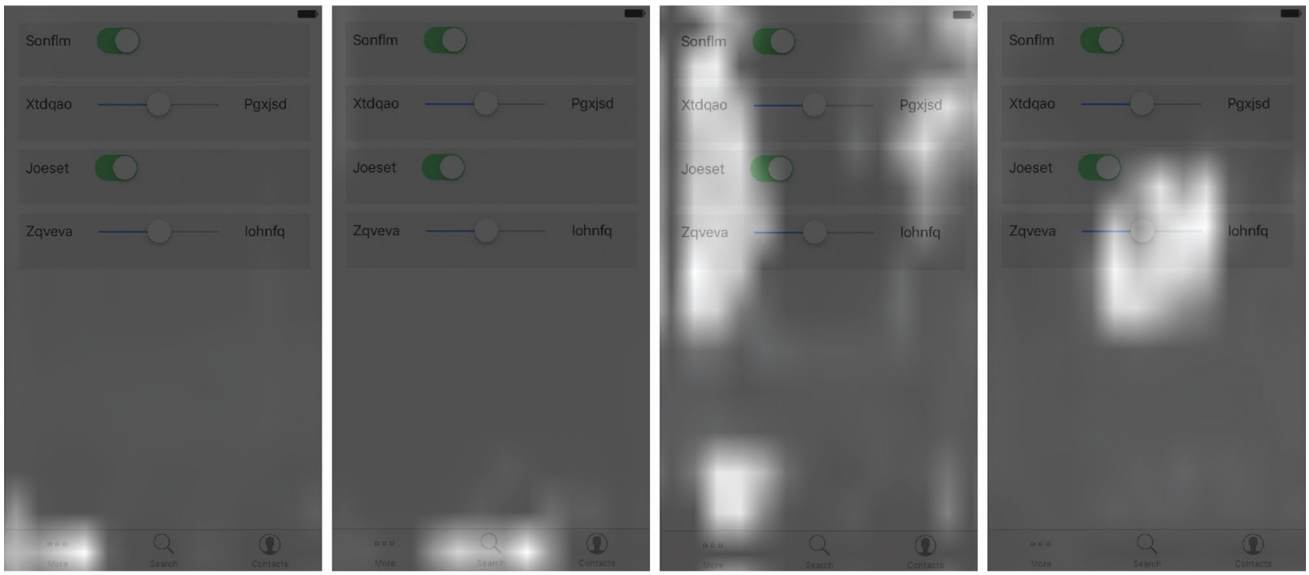
It took less than 4 h to train the model on NVIDIA Tesla K80 GPU.

### 5.2 Data

We report results on the pix2code [2] dataset, which contains 4500 training data-points and 750 test data-points divided into three sub-datasets: iOS, Android, and web-based. The summary of the dataset is presented in Table 2. Data-point contains input image representing GUI, and the string representation of the image written in DSL language. The graphical representation of the data-point can be seen in Fig. 1.

### 5.3 Quantitative analysis

We follow the same rules for evaluation, as described in pix2code [2]. The quality of the generated code is evaluated by computing the classification error for each sampled DSL token and averaged over the whole test dataset. The length difference between the generated and the expected token sequences is also counted as an error. Table 1 compares our method with the pix2code [2] and the method proposed by Zhu et al. [5]. Methods are compared on three




(a) More button focus (b) Search button focus (c) Focus on row (d) Slider focus

**Fig. 4** Visualization of the model's attention while generating more (**a**), search (**b**), row (**c**), and slider DSL token (**d**). Model learned to focus on parts of the image that are important for the token generation at a given time-step

**Table 1** Comparison of the performance of our method with pix2code, [2], Zhu et al. [5] on the test dataset using different search strategies: "greedy" represents greedy-based method, and "beam-3" and "beam-5" stand for beam search strategy using beam size 3 and 5, respectively

| Dataset type | | Error (%) | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | Greedy | | | beam-3 | | | beam-5 | | |
| | | Ours | pix2code [2] | Zhu [5] | Ours | pix2code [2] | Zhu [5] | Ours | pix2code [2] | Zhu [5] |
| pix2code | iOS | **1.61** | 22.73 | 19.00 | 1.67 | 25.22 | 17.90 | 1.67 | 23.94 | 17.43 |
| | Android | 8.41 | 22.34 | 18.65 | **6.62** | 22.58 | 18.20 | 6.62 | 40.93 | 16.31 |
| | Web | **0.13** | 12.14 | 11.50 | 0.47 | 11.01 | 11.23 | 0.47 | 22.35 | 10.88 |

**Table 2** Summary of pix2code [2] dataset containing information about training and test dataset

| Dataset type | Training set | Test set |
|---|---|---|
| iOS UI (Storyboard) | 1500 | 250 |
| Android UI (XML) | 1500 | 250 |
| Web-based UI (HTML/CSS) | 1500 | 250 |

different sub-datasets: iOS, Android, web-based using three different search strategies: greedy-search, beam search with beam size 3 and 5, respectively. Our proposed method achieved lower token generation error compare to pix2code [2] and Zhu et al. [5] over all three sub-datasets and search strategies.

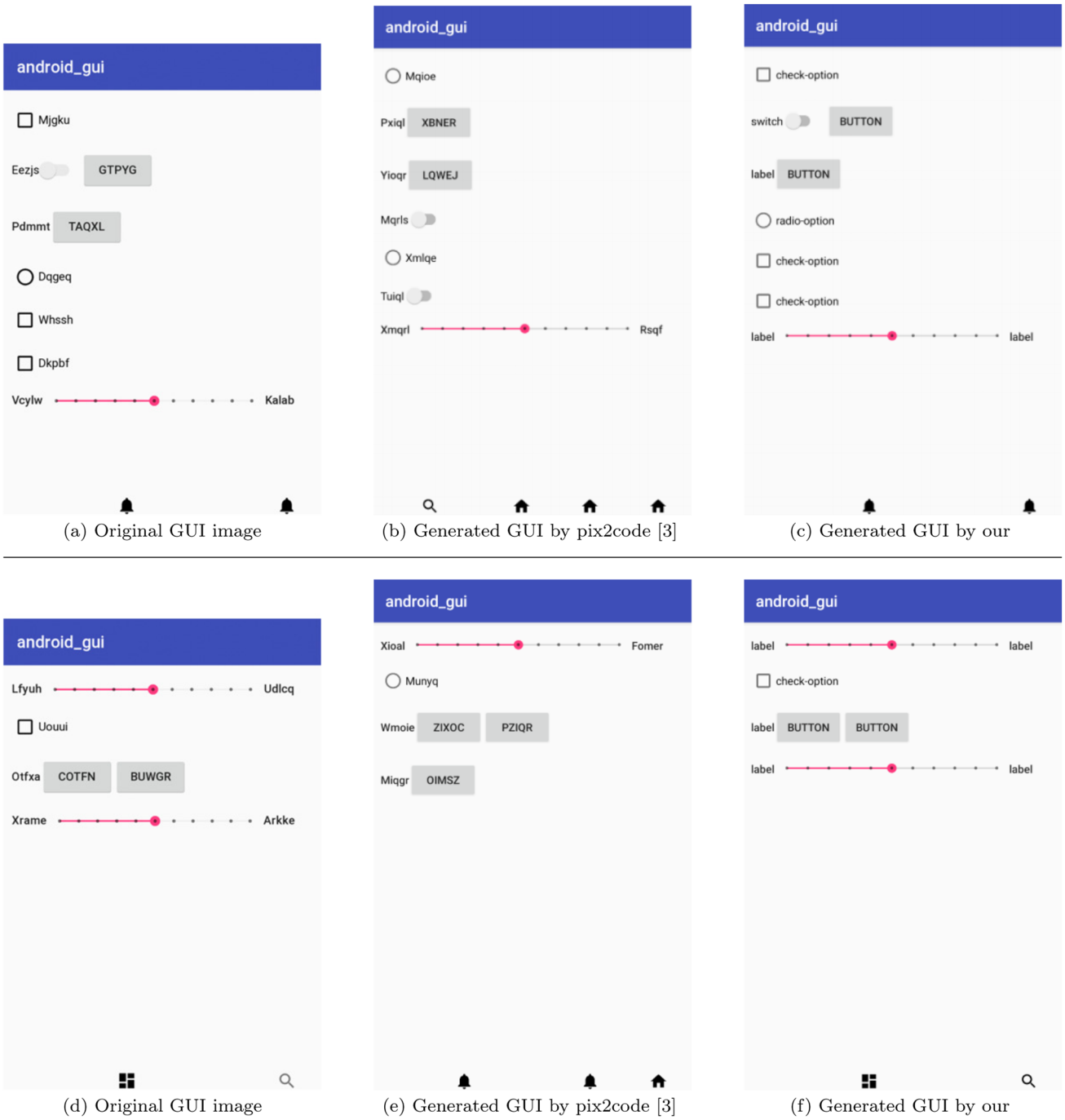
For example, using iOS sub-dataset and greedy-search strategy: our method, pix2code [2], and Zhu et al. [5] miss classified 1.6%, 22.73%, and 19.00% tokens, respectively, which show the effectiveness of our method.

## 5.4 Qualitative analysis

Visualization of the model's attention can add an extra layer of interpretability to the model's output. We use the same visualization technique introduced by Xu et al. [3]. This technique enables the visualization of the parts of the image that are used for the token generation at a given time-step.

Figure 4 demonstrates our model's capability to attend only salient parts of the image that are important for the token generation at the given time-step. This corresponds very well with the human intuition of attention that concentrates on a discrete aspect of information.

Figure 5 contains visual comparison of GUI generated by pix2code [2] and our proposed model. Based on the GUI input image (a), our model was able to predict every token correctly, which leads to generated GUI (c) that contains the




(a) Original GUI image (b) Generated GUI by pix2code [3] (c) Generated GUI by our

(d) Original GUI image (e) Generated GUI by pix2code [3] (f) Generated GUI by our

**Fig. 5** Visualization of generated android GUIs. Image (**a**) represents test image from android dataset. Images (**b**) and (**c**) represent generated GUI by pix2code [2] and our model, respectively. Image (**d**) represents test image from android dataset. Images (**e**) and (**c**) represent generated GUI by pix2code [2] and our model, respectively

same GUI elements as the input image. On the other hand, pix2code [2] model made multiple tokens' prediction errors, which leads to generated GUI (b) that does not represent the GUI input image (a).

## 6 Conclusion

We propose an attention-based method for GUI code generation. Our method achieves a state-of-the-art performance on pix2code [2] dataset using an accuracy error metric. Furthermore, we qualitatively visualize the attention of the model,

which corresponds very well to human intuition. For the next steps, we plan to work on a complex GUI dataset that captures complicated structures of real-world applications.

## References


1. Girshick, R.B.: Fast R-CNN. CoRR arXiv:1504.08083 (2015)
2. Beltramelli, T.: pix2code: Generating code from a graphical user interface screenshot. CoRR arxiv:1705.07962 (2017)
3. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A.C., Salakhutdinov, R., Zemel, R.S., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention. CoRR arXiv:1502.03044 (2015)
4. Liu, Y., Hu, Q., Shu, K.: Improving pix2code based bi-directional lstm. 2018 IEEE international conference on automation, electronics and electrical engineering (AUTEEE) p. 220–223 (2018)
5. Zhu, Z., Xue, Z., Yuan, Z.: Automatic graphics program generation using attention-based hierarchical decoder. CoRR arXiv:1810.11536 (2018)
6. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate (2014)
7. Tang, H.L., Chien, S.C., Cheng, W.H., Chen, Y.Y., Hua, K.L.: Multi-cue pedestrian detection from 3d point cloud data. In: 2017 IEEE international conference on multimedia and expo (ICME), pp. 1279–1284. IEEE (2017)
8. Hua, K.L., Hidayati, S.C., He, F.L., Wei, C.P., Wang, Y.C.F.: Context-aware joint dictionary learning for color image demosaicking. J. Vis. Commun. Image Represent. **38**, 230–245 (2016)
9. Tan, D.S., Chen, W.Y., Hua, K.L.: Deepdemosaicking: adaptive image demosaicking via multiple deep fully convolutional networks. IEEE Trans. Image Process. **27**(5), 2408–2419 (2018)
10. Sanchez-Riera, J., Hua, K.L., Hsiao, Y.S., Lim, T., Hidayati, S.C., Cheng, W.H.: A comparative study of data fusion for rgb-d based visual recognition. Pattern Recogn. Lett. **73**, 1–6 (2016)
11. Hidayati, S.C., Hua, K.L., Cheng, W.H., Sun, S.W.: What are the fashion trends in new york? In: Proceedings of the 22nd ACM international conference on multimedia, pp. 197–200 (2014)
12. Sharma, V., Srinivasan, K., Chao, H.C., Hua, K.L., Cheng, W.H.: Intelligent deployment of uavs in 5g heterogeneous communication environment for improved coverage. J. Netw. Comput. Appl. **85**, 94–105 (2017)
13. Chen, X., Zitnick, C.L.: Learning a recurrent visual representation for image caption generation. CoRR arXiv:1411.5654 (2014)
14. Mao, J., Xu, W., Yang, Y., Wang, J., Huang, Z., Yuille, A.: Deep captioning with multimodal recurrent neural networks (m-rnn) (2015)
15. Chen, L., Zhang, H., Xiao, J., Nie, L., Shao, J., Chua, T.: SCA-CNN: spatial and channel-wise attention in convolutional networks for image captioning. CoRR arxiv:1611.05594 (2016)
16. Lu, J., Xiong, C., Parikh, D., Socher, R.: Knowing when to look: adaptive attention via A visual sentinel for image captioning. CoRR arXiv:1612.01887 (2016)
17. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural Comput. **9**(8), 1735–1780 (1997)
18. Ba, L.J., Kiros, J.R., Hinton, G.E.: Layer normalization. CoRR arxiv:1607.06450 (2016)
19. Cho, K., van Merrienboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., Bengio, Y.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. CoRR arXiv:1406.1078 (2014)
20. Luong, M., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation. CoRR arXiv:1508.04025 (2015)
21. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: ICLR (2015)
22. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M.S., Berg, A.C., Li, F.: Imagenet large scale visual recognition challenge. CoRR arXiv:1409.0575 (2014)