Journal of Network and Computer Applications 161 (2020) 102636

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

A novel syntax-aware automatic graphics code generation with attention-based deep neural network

Xiongwen Pang [a], Yanqiang Zhou [a], Pengcheng Li [a], Weiwei Lin [b,*], Wentai Wu [c,**], James Z. Wang [d]

[a] School of Computer, South China Normal University, China
[b] School of Computer Science and Engineering, South China University of Technology, Guangzhou, 510006, China
[c] Department of Computer Science, University of Warwick, Coventry, CV4 7AL, United Kingdom
[d] School of Computing, Clemson University, SC, 29631, USA

ARTICLE INFO

Keywords:
Convolution neural network
Long-short term memory neural network (LSTM neural Network)
Automatic code generation
Attention mechanism
Syntax awareness

ABSTRACT

Recent advances in deep learning have made it possible to automatically translate graphical user interface (GUI) into code by an encoder-decoder framework. This framework generally uses deep convolutional neural network (CNN) to extract image features, which are then translated into hundreds of code tokens by a code generator based on a recurrent neural network (RNN). However, there are two challenges in the implementation of this framework: one is how to make full use of the information contained in the GUI and domain specified language (DSL) code, the other is how to make generated DSL code conform to syntax rules. To fully leverage the information in GUI and DSL code, we first propose a model named HGui2Code that integrates visual attention-enabled GUI features (extracted by CNN) with DSL attention-enabled semantic features (extracted by LSTM). Besides, we propose SGui2Code, a novel model that makes use of a ON-LSTM network to generate DSL code that is correct in syntax. HGui2code pays more attention to semantic information, while SGui2code focuses on grammar rules. Extensive experimental results show that our models outperform state-of-the-art methods on the web dataset, yielding 5.5% higher accuracy with the HGui2Code model and 1.5% using the SGui2Code model respectively. Although our models do not have huge boost on IOS and Android dataset, DSL code generated by our models are very close to the layout of components in corresponding GUI.

1. Introduction

With the rapid development of engineering methodology, traditional software development techniques gradually expose their limitations and inefficiency with common issues such as long development cycles, large repetitive work, and difficult system maintenance. At the same time, computer scientists have been trying to use artificial intelligence to facilitate the efficiency of programmers through new techniques like automatically generating readable programs. For example, Neelakantan et al. (2015) (2015) used a Neural Programmer to generalize combinatorial programs and Balog et al. (2016) (2016) used a deep learning system to automatically solve the input-output competition-style programming.

Successful automatic programming techniques can significantly promote the quality and efficiency of software development and release a huge of amount of human effort in producing code. A lot of envision has been outlined with the surging power of Artificial Intelligence (AI), which has been widely reckoned as a promising solution to perform server automatic programming tasks including the conversion from sketches to code, code synthesis, debugging and reconstruction, etc. It is an emerging trend to use AI in front-end development to converting GUI to code (Gui2Code) automatically. Researchers have made progress in this job by using deep neural network. Beltramelli proposed pix2code (Beltramelli, 2018) (2017) model which generates grammatically and semantically correct samples. Microsoft used Sketch2Code (TechniquesMicrosoft, 2018) (2018) project to convert hand-drawn GUI sketches into useable HTML code, etc.

Motivated by the trend of AI-aid GUI engineering, in this paper, we investigate the cutting-edge models/frameworks available for automatic generation of GUI code, which we believe has a great potential to be the

* Corresponding author.
** Corresponding author.
E-mail addresses: linww@scut.edu.cn (W. Lin), Wentai.Wu@warwick.ac.uk (W. Wu).

https://doi.org/10.1016/j.jnca.2020.102636
Received 18 November 2019; Received in revised form 19 February 2020; Accepted 20 March 2020
Available online 4 April 2020

driving force of swift GUI programming in practice. On the one hand, the coming era of "big data" allows us to access a massive volume of code resources from the Internet as a huge knowledge base for automatic generation of GUI code, which greatly helps us to build a data-driven end-to-end GUI code generation model. On the other hand, the onset of machine learning frameworks provides more opportunities for us to look for appropriate solution to a specific task in the domain of auto-programming. Especially, deep learning techniques have been proven to be effective in capturing complex patterns in big data. More specifically, we can exploit CNN to extract image features with strong expressiveness and apply RNN to the extraction of refined high-level semantic information from the context of long sentences, which makes deep learning possible and promising in automatically generating GUI code. Currently there exist a number of approaches to automatically converting GUI into code and most of them use an encoder-decoder framework with an end-to-end training process. However, there are two common problems that remain unsolved in previous solutions. Firstly, a large portion of the information contained in the GUI as well as the DSL code is hardly fully utilized, which consequently generates poor codes which loss representation for some components and layout of GUI, what is worse, it loss representation for grammar rules too. Secondly, in existing approaches, a lot of DSL code generated by the code generator does not conform to the grammar rules, making the output of the model hardly useful for practical GUI engineering.

These problems inevitably limit the performance of Gui2Code model to generate high quality code. Therefore, in this paper, we propose two methods to enhance their ability to generate code which result into two novel models respectively. Our main contributions are as follows:

1. Inspired by the attention mechanism in neural machine translation and image description, an automatic generation model of GUI code based on mixed attention (HGui2Code) is proposed. To make the best of the information in GUI and DSL code. We innovatively integrate visual attention-enabled extracting GUI features (extracted by CNN) with DSL attention-enabled exacting semantic features (extracted from DSL code by LSTM).
2. To make Gui2Code model generate code that are correct in terms of grammar rules, we use ON-LSTM (Shen et al., 2018) (2018) network to take grammar rules into account and present SGui2Code model, which also makes an increase with 1.5% on the web dataset.
3. We conducted extensive experiments on three datasets to evaluate HGui2Code and SGui2Code. The results show that in the web dataset, our HGui2Code model promotes code generation accuracy by 5.5% compared to the baseline methods while a 1.5% improvement is yielded by the SGui2Code model. As for iOS and Android datasets, our two models generate high quality DSL code which are better representing the layout of GUI.

The rest of this paper is organized as follows. Section 2 is arranged to state related works, we show the general Gui2Code framework in Section 3, then discuss our HGui2Code framework in Section 4 while SGui2Code in Section 5, and we demonstrate our models outperform baselines by contrast experiment in Section 6. Finally, we make a conclusion in Section 7.

2. Related works

2.1. Neural program synthesis

With the rapid development of deep learning, researchers begin to consider whether intelligent systems can solve the problem: program synthesis. Neural program synthesis has a large application market, and successful program synthesis systems can realize automatic machine programming in the future. Using machine learning techniques to automatically generate programs is a new area of research. Recently this research has made breakthroughs. Researchers have proposed a variety of models and architectures to meet multiple challenges in neural programs synthesis.

Graves et al. (2014) (2014) trained a "Neural Turing Machine" (NTM) model directly by the input and output of the program. This model uses deep learning network's powerful expressive ability to simulate real programs and make them behave. Neelakantan et al. (2015) (2015) proposed a Neural Programmer, which is augmented by a small part of basic arithmetic and logic operations that can be trained end-to-end using backpropagation. This neural programmer can call these enhancements in a few steps to generalize a more complex combinatorial program than built-in operations.

In actual research, training data is scarce for everyone. The main issue is how to incorporate prior knowledge into the model. Riedel et al. (2016) (2016) considered the case of a priori procedural knowledge of neural network, and propose to provide an end-to-end and differentiable interpreter, which enables programmers to write draft programs with intermediate missing, these draft programs can be filled from the behavior training of input and output data.

Balog et al. (2016) (2016) proposed a deep learning system DEEPCODER that will automatically program to solve the basic programming problems involved in competitions. Parisotto et al. (2016) (2016) proposed a new technique: Neuro-Symbolic Program Synthesis, which can automatically build computer programs using domain-specific languages. Gupta et al. (2017) (2017) proposed an end-to-end programming error solution system DeepFix, which can locate and fix multiple similar errors in a program without relying on any external tools. Becker et al. (Becker and Gottschlich, 2017) (2017) proposed the first machine learning system AI Programmer, which can automatically generate complete software programs. Based on genetic algorithm, the system is tightly constrained programming language and minimizes expenses while searching. Abolafia et al. (2018) (2018) proposed a new method for program synthesis using a cyclic neural network: Priority Queue Training (PQT). They train a cyclic neural network model on the generated best program data sets, then merge new programs and add them to the priority queue by cyclic neural network sampling. Kant (2018) (2018) summarized the development of recent neural network program generation. Firstly, this article discusses problems and challenges in program synthesis, then reviews the development process of the program induction model, finally compares the differences in program synthesis and introduces potential research directions in the future.

In summary, the ability to model complex programming languages is limited by DSL. Above documents do not consider syntax information in generated code, which leads syntax errors sometimes. To fulfill the task of programming language generation, it is very necessary to introduce grammar rules and syntax into the neural network to obtain a grammatical aware sequence generator.

2.2. Automatic GUI code generation

Although these breakthroughs in neural program synthesis indicate that automatic generation of computer programs is an active area of research. But using machine learning techniques to automatically generate GUI code is still an area of research that has hardly been explored.

Nguyen et al. (Nguyen and Csallner, 2015) (2015) first proposed the Reverse Engineer Mobile Application User Interfaces (REMAUI). In reverse engineering, this technology can get the UI code by inferring from mobile program's screenshots; In the forward engineering, the problem of converting GUI sketch into corresponding code is solved. Based on recent advances in image description and optical character recognition, Deng et al. (2016) (2016) proposed a generic, deep learning-based model to decompile images into representation marker.

This model does not require any knowledge of underlying markup language and only requires end-to-end training on real sample data. Beltramelli (2018) (2017) pioneered attempts to learn latent variables rather than complex engineering heuristics by machine learning. That model uses computer vision to understand the GUI, computer code and generates grammatically and semantically correct samples. Airbnb (Wilkins, 2019–03) (2017) introduced code generation from low fidelity wireframe. This algorithm uses machine learning model to convert product interface sketch into codes directly, and improves the conversion efficiency from sketches to code. Kumar (2018) designed a deep neural network model called Sketch2Code, which translates sketches of website's GUI into code. This model has scored 0.76 on the BLEU. Wallner (2018) published an article on FloydHub about converting a website design to code, and shared how he built a powerful front-end code generation model Screenshot to Code, which based on pix2code and other papers. Microsoft (TechniquesMicrosoft, 2018), Kabel and Spike Techniques (2108) have teamed up to release the Sketch2Code project, a web-based solution that uses AI to convert hand-drawn GUI sketches into useable HTML code. Ellis et al. (2018) (2018) proposed converting a simple hand drawing into a graphical program model written with a subset of LaTeX. This model combines deep learning and program synthesis techniques to train CNN to interpret images' primitives, which are the specifications that graphics programs need to draw. Zhu et al. (2018) (2018) proposed a layered code generation model based on attention mechanism. Tt is called ABHD model in this paper. In order to facilitate the description of the following chapters, this model first extracts enough image features by CNN, and then inputs the image features into two levels of decoders: Block-Level Decoding and Token-Level Decoding.

3. Automatic GUI code generation framework

In the automatic generation of GUI code model, the quality of the generated codes mainly depends on the objects contained in the image, the recognition ability of the scene, and the degree of cognition about the relationship between the objects. Based on encoder-decoder framework, automatic generation of GUI code model combines CNN and RNN. End-to-end method is used to train this model. Generally, the structure is shown in Fig. 1.

Fig. 1 shows the overall model structure in the field of automatic generation of GUI code based on pix2code (Beltramelli, 2018), SketchCode, and Screenshot to Code. The ABHD (Zhu et al., 2018) model innovates on this basis. In this model structure, screenshot I is encoded by the CNN visual model as V, and DSL code is encoded by the language model of the RNN as C_t. Then, the feature vectors obtained by the two coding models are fused to obtain h_t and fed into the decoder composed of RNNs. Finally, a DSL tag y_t is sampled once using the Softmax layer. The size of the output dimension of the Softmax layer corresponds to DSL vocabulary's size.

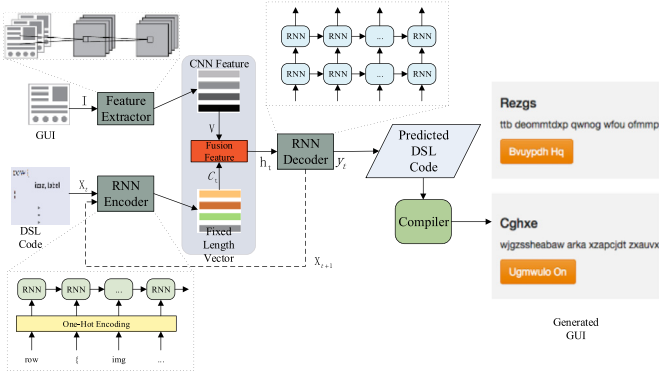


Fig. 1. GUI code auto-generating model structure overview.

As all DSL tags are created, compiler translate DSL code to real source code by taking the DSL code file and the DSL transformation pattern library as input, each label in the DSL code is mapped to the program source code through the pattern analysis framework processing, and the complete Web/Android/IOS program source code is output through the code generation module.

In general, given a GUI and all DSL code generated at the current moment, the model can be optimized end-to-end by gradient descent to predict the next DSL tag in the sequence. At the compile stage, the resulting DSL code sequence is compiled into the desire5d target language using conventional compiler design techniques. The mathematical formula is described as follows:

$$V = CNN(I) \tag{1}$$

$$C_t = \mathrm{LSTM}(X_t) \tag{2}$$

$$h_t = (V, C_t) \tag{3}$$

$$y_t = softmax\left(LSTM'(h_t)\right) \tag{4}$$

$$x_{t+1} = y_t \tag{5}$$

The existing automatic generation of GUI code model can learn a wealth of features, but has three main drawbacks:

(1) Existing models cannot make full use of extracting image features and DSL code features. Because the atomic structure of the program itself has inherent meaning, and the input of the model is not continuous domain space.
(2) RNN decoders sometimes have grammatical errors in decoding. Due to the incorrect DSL code input in the compiler, the compiler will not be able to analyze the source code of the program correctly. Almost all models do not take controlling DSL code syntax into account when generating DSL code.
(3) The vocabulary range in the program is very small, and has almost no semantic relationship between the contexts, which leads to less features extracted from the deep learning model and cannot guarantee the generalization ability.

In response to these problems, we propose HGui2Code and SGui2Code respectively.

4. Integrating with hybrid attention mechanism

We have successfully applied the attention mechanism in image and text in automatic generation of GUI code model, in which the weights of image feature and code semantic vector are automatically determined. Two weight vectors are initialized for image feature vector and code semantic vector respectively, then the weight vector multiplied by corresponding feature vector, finally we concatenate image feature and code semantic vector as a hybrid feature vector, with the execution of gradient descent algorithm, the weight vectors will be adjusted automatically, more details are shown in Section 4.1 and Section 4.2.

4.1. Attention mechanism

The encoder-decoder model is very classical. Its biggest limitation is that the only connection between the encoder and decoder is a fixed length semantic vector. It means that encoders compress information of the entire sequence into a fixed length vector, two problems occur:

(1) Semantic vectors may not be able to fully represent information of the entire sequence.

(2) The information carried by the first input sequence may be covered by the later input sequence. The longer the input sequence is, the more serious this phenomenon is.

Using a vector to represent all the content of an image can cause a partial loss of image information, therefore, to extract features efficiently, we use a soft attention mechanism similar to most image caption and neural machine translation models, for all image regions and DSL at specific times. Labels get different attention, so that LSTM decoder can selectively focus on certain parts of the image by selecting a subset of the image feature vectors, and automatically adjust the DSL code semantic vector by weights.

Taking visual attention mechanism as an example, the computational idea of DSL attention mechanism is similar. The calculation is as follows:

$$p_t = \sum_{i=1}^{L} \alpha_{ti} v_i \tag{6}$$

where p_t represents the feature vector selected by the attention mechanism; $V = \{v_1, \ldots, v_L\}, v_i \in R^D$, represents the dimension of the image feature map, v_i represents one pixel in the feature map. L represents the number of image pixel points; α_{ti} represents the weight corresponding to the time t.

The visual attention mechanism is to generate a set of weights $\alpha_t = \{a_{t0}, \ldots, a_{ti}, \ldots, a_{tL}\}$ corresponding to the feature map V at time t, which is generated as follows:

$$e_{ti} = f_{att}(v_i, s_{t-1}) \tag{7}$$

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})} \tag{8}$$

where s_{t-1} represents the hidden layer state at time t-1 of the LSTM decoder model. e_{ti} represents the correlation between the i feature map vector and the hidden layer state s_{t-1} of the LSTM decoder at time t-1, which can be calculated by a multi-layer perceptron. After getting e_{ti}, pass it to the Softmax function to get the normalized weight value α_{ti}. f_{att} is a multi-layer perceptron, and the mapping relationship is as follow:

$$f_{att} = \beta^T \tanh(Ws_{t-1} + Uv_i) \tag{9}$$

where W is the weight matrix corresponding to s_{t-1}; U is the weight matrix corresponding to v_i; β^T is the weight matrix corresponding to the neuron output.

4.2. Construction of HGui2Code

In order to make full use of the information in the image and the DSL

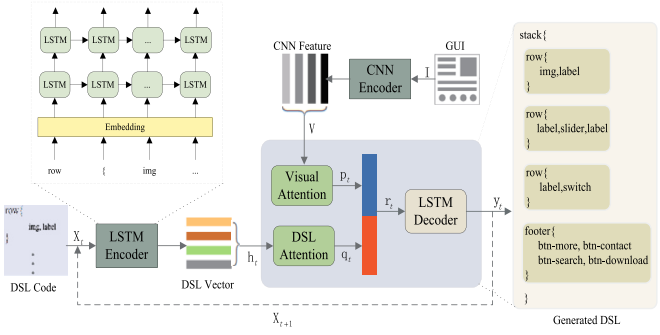


Fig. 2. HGui2Code structure diagram.

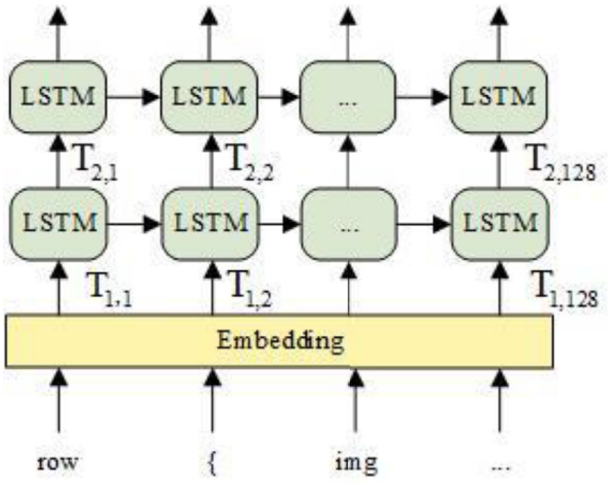


Fig. 3. LSTM Encoder network structure.

encoding, we propose this hybrid attention mechanism for decoding image features and DSL vectors. Combining the semantic features of DSL attention with each regional feature of visual attention, the decoder is guided to generate DSL code. As shown in Fig. 2.

As shown in Fig. 2, the main components are CNN encoder, LSTM encoder, attention mechanism, and LSTM decoder. The working principles are as follows:

(1) **CNN Encoder** It uses CNN as visual model to perform unsupervised features learning. A high-level visual representation V acts as an input to the visual attention component. The CNN encoder consists of a 6-layer convolution layer, a 3-layer maximum pooling layer, a 5-layer Dropout layer, and a 2-layer fully-connected layer. The convolutional layer uses a 3×3 convolution kernel with a step size of 1 to perform a fill-free convolution to get feature map. The pooling layer uses 2×2 pooling window with a step size of 1 to perform the maximum pooling in the feature map without filling. The Dropout layer sets the probability of random

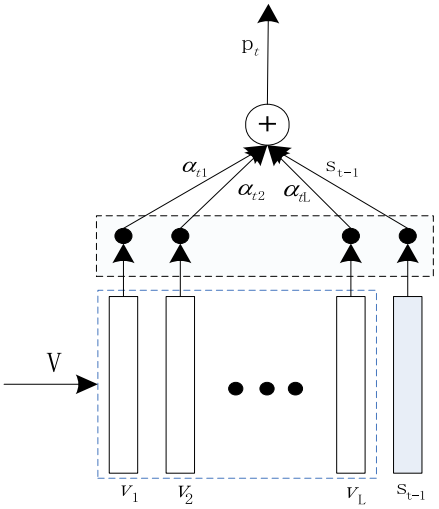


Fig. 4. Visual attention mechanism.

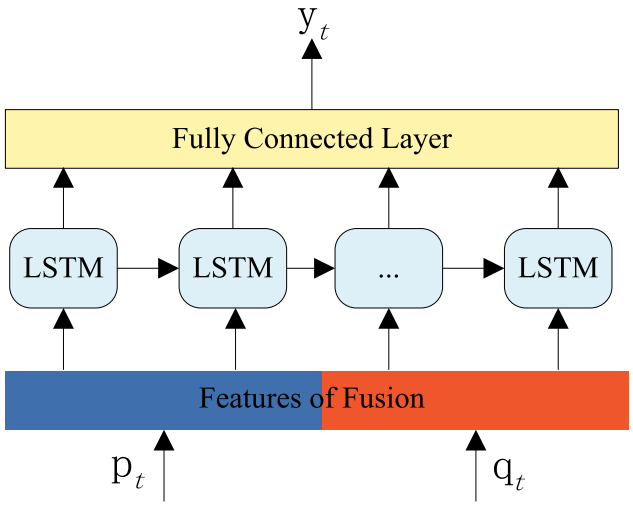


Fig. 5. LSTM decoder structure.

inactivation of neurons to 0.25 or 0.3. The fully connected layer uses *ReLU* as the activation function.

(2) **LSTM Encoder** It can be used as a language model, encoding tags vector with an embedded layer, and performing tag-level language modeling through discrete input. We use LSTM to model DSL code associated with input images. The input of LSTM encoder is the feed-in DSL code at time t, and the output represents the hidden layer state of the LSTM encoder at each step. We use a simple lightweight DSL to describe GUI. The simplicity of DSL not only reduces the size of search space, but also reduces the size of vocabulary. In our work, we are only interested in the layout of GUI, the relationship between graphical components, and ignore the actual text values of components.

The LSTM encoder in the HGui2Code model stacks two LSTM layers, each layer containing 128 storage units. The network structure is shown in Fig. 3.

(3) **Attention Mechanism** It includes visual attention and DSL attention.

Visual attention can adaptively adjust the image feature vector. The input of this component is V and s_{t-1}, V is the feature vector extracted from the image, and s_{t-1} is the hidden state of the LSTM decoder at time t-1 containing semantic information for all DSL tags generated before time t. The implicit relationship between V and s_{t-1} is learned by a multi-layer perceptron, and then a set of weights is generated to automatically adjust the feature vector to p_t. This allows the LSTM decoder to focus on corresponding parts in image based on previously predicted DSL code information, rather than focusing on entire image. The process of calculating visual attention is shown in Fig. 4.

DSL attention can give more attention to the inside tokens not the surrounding tokens, adjusting the focus of attention over time. The input to this component is h_t and s_{t-1}, and the output is q_t. The calculation process is consistent with visual attention.

According to the principle of visual attention, we use the self-adaptive automatic generation of GUI code model AGui2Code, as a baseline model of HGui2Code to explore the impact of adding visual attention components only. The AGui2Code model works are as follows: The DSL features containing grammar and semantic information are combined with image features selected by the adaptive visual attention to jointly guide the decoder to generate the DSL code.

(4) **LSTM Decoder**: First, multi-feature fusion layer is performed to connect encoded vector p_t from visual attention and the encoded vector q_t from DSL attention into a single feature vector r_t. It is then delivered to LSTM decoder. Finally, LSTM decoder predicts possible DSL label y_t. The network structure of LSTM decoder is shown in Fig. 5.

The working principle of the HGui2Code model can be expressed by mathematical formulas as follows:

$$V = CNN(I) \tag{10}$$

$$h_t = LSTM(X_t) \tag{11}$$

$$p_t = \sum_{i=1}^{L} \alpha_{ti} v_i \tag{12}$$

$$q_t = \sum_{j=1}^{T_x} \alpha_{tj} h_{tj} \tag{13}$$

$$r_t = (p_t, q_t) \tag{14}$$

$$x_{t+1} = y_t \tag{15}$$

The automatic code generation model based on mixed attention has the following characteristics:

(1) Using deep vision model to extract image features, combined with a cyclic neural network to build a language model, using end-to-end methods for training.
(2) When using the cyclic neural network to decode image features, an adaptive visual attention mechanism is introduced. The decoder adaptively selects features corresponding to each region in image and makes full use of the information contained in image. At the same time, the DSL attention mechanism is introduced too, and LSTM decoder adjusts focus of attention on DSL semantic vector over time;
(3) Using the multi-feature fusion method, the language model coding features are combined with features of each region in adaptive selection image to jointly guide the LSTM decoder to generate DSL code;
(4) In the decoding stage, the structure of the LSTM decoder is simplified, avoiding degradation of the entire network and improving learning ability effectively.

5. Integrating with syntax awareness

The grammar rules of DSL are introduced into the automatic code generation model through ON-LSTM network. The ON-LSTM decoder network is base on fusion features constructs and creates the grammar tree structure of DSL while generating DSL tokens. We can obtain a sequence generator with syntactic awareness and significantly reduce search space during the training process.

5.1. ON-LSTM model

Natural language is usually presented in the form as sequences, but the language's basic structure is not strictly serialized. Linguists (Sandra and Taft, 2014) believe that serialization structure is controlled by rules

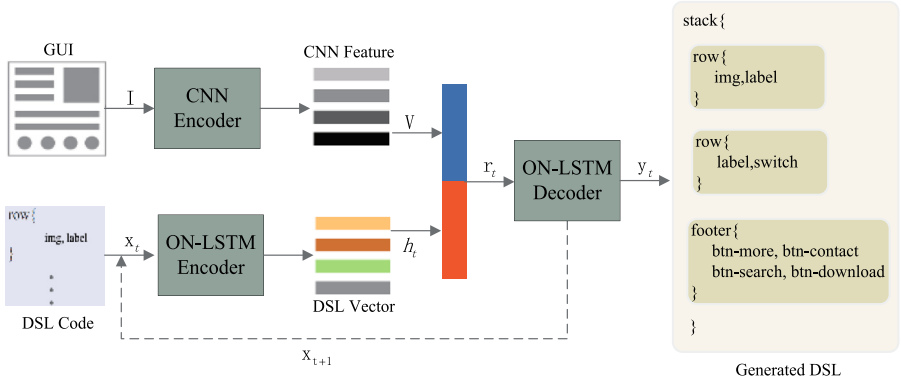


Fig. 6. SGui2Code model.

or grammar of words that make up sentences, and that structure is tree-like. In recent years, deep neural network technology that uses the potential tree structure to form better natural language sentence representation has received great attention. Tai et al. (2015), Chung et al. (2016) integrated the syntax tree structure into language model by gradient back propagation. LeCun et al. (2015), Schmidhuber et al. (Zhang et al., 2018) integrated the tree structure into the language model to obtain hierarchical representations with increasing levels of abstraction. The representation is also a key feature of deep neural networks. Programming languages are very similar to natural languages, and most programming languages are defined by context-free grammars, so programming languages can also be expressed as syntax tree structures. Liu et al. (2018) proposed a new GAN framework, TreeGAN, which incorporates a given context-free grammar into the sequence generation process. Using TreeGAN to solve the syntactic-aware sequence generation problem, the machine generates a large number of high-quality SQL query statements automatically.

Kuncoro et al. (2018) demonstrated that an RNN with sufficient capacity is potential to encode a syntax tree structure implicitly. Shen et al. (2018) proposed a new type of Inductive Bias-Ordered Neurons for cyclic neural networks. This inductive bias enhances the dependence between neurons and is based on order. A new neural network model is designed combining the structure of neuron and LSTM, ON-LSTM. It can perform tree-like composition without destroying sequence form, and the inductive bias is consistent with grammar rules proposed by human experts. In view of the excellent performance of ON-LSTM in language modeling and unsupervised component syntax analysis, we introduce the ON-LSTM model into the field of GUI code automatic generation, and propose a syntax-aware GUI code automatic generation model SGui2Code.

Inductive bias in ON-LSTM model promotes differentiation in information lifecycle stored in each neuron: advanced neurons store long-term information that is retained over a large number of time steps, while low-level neurons store short-term information. In order to avoid a fixed division between high-level and low-level neurons, ON-LSTM proposes a new activation function *cumax()* which actively allocates neurons to store long-term information and short-term information. This function helps ON-LSTM model to generate a vector of advanced input gates and advanced forgetting gates, this vector can ensure that when a given neuron is updated (forgotten), all neurons following it in the sort are also updated (forgotten). ON-LSTM model can process sequence data containing tree structures.

Table 1
Comparing the error rate of HGui2Code model and other models on test sets.

DataSet		Error (%)			
		pix2code	ABHD	AGui2Code	HGui2Code
Pix2Co	Web	12.14	11.50	9.6	6.00
	Android	22.34	18.65	18.87	34.24
	iOS	22.73	19.00	19.20	35.20

5.2. *The working principle of SGui2Code network*

In this network, ON-LSTM decoder based on fusion features constructs and improves the grammar tree structure of DSL code constantly, and obtains grammar awareness sequence generator, which greatly reduces the search space in the training process. The overall network structure of the SGui2Code model is shown in Fig. 6.

SGui2Code model is mainly composed of CNN encoder, ON-LSTM encoder and ON-LSTM decoder. The working principle of each component is as follows:

(1) CNN encoder: It performs unsupervised feature learning using a CNN as a visual model to construct a high-level abstract visual representation of images. V is an image feature extracted by this encoder.

(2) ON-LSTM encoder: Language modeling of DSL code associated with input image using ON-LSTM. The ON-LSTM network learns complex underlying syntax information and syntax patterns from the input DSL code. In order to extract from the encoder hidden layer state h_t containing the syntax tree information, the neurons in ON-LSTM are divided into two parts: high-level neurons and low-level neurons. Advanced neurons store long-term dependent information, and low-level neurons store short-term dependent information.

(3) ON-LSTM decoder: First, multi-feature fusion is performed, visual encoded vector V and language encoded vector h_t are connected into a single feature vector r_t, which is then delivered into ON-LSTM decoder. h_t contains rich syntax and semantic information, corresponding to this, we also built a syntax-aware decoder. The decoder is affected by DSL language knowledge and syntax tree when generating DSL tokens. Therefore, ON-LSTM network is also used in the decoding process and predicts possible DSL label y_t.

6. Experimental results and analysis of the model

To explore the generality of our models, what codes do they generate and how close the generated code is to the actual code, in this part, we analyze the experiment results of our two models in detail. Finally, by comparing error rate of baselines including pix2pix model (Beltramelli, 2018) (2017), ABHD (Zhu et al., 2018) (2018) model, we prove that our models are effective.

6.1. Experimental setup

6.1.1. DataSet

We use a common data set provided by Beltramelli et al. (Beltramelli, 2018), which is referred to as Pix2Co hereafter.

Pix2Co is created for research on GUI and the corresponding DSL code. It consists of three sub-data sets, corresponding to three platforms: iOS, Android and Web. Each subset has 1500 GUI code pairs for training and 250 pairs for testing. The GUI in Pix2Co is a screen shot of the program generated by the boot framework. Each GUI has an average of 65 tags, with 96,925 training samples per sub-data set. Fig. 7 shows example of the iOS GUI and DSL code in the pix2co sub-data set.

6.1.2. Evaluation metrics

This paper evaluates the proposed models' effectiveness in generating DSL code in terms of accuracy based on the metric of error rate.

The error rate is defined as the ratio of the number of samples with classification errors to the total number of samples. For sample set D, the classification error rate is defined as:

$$Error(x,y)=\frac{1}{D.l}\sum_{x,y\in D}F(f(x),y) \tag{16}$$

$$F(w,y)=\begin{cases}1, w.l\neq y.l\\ \frac{1}{y.l}\sum_{i=1}^{y.l}C(w[i],y[i]), w.l=y.l\end{cases} \tag{17}$$

$$C(k,v)=\begin{cases}0, k=v\\ 1, k\neq v\end{cases} \tag{18}$$

We assume that x and y is a pair sample of D, x represent GUI while y is ground truth DSL code. Formula 16 caculates the error rate between the predicted DSL code $f(x)$ and the groundtruth y, $D.l$ represent the number of elements in D. Formula 17 shows caculation's function F, if predicted DSL code's length $w.l$ ia not equal to groundtruth' length $y.l$, F return 1, else F compare w and y code tag by code tag with function C, and count the proportion of inconsistent code tags.

This error rate is used for quantitative analysis later, we also have qualitative analysis by analyzing GUI and DSL code generated by our models.

6.2. Evaluating the HGui2Code model

6.2.1. Visual attention analysis

As we can see from Fig. 8, after the HGui2Code model generates a DSL tag, its attention focus moves. The brighter area in image represents the current location of interest. Therefore, through visual attention our model can learn the correct alignment of DSL tags with their corresponding spatial areas in GUI.

6.2.2. Quantitative comparison

We select pix2code model proposed by Beltramelli (2018) and the ABHD model proposed by Zhu et al. (2018) as the baseline models. These two models use the same CNN encoder network structure to extract image features and experimental data is based on the Pix2Co data set.

Table 2
Comparing the error rate of SGui2Code model and other models on test sets.

Dataset		Error (%)		
		pix2code	ABHD	SGui2Code
Pix2Co	Web	12.14	11.50	10.00
	Android	22.34	18.65	22.29
	iOS	22.73	19.00	22.80

As shown in Table 1, on the Pix2Co data set, the error rates of HGui2Code model and AGui2Code model in Web data set are 6.0% and 9.60% respectively, which are lower than pix2code model and ABHD model. The error rates of AGui2Code model in the Android and iOS sub-data sets are 18.87% and 19.20% respectively, which are lower than pix2code model and basically at the same level with ABHD model. The HGui2Code model cannot perform well on the Android and iOS sub-data sets, producing error rates of 34.24% and 35.20% respectively. The experimental results show that adaptive visual attention mechanism in the AGui2Code model can extract useful image feature representations from GUI and guides LSTM decoder to generate high quality DSL code. However, the hybrid attention mechanism in the HGui2Code model does not extract useful feature representations from Android, iOS GUI and DSL code, resulting in poor quality DSL code generated by the LSTM decoder.

Since HGui2Code model is not as good as AGui2Code model in Android and iOS sub-data set, it indicates that DSL code and visual attention are not spatially aligned in that two data sets. Comparing with web dataset, images in Android and iOS sub-data sets are smaller, which make it hard to control the alignment. If the time generator pays attention to a certain graphic area in the GUI, while the DSL attention does not pay attention to the corresponding area in the DSL code, which causes the HGui2Code LSTM decoder to input misplaced image and code information when generating the DSL token.

6.2.3. Qualitative analysis of experimental results

In order to compare the qualitative results obtained by the HGui2Code, pix2code and ABHD models better, we compare real GUI with DSL code generated by the HGui2Code model, as shown in Fig. 9.

As shown in Fig. 9, the HGui2Code model has problems about misalignment and some graphic element style errors, but the result is close to actual sample. This proves that HGui2Code can better learn a simple GUI layout. Comparing the DSL code or GUI on Android platform generated by each model in (Beltramelli, 2018) (Zhu et al., 2018), many complex graphic elements in GUI training sample are hardly appearing in GUI code generated by HGui2Code. This may be due to the lack of alignment of DSL attention and visual attention. According to our analysis, because the two attention use different weight vectors respectively, and the two weight vectors lack of contact and perform their respective duties, which may lead to the two attention not being aligned in space. One possible solution is to increase the contact or share the weight values for the two weight vectors to achieve synchronous alignment. Based on current results, the HGui2Code model has no obvious advantages on mobile platform.

6.3. Evaluating the SGui2Code model

6.3.1. Qualitative analysis of experimental results

The comparison model is similar to Section 6.2.2, All baseline models use the same CNN network structure and the experimental data is based on the Pix2Co data set. The experimental results are shown in Table 2.

As shown in Table 2, SGui2Code model outperforms all the baseline models in the Web sub-data set but struggles on the Android data set with

an error rate of 3.64% higher than that of the ABHD model, and on the iOS sub-dataset its error reaches 22.80% - higher than the 22.73% of pix2code model and the 19.0% of ABHD model.

The experimental results show that ON-LSTM encoder in SGui2Code model can learn complex underlying syntax information and syntax patterns from DSL code. When encoding DSL code, the internals of the ordered neurons construct potential syntax tree information. During the decoding process, ON-LSTM network continuously constructs and refines grammar tree structure of DSL code. It is possible to have enough syntax information to decoder to generate DSL tags that conform to syntax semantics.

6.3.2. Qualitative analysis of experimental results

This section compares real GUI and the GUI generated by models from a visual perspective. Since the models such as pix2code, ABHD, HGui2-Code, and SGui2Code ignore the actual text values in the image, the compiler randomly assigns text content in the label when rendering the DSL code to the GUI. In order to better compare the qualitative results obtained by SGui2Code and pix2code, ABHD, we render the DSL code generated by the SGui2Code model into a GUI, as shown in Fig. 10.

Through the generated result in Fig. 10, we can see that the real GUI contains a long source code sequence, so the cyclic neural network needs to rely on previously hidden layer state information for a long time in decoding stage. Because ON-LSTM network in the SGui2Code model can not only learn the grammatical information in the sequence, but also determine life cycle of information stored in neurons and enhance dependence between neurons. This model is superior to the standard LSTM model in terms of long-term dependency and generalization performance of long sequences. Therefore, it can be concluded that SGui2-Code model is effective in solving the problem of long-term dependency of sequences.

7. Conclusion and future work

This paper mainly studies the application of convolutional neural network and recurrent neural network in the automatic generation of GUI code. Although ordinary neural network encoders can extract features from images and DSL code, interpreting these abstract features into hundreds of code tokens pose challenges to the decoding capabilities of decoder. In view of this, On the one hand, the atomic structure of the program itself has inherent meaning, and the input of the model is not continuous domain space, original automatic GUI code generation framework cannot make full use of the hidden information in GUI and DSL code, therefore we propose the HGui2Code model using attention-based deep neural network to align DSL code with corresponding GUI pixels by hybrid attention mechanism. This framework yields an increase 5.5% in web dataset compare to the best baselines. On the other hand, in contrast to existing automatic GUI code generation frameworks that generally decode fusion features into DSL code without considering grammar rules, we take advantage of the ON-LSTM and propose the SGui2Code model to make the generated DSL code compliant with grammar rules which results in a 1.5% increase of accuracy in the web dataset.

We plan to carry out further research in the future mainly in the following aspects:

(1) The size and position recognition of components in GUI. The current method can only identify the interface components (such as button, input box, etc.) in GUI, but cannot the text, color and their locations.
(2) Alignment issues in attention area. Since there is a correspondence between GUI and DSL code, their alignment needs more control.
(3) We use a fixed way to mix visual attention and DSL attention, it is not flexible enough. When using hybrid attention, it is necessary to study how to dynamically set the fusion method in different steps in code generation.
(4) Code generator with DSL syntax awareness. RNN can model the dependencies between components in natural language, but the dependency between DSL grammatical components is much longer than the dependency between natural components. Therefore, we need to study how to introduce DSL syntax rules into the code generator.

Declaration of competing interest

We declare that we have no financial and personal relationships with other people or organizations that can inappropriately influence our work, there is no professional or other personal interest of any nature or kind in any product, service and/or company that could be construed as influencing the position presented in, or the review of, the manuscript entitled"Syntax-aware AutomaticGraphicsProgram Generation with Attention-based Deep NeuralNetwork".

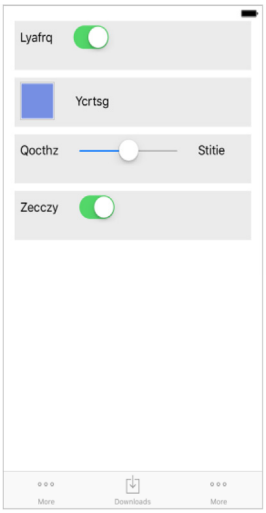
CRediT authorship contribution statement

Xiongwen Pang: Writing - original draft, Methodology. **Yanqiang Zhou:** Data curation. **Pengcheng Li:** Writing - original draft. **Weiwei Lin:** Writing - review & editing, Supervision, Funding acquisition. **Wentai Wu:** Writing - review & editing. **James Z. Wang:** Supervision.

Acknowledgements

This work is supported by National Natural Science Foundation of China (Grant Nos. 61772205, 61872084), Guangzhou Science and Technology Program key projects (Grant Nos. 201902010040 and 201907010001), Key-Area Research and Development Program of Guangdong Province (Grant No. 2020B010164003), Nansha Science and Technology Projects (Grant No. 2017GJ001), Guangzhou Development Zone Science and Technology (Grant No. 2018GH17), the Fundamental Research Funds for the Central Universities, SCUT (Grant No. 2019ZD26), and Major Program and of Guangdong Basic and Applied Research (Grant No. 2019B030302002). James Z. Wang's is partially supported by NSF grant #1759856 and NIH grant #2R01HD069374-0681.

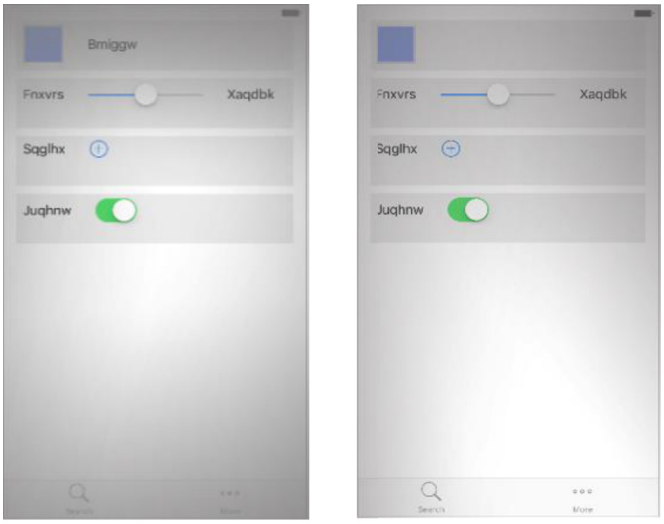
Appendix




```
stack {
  row {
    label, switch
  }
  row {
    img, label
  }
  row {
    label, slider, label
  }
  row {
    label, switch
  }
}
footer {
  btn-more, btn-download, btn-more
}
```

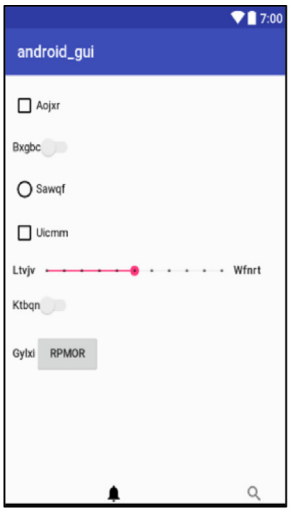
(a) iOS GUI screenshot (b) DSL code

Fig. 7. IOS GUI with DSL.



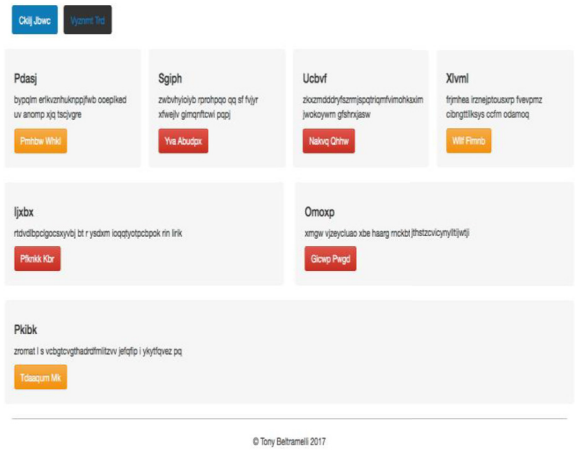
(1) switch (2) stack

Fig. 8. Attention to image position that changes over time.




```
stack {
   row {
      check
   }
   row {
      switch
   }
   row {
      radio
   }
   row {
      check
   }
   row {
      label, slider, label
   }
   row {
      switch, btn
   }
   row {
      label
   }
}
footer {
   btn-notifications, btn-search
}
```

(a) Real GUI and Generated DSL by HGui2Code in Android

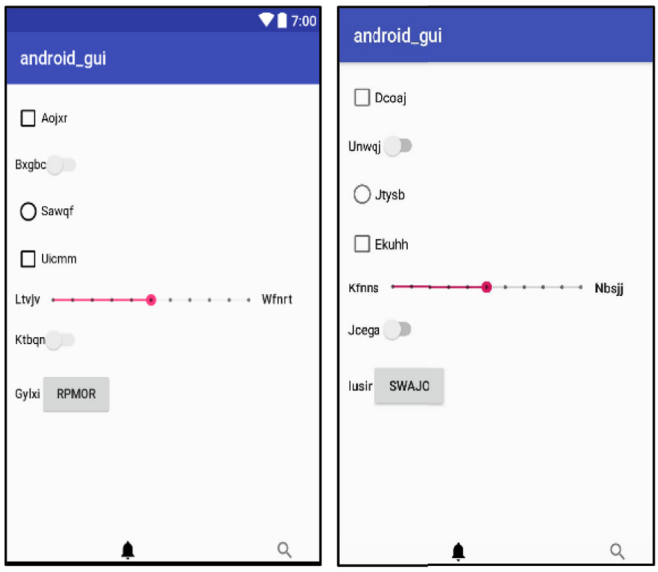



```
header{
   btn-active, btn-inactive
}
row{
   quadruple{
      small-title, text, btn-orange
      }
   quadruple{
      small-title, text, btn-orange}
   quadruple{
      small-title, text, btn-orange}
   quadruple{
      small-title, text, btn-orange}
   }
row{
   double{
      small-title, text, btn-orange
      }
   double{
      small-title, text, btn-orange}
   }
row{
   single{
     small-title, text, btn-orange}
}
```

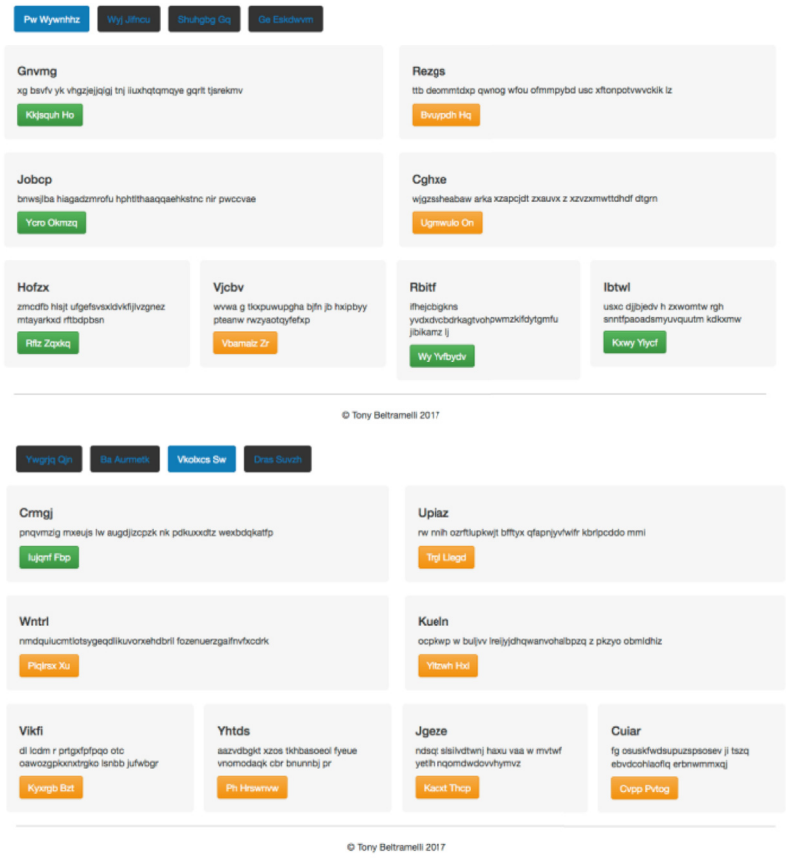
(b) Real GUI and Generated DSL by HGui2Code in Web

Fig. 9. The HGui2Code model generates DSL codes corresponding to the three platform GUIs.

corresponding to the three platform GUIs.



(a) Real GUI (left) and Generated GUI (right) by SGui2Code in Android



(b) Real GUI (above) and Generated GUI (below) by SGui2Code in Web

Fig. 10. Comparison of real GUI and SGui2Code model generation GUI.

References

Abolafia, D.A., Norouzi, M., Shen, J., et al., 2018. Neural Program Synthesis with Priority Queue training[J] arXiv preprint. 1801.03526.

Balog, M., Gaunt, A.L., Brockschmidt, M., et al., 2016. Deepcoder: Learning to Write programs[J] arXiv preprint. 1611.01989.

Becker, K., Gottschlich, J., 2017. AI Programmer: Autonomously Creating Software Programs Using Genetic Algorithms[J] arXiv preprint. 1709.05703.

Beltramelli, T., 2018. pix2code: generating code from a graphical user interface screenshot[C]. In: Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems. ACM, vol. 3.

Chung, J., Ahn, S., Bengio, Y., 2016. Hierarchical Multiscale Recurrent Neural networks [J] arXiv preprint. 1609.01704.

Deng, Y., Kanervisto, A., Rush, A.M., 2016. What You Get Is what You See: A Visual Markup decompiler[J], vol. 10, pp. 32–37 arXiv preprint. 1609.04938.

Ellis, K., Ritchie, D., Solar-Lezama, A., et al., 2018. Learning to infer graphics programs from hand-drawn images[C]. In: Advances in Neural Information Processing Systems, pp. 6060–6069.

Graves, A., Wayne, G., Danihelka, I., 2014. Neural Turing machines[J] arXiv preprint. 1410.5401.

Gupta, R., Pal, S., Kanade, A., et al., 2017. Deepfix: fixing common c language errors by deep learning[C]. In: Thirty-First AAAI Conference on Artificial Intelligence.

Kant, N., 2018. Recent Advances in Neural Program Synthesis[J] arXiv preprint. 1802.02353.

Kuncoro, A., Dyer, C., Hale, J., et al., 2018. LSTMs can learn syntax-sensitive dependencies well, but modeling structure makes them better[C]. In: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), vol. 1, pp. 1426–1436.

LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep Learning. Nature 521[J].

Liu, X., Kong, X., Liu, L., et al., 2018. TreeGAN: syntax-aware sequence generation with generative adversarial networks[C]. In: 2018 IEEE International Conference on Data Mining (ICDM). IEEE, pp. 1140–1145.

Neelakantan, A., Le, Q.V., Sutskever, I., 2015. Neural Programmer: Inducing Latent Programs with Gradient descent[J] arXiv preprint. 1511.04834.

Nguyen, T.A., Csallner, C., 2015. Reverse engineering mobile application user interfaces with remaui (t)[C]. In: Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on. IEEE, pp. 248–259.

Parisotto, E., Mohamed, A., Singh, R., et al., 2016. Neuro-symbolic Program synthesis[J] arXiv preprint. 1611.01855.

Riedel, S., Bosnjak, M., Rocktäschel, T., 2016. Programming with a Differentiable Forth interpreter[J]. CoRR, abs/1605.06640.

Sandra, Dominiek, Taft, Marcus, 2014. Morphological Structure, Lexical Representation and Lexical Access (RLE Linguistics C: Applied Linguistics): A Special Issue of Language and Cognitive Processes. Routledge.

Shen, Y., Tan, S., Sordoni, A., et al., 2018. Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks[J] arXiv preprint. 1810.09536.

Tai, K.S., Socher, R., Manning, C.D., 2015. Improved Semantic Representations from Tree-Structured Long Short-Term Memory networks[J] arXiv preprint. 1503.00075.

Techniques, Spike, Microsoft, Kabel. Turn whiteboard UX sketches into working HTML in seconds [EB/OL]. https://www.ailab.microsoft.com/experiments/sketch2code,%202018-08-23/2019-03-29.

Wilkins, B., 2019-03-29. Sketching Interfaces Generating Code from Low Fidelity Wireframes [EB/OL]. https://airbnb.design/sketching-interfaces/.

Zhang, W., Wulan, G., Zhai, J., et al., 2018. An intelligent power distribution service architecture using cloud computing and deep learning techniques[J]. J. Netw. Comput. Appl. 103, 239–248.

Zhu, Z., Xue, Z., Yuan, Z., 2018. Automatic Graphics Program Generation Using Attention-Based Hierarchical Decoder[J] arXiv preprint. 1810.11536.

Xiongwen Pang received his Bachelor from ChongQing Univeristy in 1994, M.S. degrees from Harbin Institute of Technology in 1996., and the PhD degree in Computer Ap-plication from South China University of Technology in 2007. Currently, he is an associate professor in the School of Computer Science, South China Normal University. His research interests include big data, deep learning and AI application technologies. He has published more than 20 papers in refereed journals and conference proceedings. He is a senior member of CCF.

Yanqiang Zhou is a graduate candinate supervised by associate professor Xiongwen Pang with the Department of Computer Science South China Normal University. His research interests mainly include deep learning.

Pengcheng Li received his Bachelor in computer science from South China Normal University in 2019. Currently, he is a graduate candinate supervised by associate professor Xiongwen Pang with the Department of Computer Science South China Normal University. His research interests mainly include deep learning.

Weiwei Lin received his B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and the PhD degree in Computer Ap-plication from South China University of Technology in 2007. Currently, he is a professor in the School of Computer Science and Engineering, South China University of Technology. His research interests include distributed systems, cloud computing, big data computing and AI application technologies. He has published more than 80 papers in refereed journals and conference proceedings. He is a senior member of CCF.

Wentai Wu received his Bachelor and Master degrees in computer science from South China University of Technology in 2015 and 2018, respectively. Currently, he is a Ph.D. candidate supervised by Dr. Ligang He with the Department of Computer Science, the University of Warwick, United Kingdom. His research interests mainly include parallel and distributed computing, distributed learning, energy-efficient computing and predictive analytics.

James Z. Wang received his B.S. and M.S degrees in Computer Science from University of Science and Technology of China. He obtained his Ph.D. degree in Computer Science from University of Central Florida. He is currently a professor in the School of Computing at Clemson University, South Carolina. His research includes storage network, database system, distributed system, cloud computing and multimedia technologies. Dr. Wang is a senior member of IEEE and ACM. IEEE and ACM.