***Journal of Artificial Intelligence and Technology*, 2024, 4, 325-331**
https://doi.org/10.37965/jait.2024.0515

ISTP
RESEARCH ARTICLE

Automatic Code Generation for Android Applications Based on Improved Pix2code

Donglan Zou[1,2] **and Guangsheng Wu**[1]
[1]School of Mathematics and Computer Science, Xinyu University, Xinyu, China
[2]College of Computing, Informatics and Mathematics, UiTM, Malaysia

(Received 26 February 2024; Revised 03 June 2024; Accepted 03 June 2024; Published online 15 July 2024)

***Abstract*:** With the expansion of the Internet market, the traditional software development method has been difficult to meet the market demand due to the problems of long development cycle, tedious work, and difficult system maintenance. Therefore, to improve software development efficiency, this study uses residual networks and bidirectional long short-term memory (BLSTM) networks to improve the Pix2code model. The experiment results show that after improving the visual module of the Pix2code model using residual networks, the accuracy of the training set improves from 0.92 to 0.96, and the convergence time is shortened from 3 hours to 2 hours. After using a BLSTM network to improve the language module and decoding layer, the accuracy and convergence speed of the model have also been improved. The accuracy of the training set grew from 0.88 to 0.92, and the convergence time was shortened by 0.5 hours. However, models improved by BLSTM networks might exhibit overfitting, and thus this study uses Dropout and Xavier normal distribution to improve the memory network. The results validate that the training set accuracy of the optimized BLSTM network remains around 0.92, but the accuracy of the test set has improved to a maximum of 85%. Dropout and Xavier normal distributions can effectively improve the overfitting problem of BLSTM networks. Although they can also decrease the model's stability, their gain is higher. The training and testing accuracy of the Pix2code improved by residual network and BLSTM network are 0.95 and 0.82, respectively, while the code generation accuracy of the original Pix2code is only 0.77. The above data indicate that the improved Pix2code model has improved the accuracy and stability of code automatic generation.

***Keywords*:** automatic code generation; deep learning; long short-term memory network; Pix2code; residual network

I. INTRODUCTION

Since the birth of the World Wide Web in 1991, the Internet has officially entered the era of explosion. The emergence of the World Wide Web makes it possible for the popularization of the Internet, and people can obtain all kinds of information through browsers. In the 21st century, the Internet has become more convenient and efficient. The popularity of intelligent devices has expanded people's network use scenarios. Today, the Internet is becoming an indispensable part of human life. According to statistics, by 2022, the global Internet users have reached 5.16 billion, and the Internet penetration rate is 70.0%, of which the mobile Internet penetration rate has reached 63.5%. As one of the three elements of mobile Internet, applications can provide services, contents, and entertainment for mobile Internet users [1,2]. When developing applications, programmers need to write code based on design documents and requirements. However, coding often involves a lot of repetitive work and requires a lot of time, which leads to low efficiency of software development, difficulty in operation and maintenance, and difficulty in meeting the expanding Internet market. Code automatic generation (CAG) technology can effectively reduce repetitive code writing work, while making it easier to modify and add components, making it convenient for later operation and maintenance. The traditional CAG methods are divided into template-based CAG and model-driven CAG. Although the former has high flexibility and is easy to develop templates, the generated program has poor readability a high dependence on programming languages and frameworks, and template files need to be updated during later updates [3,4]. Although the latter has good maintainability and portability compared to template generated code, model development is difficult, and there is a lot of redundant code generated in the program, which can only generate skeleton code. This indicates that traditional CAG methods fundamentally improve the efficiency of software development, while also increasing the cost of software development. Pix2code is a kind of CAG framework based on deep learning that can automatically generate code according to the design requirements. Pix2code can train a single GUI image into a domain-specific language (DSL) through neural networks and pattern recognition techniques and then convert the DSL into code. Its performance is good, but its accuracy in code generation needs to be improved. Therefore, to improve the code generation accuracy (CGA) of the Pix2code model, this study utilizes residual network (ResNet) and bidirectional long short-term memory (BLSTM) to improve the model. The innovation of this study lies in the use of ResNet to improve the visual module of the Pix2code to enhance the model's feature extraction ability. Moreover, the language module and decoding layer of the Pix2code are strengthened adopting BLSTM to improve the model's word prediction ability and robustness.

The rest of the paper is organized as follows: Part 1 is a literature review with brief studies on ResNet and BLSTM. Part 2 examines the improved Pix2code model. Part 3 analyzes the

Corresponding author: Guangsheng Wu (e-mail: 18107903152@163.com).

experimental results of the improved Pix2code. Part 4 summarizes the research findings throughout this paper.

II. RELATED WORKS

Compared with traditional deep convolutional neural networks (CNNs), ResNet has the advantages of optimizing easily and the capacity to lift accuracy by lifting a considerable depth. It can effectively alleviate the problem of gradient vanishing and is widely utilized in image recognition. Wang C *et al.* proposed a subdomain adaptive transfer learning based on ResNet for bearing fault diagnosis. This method could achieve accurate diagnosis of bearing faults in the presence of small training samples and noise [5]. Deelip MS and Govinda K proposed a disease and pest identification model based on the exponential sunflower knight optimization algorithm and deep ResNet for plant disease monitoring. The maximum recognition accuracy of this model was 0.95, with a maximum throughput of 7533350 bps, a minimum energy of 49.74 J, and a maximum true positive rate of 0.956, which was superior to other algorithms [6]. Chengdong L and his team proposed a fault diagnosis model for air handling units in air conditioning systems based on probabilistic slow feature analysis (PSFA) and attention ResNet. In this model, PSFA was used for feature extraction to suppress noise interference, and attention ResNet was used to construct a fault diagnosis classifier. This model could effectively improve the accuracy of fault diagnosis [7]. Wei M proposed a face recognition algorithm based on block-centered symmetric local binary pattern and deep ResNet for face recognition problems in uncontrolled environments. This algorithm utilized block-centered symmetric local binary patterns to extract facial features and then utilized DRN for facial recognition. This algorithm fundamentally solved the problem of facial identity recognition in uncontrollable environments [8]. Bansal G *et al.* designed a lung cancer classification framework built on ResNet to address the issues of lung cancer classification and image segmentation. The depth features and manual descriptors of this framework were extracted using micro-ResNet and morphological techniques, respectively, with a segmentation accuracy of 0.927 and a matching accuracy of 0.93 [9].

Compared to LSTM, BLSTM can achieve information encoding from back to front, better capturing bidirectional semantic dependencies, and is widely used in various fields. Wang Z and Wang Y proposed a weighing model based on BLSTM and attention mechanism for the dynamic weighing problem of bridges. Compared to other models, this model had a higher accuracy in estimating bridge axle load [10]. Du W's team proposed a recommendation algorithm based on knowledge attention mechanism and BLSTM to address the issue of patent transaction recommendation. This algorithm captured sequential patterns in company history records through BLSTM and utilized attention mechanisms to aggregate company history patents for a given candidate patent. The F1 and normalized discounted cumulative returns of this algorithm have significantly improved [11]. Jiao M and Wang D proposed a state of charge estimation model based on Savitzky–Golay filter and BLSTM for lithium-ion batteries. This model estimated the state of charge of the battery through BLSTM and processes the estimation results through Savitzky–Golay filter. Through experiments, the estimation accuracy of this model was significantly higher than other models, and it had strong robustness against random noise [12].

In terms of visual programming technology, it can significantly reduce the time-consuming and tedious programming work. For example, Microsoft builds a power application using AI Copilot, whose code is generated by AI Copilot using the workflow. Hu Z *et al.* designed an open-source visualization programming solution based on Asyncflow, which includes a flowchart generator interpreted by game logic and an asynchronous mechanism runtime framework based on an event-driven architecture [13]. In response to the issue of building user interface systems, Ren S *et al.* studied that in this environment, users can use a Web browser to process DSLs (such as FAUST or Gen) with audio for graphical design and operation of digital signal processor algorithms. These algorithms were executed in a dedicated high-priority thread called AudioWorklet [14]. Although the above visual programming techniques can convert images into code, their accuracy is difficult to guarantee, and their conversion performance is poor when faced with more complex code. Pix2-Code can convert graphic user interface screenshots created by designers into computer code. Moreover, on three different platforms (iOS, Android, and web-based technology), its accuracy has reached over 77% and it has high conversion efficiency. However, there are still problems such as poor code quality, errors or redundant code, and its conversion speed needs to be further improved.

In summary, ResNet can effectively alleviate the problem of gradient vanishing and has advantages in image processing. Compared to BRNN, BLSTM can better handle the problems of vanishing and exploding gradients, and it can better capture bidirectional semantic dependencies compared to LSTM. Therefore, it is widely used in the processing of sequence problems. The essence of the Pix2code CAG model is image processing and sequence problem processing. Therefore, to perfect the Pix2code performance, this study utilizes ResNet and BLSTM to improve its accuracy and stability.

III. A CAG FRAMEWORK BASED ON IMPROVED Pix2code

In the process of program development, coding is a very complex task. Due to the fact that code writing often involves a lot of repetitive work, it can consume a lot of effort from programmers. Pix2code, as a neural network tool that directly generates code from UI screenshots, can convert software design diagrams into executable code and is compatible with iOS, Android, and web interfaces. However, its accuracy in code generation is relatively low. Therefore, to improve the accuracy of code generation, this study proposes an improved Pix2code by introducing ResNet and BLSTM.

A. KEY TECHNOLOGIES OF CAG FRAMEWORK

Pix2Code can use the software design diagram to convert it into the executable code, which is mainly composed of the visual model and the code generation model, in which the visual model generally uses CNN to extract the image features. Taking the sequential structure as an example, the code generation method of Pix2Code is shown in Fig. 1.

As can be seen from Fig. 1, activities A, B, and C are sequential relationships. Activity B consists of input, output, and execution, corresponding to the execution entry action in the code, execution exit action, and Guard1 to GuardN execution internal action. As one of the classic algorithms in deep learning, CNN has strong representation learning ability and can perform

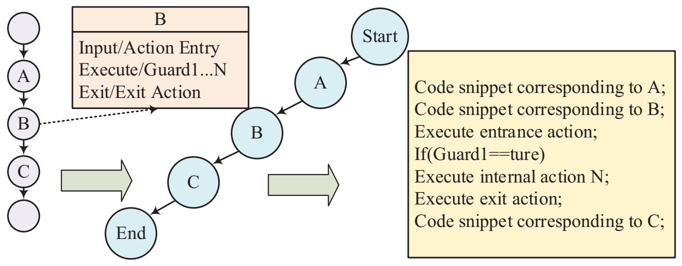

Fig. 1. The code generation method of Pix2code.

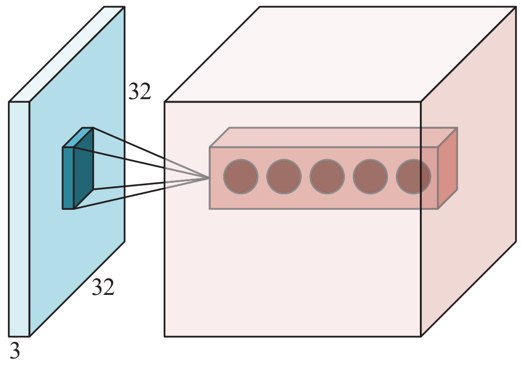

Fig. 3. Schematic representation of the local connection.

translation invariant classification of input information according to its hierarchical structure. At the same time, in the training process of images with higher data dimensions, it can use convolutional layers to achieve local connections and parameter sharing, and it uses pooling layers to achieve data dimensionality reduction. In CNN, convolutional layers, as the most special structure, can extract data features and achieve local connections. Taking 2D convolution as an example, the schematic diagram of convolution operation is Fig. 2.

In Fig. 2, assuming the input image is a matrix of 4×4 and the convolution kernel size is 2×2, the convolution kernel can only perceive regions equal in size to the convolution. The feature matrix can be obtained by performing a 2×2 matrix dot multiplication operation between the convolutional kernel and the images within the interval. When the convolution step size > 1, the output result is equivalent to downsampling at a step size of 1 [15–18]. In convolutional layers, the two main characteristics are weight sharing and local connections. Weight sharing refers to a convolutional kernel that uses consistent parameters when convolving different regions. Local connection refers to the fact that the convolutional kernel only connects to a portion of the previous layer during convolution operations. Figure 3 shows the local connection.

In Fig. 3, during the convolution process, the convolution kernel is only connected to a local area of equal size in the previous layer, which is called the perceptual domain. This is because when determining the attributes of a certain area of an image, pixels with close proximity and strong correlation are the most important. Therefore, the convolutional kernel only needs to detect pixels with strong correlation in the perceptual domain to achieve feature extraction. As a structure for further extracting features from convolutional results, the pooling layer can reduce the dimensionality of the feature matrix and expand the perceptual domain. The schematic diagram of the pooling process is Fig. 4.

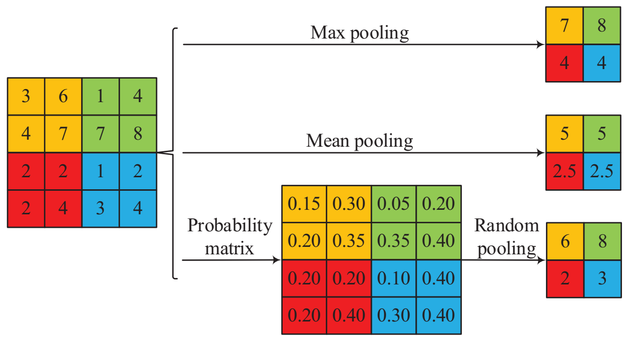

Fig. 4. Schematic representation of the pooling process.

In Fig. 4, the pooling process is divided into three types: maximum pooling, average pooling, and probability pooling. Maximizing pooling is the process of dividing a 4×4 matrix into 4 equal blocks, with each equal block having its max-value as the final output. Average pooling is taking the mean value of each equal block as the final output. Probability pooling is the process of assigning probabilities to pixels within a domain based on their numerical values and then performing subsampling based on these probabilities. In terms of image processing, maximum pooling yields better texture extraction results, average pooling is more conducive to background preservation, and probability pooling falls between the two [19,20]. Although CNN can design diagrams for learning, it is difficult to understand DSL code, while LSTM can understand DSL code well. Therefore, this study introduced LSTM in Pix2code. Figure 5 shows the LSTM structure.

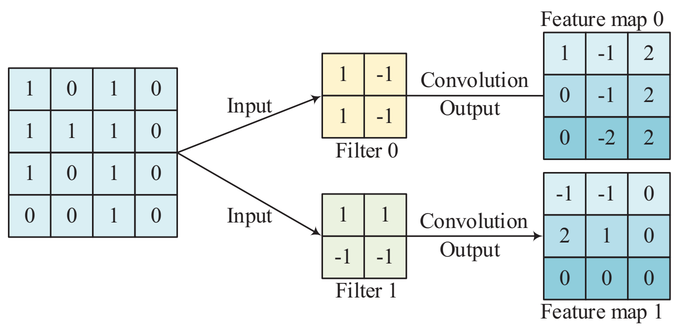

Fig. 2. Schematic diagram of the convolution operation.

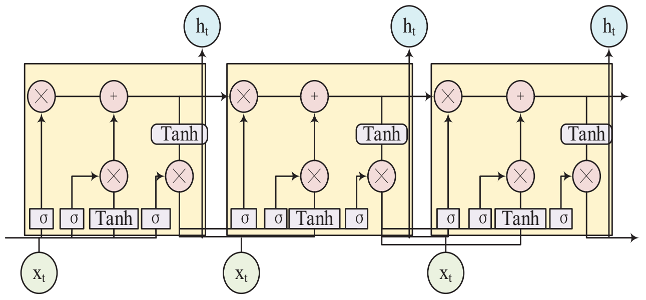

Fig. 5. Structure of the LSTM.

In Fig. 5, the core structure of LSTM can be divided into four parts, namely forget gate, input gate, output gate, and cell state. The forget gate can concatenate the input of the current time step with the hidden state of the previous time step and then it transforms it through a fully connected layer (FCL) and activates it with a sigmoid function to obtain the gate value. The gate value acting on the cell state of the previous layer will determine how much information is forgotten [21–23]. The representation of the forgetting gate is equation (1):

$$f_t = \sigma(w_f * [h_{t-1}, x_t] + b_t) \tag{1}$$

In equation (1), f_t represents the gate value. w_f represents the weight matrix. h_{t-1} is the neuron output from the previous moment. x_t means the input at the current moment. b_t is the bias amount. The formula for updating the information of the input gate is equation (2):

$$i_t = \sigma(w_i \cdot [h_{t-1}, x_t]) \tag{2}$$

In equation (2), i_t represents the information that needs to be updated. w_i represents a parameter. The calculation formula for obtaining candidate vectors from the input gate is equation (3):

$$\tilde{C}_t = \tanh(w_c \cdot [h_{t-1}, x_t] + b_c) \tag{3}$$

In equation (3), $\tilde{C}_t$ represents the candidate vector. w_c is the weight vector. b_c is bias. The formula for updating cell status is equation (4):

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{4}$$

In equation (4), C_t and C_{t-1} mean the current and previous cell states. The cell state will determine the output of LSTM. First, the output of the cell state is calculated using the formula shown in equation (5):

$$o_t = \sigma(w_o[h_{t-1}, x_t] + b_o) \tag{5}$$

In equation (5), o_t represents the output of cell state. w_o and b_o are the weight matrix and bias of the output gate, respectively. The output formula of neurons is equation (6):

$$h_t = o_t * \tanh(C_t) \tag{6}$$

In equation (6), h_t represents the output of the neuron. In LSTM, the partial form of the function is changed to the cumulative form, which alleviates the gradient vanishing problem of RNN.

B. IMPROVED Pix2code APPLICATION CAG FRAMEWORK

Pix2code was developed by the UIzard formula in Copenhagen, which can directly generate code based on design drawings. The neural network in Pix2code is a typical CNN that can analyze images at different scales. However, due to the fact that the CGA of Pix2code is only 77%, in order to improve CGA, this study has made improvements to Pix2code. To improve Pix2code and enable the model to learn the impact of each line of code on design elements, this study divided the training process into three stages. The training diagram of Pix2code is Fig. 6.

In Fig. 6, Pix2code consists of a visual module, a language module, and a decoding layer. The visual module is composed of CNN, the language module consists of the first LSTM, and the second LSTM is made up with Softmax to understand the code layer. During the training process, CNN will first be used to learn GUI images, and then the DSL code will be understood through the first LSTM module, and then to use the second LSTM to understand the output of CNN and the first LSTM and describe the code content based on GUI images. The code generation process of Pix2code is Fig. 7.

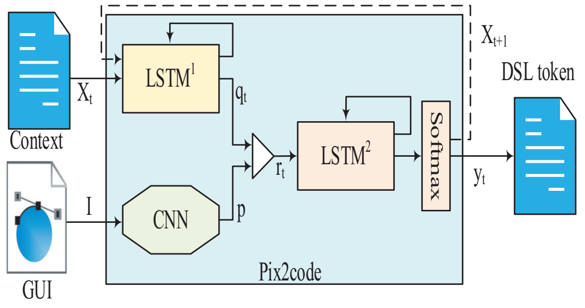

Fig. 6. Training diagram of Pix2code.

In Fig. 7, during the code generation, the GUI image and empty DSL file are used as inputs, and Pix2code uses a visual module to extract the GUI image features, to use language modules to understand the basic rules of DSL files, that is, to understand the generation rules of the next word, and then to understand the correspondence between DSL files and GUI image elements through the decoding layer. In the visual module of Pix2code, CNN can utilize its characteristics of local connections and weight sharing to process GUI images. The traditional Pix2code uses VGGNet for feature extraction. However, due to the large FCL parameters of VGGNet, the model requires lots of memory and is prone to overfitting. Therefore, to compress the size of the model and avoid overfitting issues, this study uses ResNet to replace the original VGGNet. ResNet, as a deep CNN, is easier to optimize compared to VGGNet. The inside residual blocks skip correlations to alleviate the gradient vanishing issue caused by growing depth. In the language module, the original Pix2code used LSTM to understand the internal connections between codes. Compared to traditional RNN, LSTM uses forget gates to select output information, effectively alleviating the problems of gradient vanishing and exploding. When the forget door opens, the gradient approaches 1, and there is no problem of gradient disappearance. And since the sigmoid is always less than 1, there will be no gradient explosion problem [24–26]. However, LSTM finds it difficult to predict the current word based on sequence information when understanding code patterns. Therefore, to improve the accuracy of single prediction, this study replaced LSTM with BLSTM. Compared to LSTM, BLSTM can achieve bidirectional transmission, even if the sentence order is reversed and the keywords come after it, it will not affect BLSTM. In the decoding

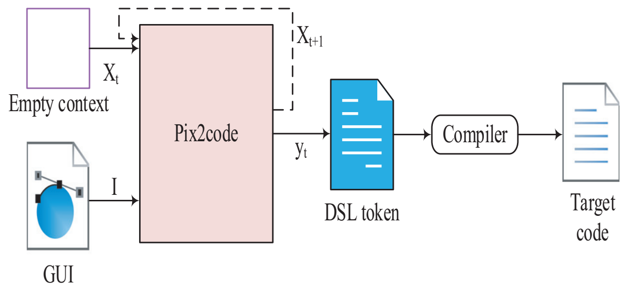

Fig. 7. The code generation process.

layer, the traditional Pix2code model uses a 2-layer LSTM with 512 neurons per layer as the structural foundation. However, due to the fact that the decoding layer not only needs to learn the type of code but also needs to learn the layout of the code and generate predictive code, using the feature extraction results of the visual module and combining with LSTM can achieve the processing of sequence problems. However, due to the heavy workload of the decoding layer, relying solely on LSTM makes it difficult to understand and correspond the relationship between image elements and code. Therefore, to improve the accuracy of the decoding layer, this study also uses BLSTM to replace the original LSTM.

IV. EVALUATION RESULTS AND ANALYSIS

The current Pix2code model consists of simple classical algorithms. To enhance the CGA, this study utilizes ResNet and BLSTM to improve it. To test the enhanced Pix2code performance, this study tests the visual module, language module, and decoding layer separately and compares the improved Pix2code with the original Pix2code. Table I lists the experimental settings.

In Table I, the processor used in the experiment is Intel Core I7-6700, the graphics card is GTX 1080Ti 11 GB, and the memory displacement is 16 GB. In the early stage of the experiment, the system used was Linux Deepin 15.9, and subsequent experiments are conducted on Windows 10. The development language for the experiment is Python 3.5.3, and the deep learning framework is TensorFlow GPU 1.4.0 and Keras 2.1.2. Three publicly available datasets, namely, the web-based UI (HTML/CSS), Android UI (XML), and iOS UI (Storyboard), are used in the experiment. The training and test set sizes for the Web-based UI (HTML/CSS) dataset are 143850 and 24108, 85756 and 14265 for the Android UI (XML) dataset, and 93672 and 15984 for the iOS UI (Storyboard) dataset. The learning rate of BLSTM ranges from 0.002 to 0.0025. The loss curves of the Pix2code model on different datasets are shown in Fig. 8.

In Fig. 8, in Web-based UI (HTML/CSS), the loss value of the Pix2code model gradually converges after 2 epochs, with a minimum loss value of around 0.1. In Android UI (XML), the loss value of the Pix2code model gradually converges after 3 epochs, with a minimum loss value of around 0.2. In the iOS UI (Storyboard), the loss value of the model also gradually stabilizes after 3 epochs, with a minimum loss value of around 0.3. The accuracy and loss curves of the visual module before and after improvement are shown in Fig. 9.

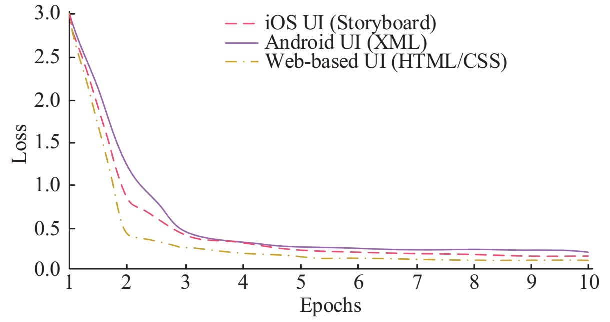

Fig. 8. Loss curves of the Pix2code model on different datasets.

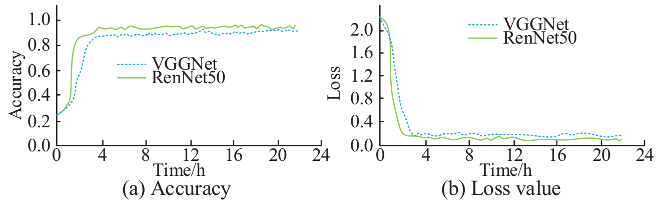

Fig. 9. Accuracy and loss curve before and after the improvement of the visual module.

In Fig. 9(a), before improvement, the accuracy of the visual module based on VGGNet stabilizes after 3 hours, with the highest accuracy of around 0.92. The accuracy of the visual module based on ResNet tends to stabilize after 2 hours, with the highest accuracy of about 0.96. In Fig. 9(b), the loss values of the visual module based on VGGNet and ResNet start to converge at 3 h and 2 h, respectively, with the minimum loss values being 0.15 and 0.08. The above results indicate that the accuracy of the visual module based on ResNet has been improved, and the generalization capacity has been enhanced. The test results before and after the improvement of the language module and decoding layer are shown in Fig. 10.

In Fig. 10(a), the accuracy of the training set (TSA) for the LSTM-based language module and decoding layer before improvement gradually stabilizes after 2 hours, while the TSA for the improved BLSTM-based language module and decoding layer

Table I. Experimental environment and dataset setting

Experimental environment	Version	Function
Processor	Intel Core I7-6700	The main frequency and maximum turbo frequency core frequency are 3.4 GHz and 4.0 GHz, respectively, and the core number is 4 cores, 8 MB level 3 cache.
Graphics card	GTX 1080Ti 11 GB	The core frequency is 1480–1584 MHz, the memory type is GDDR5X, the memory frequency is 11 Gbps, the memory bandwidth is 484 GB/s, and the memory width is 352 bit.
Internal storage	/	16 GB
System	Linux Deepin 15.9	/
	Windows 10	/
Dataset	Training set size	Test set size
Web-based UI (HTML/CSS)	142850	24108
Android UI (XML)	85756	14265
iOS UI (Storyboard)	93672	15984

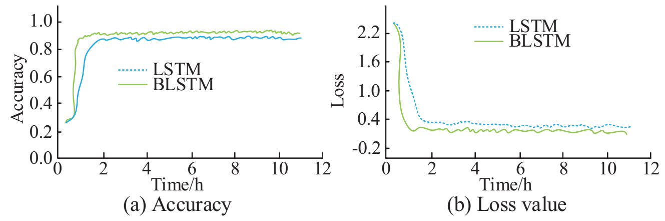

Fig. 10. Test results of the language module and the decoding layer.

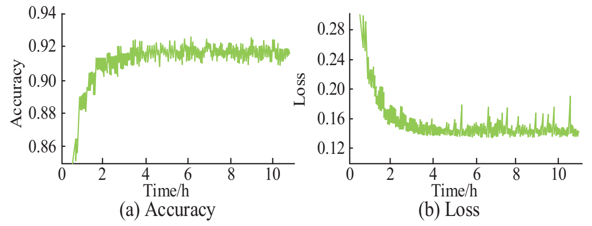

Fig. 11. Optimized BLSTM test results.

gradually stabilizes after 1 hour. The highest accuracy of BLSTM is 0.92, while the accuracy of LSTM is only 0.88. In Fig. 10(b), the loss values of LSTM and BLSTM begin to converge after 2 h and 1.5 h, respectively. Among them, there is a significant pause in the loss curve of BLSTM in the early stage of training, which is caused by unreasonable initialization parameter settings. The minimum loss values for LSTM and BLSTM are around 0.33 and 0.18, respectively. The above results indicate that BLSTM can perfect the CAG accuracy of the Pix2code. Although BLSTM can improve the accuracy and robustness of the Pix2code model, there is a significant difference in accuracy between its training and testing sets, resulting in overfitting issues. Therefore, to alleviate overfitting issues, this study set a 25% Dropout between the two BLSTM layers, and to accelerate convergence speed, Xavier normal distribution is used for initialization. The optimized BLSTM test results are shown in Fig. 11.

In Fig. 11(a), the TSA curve of the optimized BLSTM is steeper compared to before optimization, and the accuracy remains around 0.92. The test set accuracy has obviously improved to 85%. From Fig. 11(b), the stability of the optimized BLSTM loss value decreases, but its minimum loss value is only 0.14. This indicates that Dropout and Xavier normal distributions can effectively improve the overfitting problem of BLSTM, although it can also lead to a decrease in the stability of the model, its degree of improvement is higher. The experimental results of improved Pix2code and Pix2code are shown in Fig. 12.

In Fig. 12(a), the TSA of the Pix2code model is highest at about 0.89, while the improved Pix2code model is highest at around 0.95. In the test set, the highest accuracy of Pix2code is 77%, while the highest accuracy of improved Pix2code is 0.82. Although the accuracy has decreased compared to the Pix2code model improved solely using BLSTM, the stability of the model has significantly improved. In Fig. 12(b), the loss values of the Pix2code model before and after improvement all begin to converge after 3 hours, but the minimum loss value of the improved model is 0.09, which is smaller than before improvement. This indicates that the Pix2code model improved based on ResNet and BLSTM can effectively improve the accuracy of CAG and reduce the loss value. Compared to the Pix2code model improved solely by BLSTM, its stability has significantly improved.

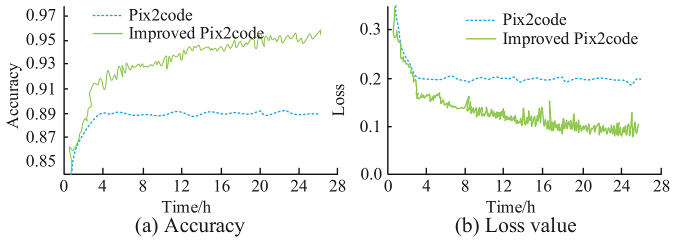

Fig. 12. Results of the modified Pix2code and Pix2code experiments.

V. CONCLUSION

This study proposed an improved Pix2code model based on ResNet and BLSTM. The experiment results showed that taking ResNet to improve the visual module of the Pix2code could greatly enhance the accuracy and convergence speed and reduce its loss value. The TSA improved by the visual module increased from 0.92 to 0.96, and the convergence time was shortened from 3 hours to 2 hours. After improving the language module and decoding layer using BLTM, the accuracy and convergence speed of the model have also been improved. The TSA lifted from 0.88 to 0.92, and the convergence time was shortened by 0.5 hours. However, the model improved by BLSTM might exhibit overfitting, so this study used Dropout and Xavier normal distribution to improve BLSTM. The data showed that the accuracy curve of the optimized BLSTM training set was steeper compared to before optimization, and the accuracy was still maintained at around 0.92. The testing set accuracy has improved, achieving 85%. Dropout and Xavier normal distribution could effectively improve the overfitting problem of BLSTM. Although it could also lead to a decrease in model stability, its improvement degree was higher. The Pix2code's TSAs improved by ResNet and BLSTM were 0.95 and 0.82, respectively, while the CGA of the original Pix2code was only 0.77. The above data implied that the CAG accuracy and stability of the Pix2code improved by ResNet and BLSTM have been improved. Due to the emerging research field of CAG, data acquisition is limited, resulting in a smaller dataset size and a simple hierarchical structure of GUI containers. Due to the improvement, the reliability of the performance testing of the good Pix2code model is poor. The focus of future work will be on how to expand the scale of datasets and improve the complexity of GUI structures to obtain better CAG models.

ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (No. 62062063), the Science and Technology Research Project of Jiangxi Provincial Department of Education, China (No. GJJ202310), and the Jiangxi Provincial Natural Science Foundation, China (No. 20224BAB202022).

CONFLICT OF INTEREST STATEMENT

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

REFERENCES

[1] W. Wang et al., “Reinforcement-learning-guided source code summarization using hierarchical attention,” *IEEE Trans. Softw. Eng.,* vol. 48, no. 1, pp. 102–119, 2020.

[2] M. Li et al., “The deep learning compiler: a comprehensive survey,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 708–727, 2020.

[3] O. Topsakal and T. C. Akinci, “Classification and regression using automatic machine learning (AutoML)–open source code for quick adaptation and comparison,” *Balkan J. Electr. Comput. Engineering*, vol. 11, no. 3, pp. 257–261, 2023.

[4] X. Wang, M. Magno, L. Cavigelli, and L. Benini, “FANN-on-MCU: an open-source toolkit for energy-efficient neural network inference at the edge of the Internet of Things,” *IEEE Internet Things J.*, vol. 7, no. 5, pp. 4403–4417, 2020.

[5] C. Wang, G. Zhu, T. Liu, Y. Xie, and D. Zhang, “A subdomain adaptive transfer learning base on residual network for bearing fault diagnosis,” *J. Vib. Control*, vol. 29, no. 1, pp. 105–117, 2023.

[6] M. S. Deelip and K. Govinda, “ExpSFROA-based DRN: exponential sunflower rider optimization algorithm-driven deep residual network for the intrusion detection in IOT-based plant disease monitoring,” *Int. J. Semantic Comput.*, vol. 17, no. 1, pp. 5–31, 2023.

[7] L. Chengdong, Y. Yulong, S. Linyuan, Z. Hanyuan, and J. Yongqing, “Fault diagnosis of air handling unit via combining probabilistic slow feature analysis and attention residual network,” *Neural Comput. Appl.*, vol. 35, no. 30, pp. 22449–22467, 2023.

[8] M. Wei, “A novel face recognition in uncontrolled environment based on block 2D-CS-LBP features and deep residual network,” *Int. J. Intell. Comput. Cybern.*, vol. 13, no. 2, pp. 207–221, 2020.

[9] G. Bansal, V. Chamola, P. Narang, S. Kumar, and S. Raman, “Deep3DSCan: deep residual network and morphological descriptor based framework for lung cancer classification and 3D segmentation,” *IET Image Proc.*, vol. 14, no. 7, pp. 1240–1247, 2020.

[10] Z. Wang and Y. Wang, “Bridge weigh-in-motion through bidirectional Recurrent Neural Network with long short-term memory and attention mechanism,” *Smart Struct. Syst.*, vol. 27, no. 2, pp. 241–256, 2021.

[11] W. Du, G. Jiang, W. Xu, and J. Ma, “Sequential patent trading recommendation using knowledge-aware attentional bidirectional long short-term memory network (KBiLSTM),” *J. Inf. Sci.*, vol. 49, no. 3, pp. 814–830, 2023.

[12] M. Jiao and D. Wang, “The Savitzky-Golay filter based bidirectional long short-term memory network for SOC estimation,” *Int. J. Energy Res.*, vol. 45, no. 13, pp. 19467–19480, 2021.

[13] Z. Hu, C. Fan, Q. Zheng, W. Wu, and B. Liu, “Asyncflow: a visual programming tool for game artificial intelligence,” *Vis. Inf.*, vol. 5, no. 4, pp. 20–25, 2021.

[14] S. Ren, L. Pottier, M. Buffa, and Y. Yu, “JSPatcher, a visual programming environment for building high-performance web audio applications,” *J. Audio Eng. Soc.*, vol. 70, no. 11, pp. 938–950, 2022.

[15] W. Ying, T. Dong, and C. Shentu, “Accurate stereo image super-resolution using spatial-attention-enhance residual network,” *Multimedia Tools Appl.*, vol. 82, no. 8, pp. 12117–12133, 2023.

[16] V. Gupta and V. Bibhu, “Deep residual network based brain tumor segmentation and detection with MRI using improved invasive bat algorithm,” *Multimedia Tools Appl.*, vol. 82, no. 8, pp. 12445–12467, 2023.

[17] K. X. Sang, J. Shang, and T. R. Lin, “Synchroextracting transform and deep residual network for varying speed bearing fault diagnostic,” *J. Vib. Eng. Technol.*, vol. 11, no. 1, pp. 343–353, 2023.

[18] X. Xue, C. Jiang, J. Zhang, and C. Hu, “Biomedical ontology matching through attention-based bidirectional long short-term memory network,” *J Database Manag*, vol. 32, no. 4, pp. 14–27, 2021.

[19] G. Du, Z. Wang, B. Gao, S. Mumtaz, and C. Du, “A convolution bidirectional long short-term memory neural network for driver emotion recognition,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 7, pp. 4570–4578, 2020.

[20] X. Chen, J. He, X. Wu, W. Yan, and W. Wei, “Sleep staging by bidirectional long short-term memory convolution neural network,” *Future Gener. Comput. Syst.*, vol. 109, no. 6, pp. 188–196, 2020.

[21] Z. Wang and Y. Wang, “Bridge weigh-in-motion through bidirectional Recurrent Neural Network with long short-term memory and attention mechanism,” *Smart Struct. Syst.*, vol. 27, no. 2, pp. 241–256, 2021.

[22] K. Puh and M. B. Babac, “Predicting sentiment and rating of tourist reviews using machine learning,” *J. Hospitality Tourism Insights*, vol. 6, no. 3, pp. 1188–1204, 2023.

[23] X. Zhou, F. Teng, and Y. Zhang, “Automatic international classification of diseases coding model based on meta-network,” *J. Comput. Appl.*, vol. 43, no. 9, pp. 2721–2726, 2023.

[24] N. Andersen, F. Zehner, and F. Goldhammer, “Semi-automatic coding of open-ended text responses in large-scale assessments,” *J. Comput. Assist. Learn.*, vol. 39, no. 3, pp. 841–854, 2023.

[25] S. Qi et al., “Dynamically relative position encoding-based transformer for automatic code edit,” *IEEE Trans. Reliab.*, vol. 72, no. 3, pp. 1147–1160, 2023.

[26] Y. Jia, L. Qu, and X. Li, “Automatic path planning of unmanned combat aerial vehicle based on double-layer coding method with enhanced grey wolf optimizer,” *Artif. Intell. Rev.*, vol. 56, no. 10, pp. 12257–12314, 2023.