

Received November 3, 2021, accepted November 22, 2021, date of publication November 30, 2021, date of current version December 10, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3131753

GUI Widget Detection and Intent Generation via Image Understanding

PENGHUA ZHU¹, YING LI¹, TONGYU LI², WEI YANG³, AND YIHAN XU⁴

¹North China Institute of Aerospace Engineering, Langfang 065000, China

²State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China

³Beijing Aerospace Automatic Control Institute, Beijing 100854, China

⁴Jiangsu Vocational College of Electronics and Information, Huai'an 223003, China

Corresponding author: Wei Yang (18618499929@163.com)

This work was supported in part by the Special Fund for Military-Civilian Integration Development of Hebei Province under Grant JMYB-2020-01, and in part by the Huai'an City Innovation Service Capability Building Plan Project Fund-Huai'an City Software Testing Technology Key Laboratory under Grant HAP201904.

ABSTRACT Aerospace control software is the most important part of aerospace software. Since its potential defects endanger life and safety, there are strict requirements on product quality. Therefore, efficient and reliable software testing is essential. The traditional testing method has been challenging to meet its development requirements, and software automation testing has gradually become the main tool for testing aerospace control software. For the automation testing of aerospace control software, the core problem is to locate the GUI widgets on the software screenshots and identify their intent, which directly affects the accuracy of the test. Because of this, we use the widget recognition technology based on image matching and use the image understanding and analysis technology to extract the widget image in the screenshots. After obtaining the widget image, we use a convolutional neural network to extract image features and use the encoder module to encode the extracted information features as a tensor. The decoder module generates a word sequence conditional on tensor and previous output based on the encoded information. We also conduct an empirical study to evaluate the accuracy of widget recognition and intention generation. For widget recognition, our average IoU reached 0.81. For widget intent generation, our model BLEU-1 is 0.567, BLEU-2 is 0.356, BLEU-3 is 0.261, BLEU-4 is 0.131. The results show that our method is very effective.

INDEX TERMS GUI widget detection, GUI widget intent generation, aerospace control software.

I. INTRODUCTION

As a core part of aerospace software, aerospace control software has a unique working environment and strict requirements on reliability and functionality. Product quality has always been valued. Therefore, in the process of developing aerospace control software, efficient and reliable software testing is particularly important [1]. It is the final review of software requirements, design specifications and coding, and is the critical step in software quality assurance. With the increasing requirements for developing control software in aerospace missions, traditional testing methods can no longer meet the new development requirements. The main problem is in the complex structure of the software system, higher and higher reliability requirements, and the lack of a unified verification environment after the completion of the software

coding, resulting in a longer software testing cycle [2]. The urgency of aerospace control software development requires that the task be completed as soon as possible on the premise of ensuring reliability, which requires the use of existing resources to build a test environment as soon as possible to reduce repetitive work. The establishment of domestic self-control automatic test platform can realize the automation and intelligence of software tests, which can thoroughly verify the requirements of aerospace models for the functions, performance, reliability, and security of operating systems and applications. System-level functional testing is a critical task in the development of aerospace models and their control software. This task is faced with heavy test regression workload and low efficiency, and difficulties in test organization and management. Automated testing technology can provide an excellent convenience for aerospace control software system-level functional testing. For automation testing frameworks, detecting Graphical User Interface (GUI) elements

The associate editor coordinating the review of this manuscript and approving it for publication was Porfirio Tramontana.

in GUI images are the focus of research, and its efficiency directly affects the accuracy of automation testing. Common GUI element detection methods are mainly divided into three types: coordinate-based, source code-based, and component tree traversal-based element detection methods [3]. These identification methods have shortcomings, such as insufficient support for cross-device testing, inapplicable black box testing scenarios, or low recognition accuracy. We adopt GUI element detection technology based on image matching to meet software characteristics under different operating systems and development tools, such as various types and significant differences in interface styles. We use image understanding and analysis technology to extract widgets in the screenshot. In addition to detecting GUI elements, we also need to improve the adaptability and accuracy of our image recognition algorithms.

After identifying the widgets in the screenshot, clarifying the intent of these widgets is also a crucial link for automated testing. The intention to generate the widget image is to summarize the image into text and describe the target information in the widget image and the relationship between the targets through a paragraph of text to assist the computer in understanding and facilitating automated testing. We use a convolution neural network (CNN) to extract image features and then use the encoder module to encode this extracted information feature into a tensor. The decoder module generates an output (word sequence) conditioned on the tensor and the previous output based on the encoded information.

We also designed an empirical study to evaluate the accuracy of widget recognition and intent generation. The results show that our method is universal and effective.

In summary, we declare the following contributions:

- We propose a new method to improve the automated testing level of general software and aerospace control software through deep image understanding. We extract all the widgets from the screenshots and generate natural language descriptions that match their meanings.
- We conduct an empirical evaluation of the proposed method, and the results show that our method can identify more precise widgets and generate their intents more accurately.

The rest of this paper is organized as follows. In the section II, we introduce the relevant background. Section III describes the framework and technology of the experiment in detail. In section IV, we conduct empirical experiments to evaluate the effectiveness of the proposed results. Section V introduces some related work of this paper. Finally, we draw a conclusion in section VI.

II. BACKGROUND

A. AEROSPACE CONTROL SOFTWARE

Due to the particularity of its products, the aerospace industry often adheres to the principle of excellence and does not allow product defects to endanger life and safety [4], [5]. The aerospace control software system is characterized by many

parallel tasks, complex software logic and interfaces, and harsh external and electromagnetic environments for software operation. Due to its special working environment, the aerospace control software puts forward higher requirements for its product quality.

In the area of aerospace control software testing, as a key technology to ensure software quality and reliability, aerospace software testing is receiving more and more attention [6], [7]. However, as the scale of software becomes larger and larger, the workload of testing is also increasing. This urgently requires a more efficient, general, and more automatic and intelligent test platform to control product quality. Automated testing technology can provide an excellent convenience for aerospace control software testing. In automated testing, detecting graphical user interface (GUI) elements and their intentions is the focus of research, and its efficiency directly affects the accuracy of automated testing. However, the existing automated test technology has a single object recognition method.

B. GUI ELEMENT DETECTION

The traditional automated testing technology mainly obtains the test source GUI object through the API function of the automated testing framework. The related user events are triggered by the testing framework's API based on the test source object. However, this method is difficult to meet the diverse requirements of the aerospace control software operating system and development environment.

With the development of pattern recognition, digital image processing, and the improvement of computer data processing capabilities, GUI object detection based on image understanding has become more and more popular [8], [9]. GUI elements can be roughly divided into text elements and non-text elements. For text elements, most studies use Optical Character Recognition (OCR) [10] technology to identify them. OCR technology can better identify text elements in screenshots. As for the recognition of text elements, there are mainly two methods of edge/contour aggregation and image matching. Edge/contour aggregation detects raw edges and regions and merges them into a larger area, of which Canny edge detection [11] is the most commonly used. Canny edge detection uses a multi-stage algorithm to detect a wide range of edges in the image. Its goal is to capture as many edges as possible and locate the center of the edge. It can be used to detect the edges of components in a given GUI page. After Canny edge detection, Edge dilation [12] is usually used to fuse the adjacent elements further. But this method is not sensitive to artificial GUI elements. The image matching method matches samples/prototypes to detect object bounding boxes and classes simultaneously, but it is only suitable for standard and straightforward GUI elements.

C. IMAGE CAPTION

Image caption generation is to summarize the image into text and describe the target information in the image and the relationship between the targets through a paragraph of text [13].

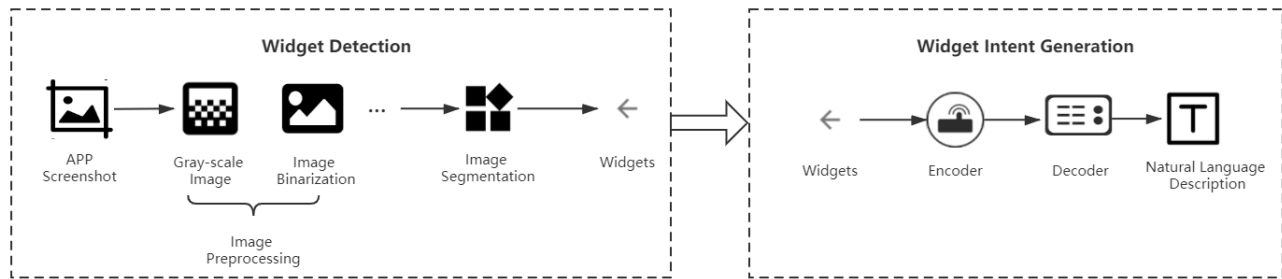


FIGURE 1. Overall flow of the system.

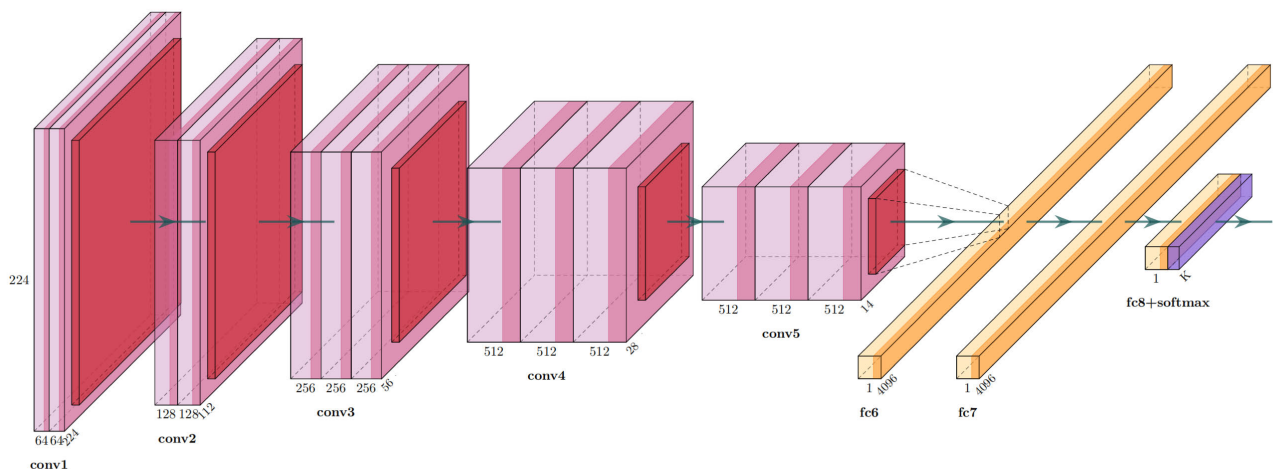


FIGURE 2. VGG16 structure.

There are many classification methods for image captions. The generation of image captions is based on the original sentence retrieval method, to the method based on template matching and replacement to increase sentence diversity, to the deep learning method.

The deep learning method consists of a two-stage non-end-to-end method. After CNN classification or target detection generates labels, the CNN results are input to RNN to generate sentences. The end-to-end method is developed to directly use the feature maps generated by CNN in RNN to achieve end-to-end training. The end-to-end image caption generation method is continuously improved from the original basic framework of CNN and RNN, introducing attention mechanism, replacing RNN with transformer [14], or using Generative Adversarial Network (GAN) [15] to generate sentences and discriminate. In addition, there are methods such as further optimizing image captions by optimizing evaluation indicators through reinforcement learning or generating multi-emotional captions. With the continuous development of the image object detection model and natural language processing machine translation model [16], [17], which can improve the CNN of image subtitle according to the optimization of the object detection model. The CNN structure with higher precision is used, while the subtitle generation model can be improved according to the machine

translation model to achieve high parallel subtitle generation. In addition, optimizing the reinforcement learning mechanism and antagonistic mechanism can further improve the model's performance.

III. APPROACH

To recognize the GUI widgets in the screenshot of aerospace control software, we use the widget region detection based on image understanding analysis to preprocess and segment the image. After obtaining the widget's image, we use the "encoder-decoder" model to obtain the intention of the widget. The overall flow of the system is shown in Figure 1.

A. WIDGET DETECTION

Due to the unknown equipment of the aerospace control software, and the resolution of screenshots captured by different equipment is different, so higher requirements are put forward for identifying the widgets in the screenshot. Using widget area detection based on image understanding analysis can avoid the impact of resolution on recognition. Therefore, in terms of image processing, based on the characteristics of the screenshot itself, we performed a series of processing on the screenshot.

The image understanding analysis of screenshots is divided into two parts, image preprocessing and image segmentation.

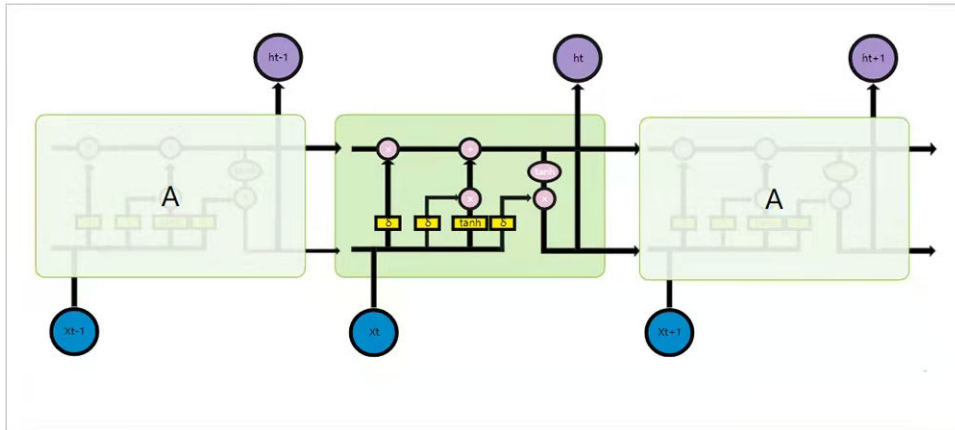


FIGURE 3. LSTM structure.

In the process of image preprocessing, we perform grayscale, gradient, image binarization, and noise removal operations on the screenshot. The primary purpose of image preprocessing is to eliminate irrelevant information in the image, restore real useful information, enhance the detectability of related information and simplify data to the greatest extent, thereby improving the reliability of image segmentation, matching, and recognition.

After preprocessing the image, we segment the image. The segmented image is convenient for detecting widgets. We use the Flood Fill Algorithm to mark the position of the layout block in the image. The Flood Fill algorithm is a filling algorithm commonly used in many graphics drawing software. Its principle is to start from a point and fill nearby pixels with a new color until all enclosed areas are filled with the new color. In the image, the layout blocks may be nested or overlapped. Therefore, after the layout blocks are detected, the level of the blocks needs to be calculated. We calculate the hierarchical relationship by comparing the relative sizes of the two blocks. Subsequently, we detect whether there are widgets in the block. We check whether it is a widget itself based on the relative size of the block, and if it meets our standards, we consider it to be. Then, the binarized image is processed, and the block is cut according to the relative size of the block. Therefore the area containing the widget in the block is obtained. Finally, extract the widget in the area containing widgets. We take the binary image as input, calculate the connected areas, get their boundaries, and extract the widget.

B. WIDGET CLASSIFICATION

After identifying the GUI widgets in the screenshots of the aerospace control software, generating a natural language description that matches their meaning is the next step, which is an image caption problem. Image captioning is a challenging artificial intelligence problem that involves generating textual descriptions for a given widget image. Presenting UI widgets to their corresponding content descriptions is a typical task of image captioning.

1) FEATURE EXTRACTION

a: WIDGET IMAGE FEATURE

Convolutional neural network (CNN) is a kind of feed-forward neural network. Its artificial neurons can respond to some surrounding cells in the coverage area, and it has excellent performance for large-scale image processing. We use CNN to extract image features. CNN usually contains two types of layers, convolutional layer, and pooling layer.

b: CONVOLUTIONAL LAYER

In a convolution neural network, each convolution layer is composed of several convolution units, and a backpropagation algorithm optimizes the parameters of each convolution unit. The purpose of convolution operation is to extract different features of the input. The first convolution layer can only extract some low-level features, such as edge, line, and angle. More layers of the network can extract more complex features iteratively from low-level features. Usually, convolution helps us find specific local image features (such as edges) for later networks.

c: POOLING LAYER

Pooling is another important concept in convolutional neural networks, which is a form of downsampling. There are many different forms of nonlinear pooling functions, among which “Max pooling” is the most common. It divides the input image into several rectangular regions and outputs the maximum value for each sub-region. Intuitively, this mechanism can work effectively because its precise position is far less important than its relative position with other features after a feature is found. The pooling layer will continuously reduce the spatial size of the data, so the number of parameters and the amount of calculation will also decrease, which also controls the overfitting to a certain extent.

Due to VGG16’s excellent performance, we use the pre-trained VGG16 [18] to analyze the image content, calculate the image feature, and save it to a file. VGGNet-16 can be divided into 8 segments as a whole, the first 5 segments are convolutional networks, and the last 3 segments are fully

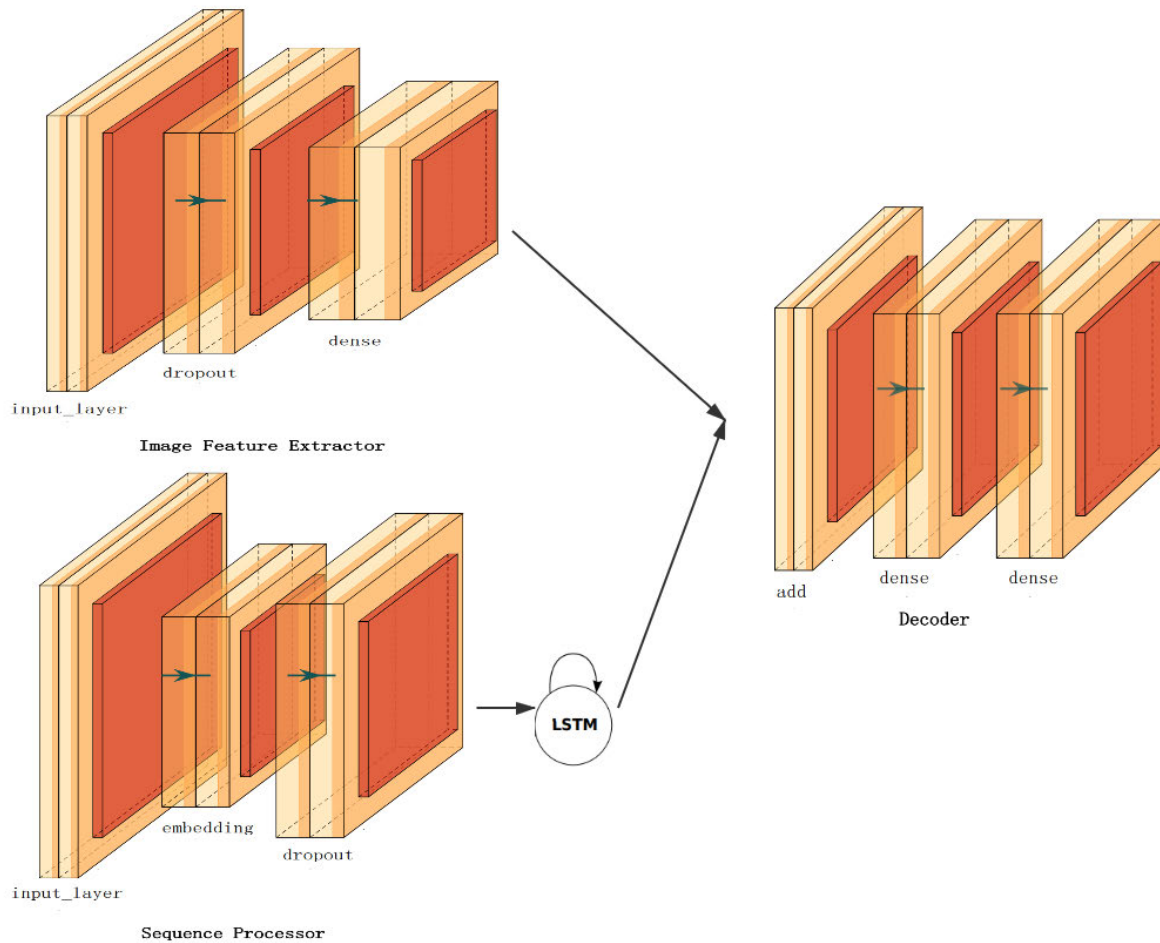


FIGURE 4. Network architecture.

connected networks. First, create the first convolutional network. This convolutional network consists of 2 convolutional layers and 1 maximum pooling layer, a total of 3 layers. For the two convolutional layers, the size of the convolution kernel is 3×3 , and the number of convolution kernels (the number of output channels) is also 64, the step size is 1×1 , and the padding is both 1. These are the parameters that we care about when doing convolution: the size of the convolution kernel, the depth of the convolution kernel, the step size, and the padding. Therefore, the input size of the first convolutional layer is $224 \times 224 \times 3$ and the output size is $224 \times 224 \times 64$; the input size of the second convolutional layer is $224 \times 224 \times 64$, and the output size is $224 \times 224 \times 64$; after the two convolutions, there is a 2×2 maximum pooling layer. The step size is 2, so after the maximum pooling layer, the output result size becomes $112 \times 112 \times 64$. The network structure is shown in Figure 2.

We will remove the last layer of the loaded model because this layer is used to predict the classification of the image. We are not interested in image classification; we are interested in the internal representation of the image before

classification. These are the features that the model extracts from the image.

Widget image has a corresponding description, and we need to clean these textual descriptions to a minimum. First, load the file containing all the textual descriptions. Each image has a unique identifier, which appears in the file name and textual description file. Next, we will deal with the image description step by step to obtain the image identifier dictionary. Each image identifier is mapped to one or more textual descriptions. Next, we need to clean up the description text. Because the description has been symbolized, it is straightforward to handle. We will clean the text in the following ways to reduce the amount of vocabulary that needs to be processed:

- All words are converted to lowercase.
- Remove all punctuation.
- Remove all words less than or equal to one character (such as 'a').
- Remove all words with numbers.

After obtaining the image identifier dictionary of the description, we traverse each description and clean the text.

After cleaning, we can summarize the vocabulary. Ideally, we want to use as few words as possible to get strong expressiveness. The smaller the vocabulary, the smaller the model and the faster the training speed. We convert the clean description into a set and print out its scale to understand the size of our data set vocabulary. Finally, we save a dictionary of image identifiers and descriptions, with only one image and one description per line.

2) DATA LOADING

First, we must load the prepared image and text data to fit the model. We will train data on all images and descriptions in the training data set. During the training process, we monitor the model's performance on the data set and use the performance to determine when to save the model. From the image's filename, we can extract the image identifiers and use them to filter the image and description for each set of identifiers. Then, we load the image and description using the pre-defined trained set of identifiers.

Our model can generate a caption for a given image, one word at a time, and take the previously generated word sequence as input. Therefore, we need a first word to start the generation step and the last word to indicate the end of the subtitle generation. We use the strings "startseq" and "endseq" for this purpose. These tags are added to the load description as if they were loaded out by themselves. It is very important to do this before encoding the text so that these tags can be correctly encoded.

The textual description needs to be encoded into numerical value before being fed to the model as input or compared with model prediction. The first step in encoding numerical values is to create a continuous mapping between words and unique integer values. Keras provides the Tokenizer class, which can learn the mapping based on the loaded description data. We now encode the text so that each description will be split into words. We provide one word and image to the model, and then the model generates the next word. The first two words and images described will be used as input of the model to generate the next word, which is how the model is trained. Later, when the model is used to generate a description, the generated words will be concatenated and used as input recursively to generate an image caption.

3) MODEL IMPLEMENTATION

We define the deep learning model based on the "merge-model" described in the [19], [20] by Marc Tanti, *et al.* We describe the model from three parts, and the structure of the model is shown in Figure 4.

a: IMAGE FEATURE EXTRACTOR

This is a pre-trained 16-layer VGG model. We use the VGG model (no output layer) to preprocess the image and use the extracted features predicted by the model as input. The input image features of the image feature extractor model are 4096-dimensional vectors, which are processed by the fully

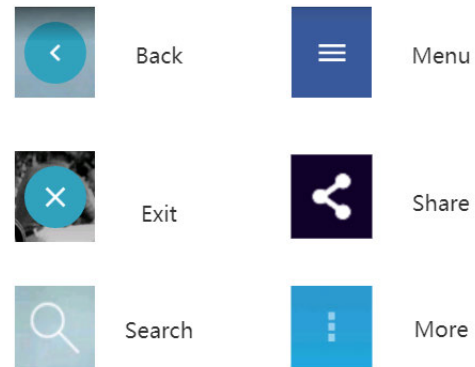


FIGURE 5. Examples of widgets in dataset.

connected layer to generate 256-element representations of the image.

b: SEQUENCE PROCESSOR

A word embedding layer for processing text input, followed by a long and short-term memory (LSTM) [21] recurrent neural network layer. Long short term memory is a special RNN, which is mainly used to solve the problem of gradient disappearance and gradient explosion in long sequence training. In short, it is better than ordinary RNN in longer sequences. There are four states inside a single loop structure (also called a cell) of LSTM. Compared with RNN, the LSTM loop structure maintains a persistent unit state and is continuously transmitted, which is used to determine which information to forget or continue to be transmitted. The LSTM with three continuous loop structures is shown in the Figure 3. Each loop structure has two outputs, one of which is the unit state. LSTM is a kind of threshold RNN. The ingenuity of LSTM is that by increasing the input threshold, forgetting threshold, and output threshold, the weight of the self-loop is changed. In this way, when the model parameters are fixed, the integration scale at different times can be dynamically changed, thereby avoiding gradients. The problem of disappearing or gradient expansion.

The sequence processor model expects that the predefined length input sequence fed to the embedding layer uses a mask to ignore the padded value. After that is the LSTM layer with 256 cyclic units.

c: DECODER

The feature extractor and sequence processor output a fixed-length vector. These vectors are fused and processed by the Dense layer to make the final prediction.

Both the image feature extractor and the sequence processor output 256-element vectors. In addition, the input model uses regularization at a dropout rate of 50% to reduce overfitting of the training data set because the model configuration learning is very fast. The decoder model uses additional operations to fuse the vectors from the two input models. It is then fed to a dense layer of 256 neurons and then sent to the

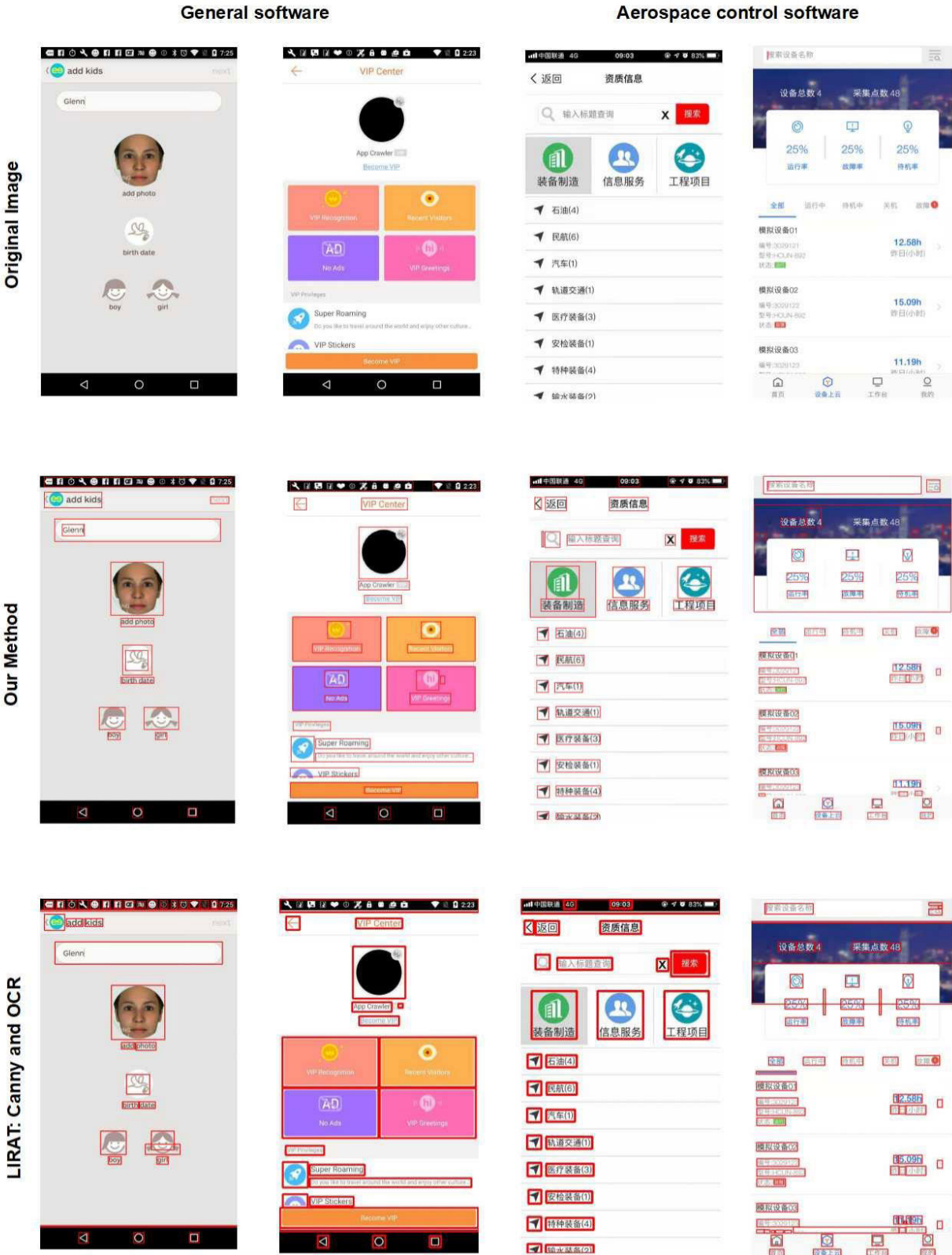







FIGURE 6. Widget detection result.

TABLE 1. Examples of widget intent generalization.

	E1	E2	E3	E4	E5
Image					
Ground-truth	share	like	menu	refresh	search
LabelDroid	share	like tweet	more options	refresh	search
Generation	share	add to favorites	open menu	refresh page	search

final output dense layer to perform softmax prediction on the next word in the sequence on all output words.

IV. EVALUATION

We define 2 research questions (RQ), respectively evaluate the widget detection effectiveness, and the widget intent generation effectiveness.

- How effective is our method in the widget recognition?
- How effective is our method in the widget intent generation?

A. WIDGET RECOGNITION EFFECTIVENESS

For automated testing of aerospace control software, identifying the widgets in the screenshot is crucial, so we first detect the widgets included in the screenshot. By performing grayscale, gradient, image binarization, denoising, and segmentation operations on the image, we can get the bounding boxes of the widgets and mark them on the original image.

We conducted experiments on screenshots of multiple mobile applications, including aerospace control software and general-purpose software. We compare our method with the method of detecting widgets in [22]. They use canny and OCR to identify widgets in mobile app screenshots. They extend widget edges, connect redundant outlines, retain large and meaningful widgets, text, etc., and then calculate the size of the extracted widgets. Small widgets smaller than 1% of the active size will be discarded. At the same time, OCR technology is introduced to help detection.

Figure 6 shows an example of our method and baseline detection results. We show the widget recognition effect of two sets of general software screenshots and two sets of aerospace control software screenshots. The result shows that our method is more accurate and finer than the baseline recognition of widgets in both general software and aerospace control software. We also use IoU (Intersection over Union) to evaluate the effectiveness of our recognition. IoU calculates the intersection and union ratio of the predicted bounding box and the real bounding box [23]. The average IoU of our method reached 0.81, and the baseline method reached 0.76. It can be seen that our method is very effective.

B. WIDGET INTENT GENERATION EFFECTIVENESS

After detecting the widgets in the screenshot, we use the “encoder-decoder” model to generate the widget intent. To evaluate our model, we constructed a dataset of 2,000 in size, which contains general images and domain-specific

images. We divide the dataset into training set (70%), validation set (20%), and test set (10%). We record the accuracy on the validation set and the final accuracy on the test set. And we also adjust the parameters and structure of the neural network until the test accuracy reaches the peak. Some examples in the data set are shown in Figure 5.

We compared our method with LabelDroid [24]. LabelDroid uses CNN to extract image features. To further encode the visual features extracted from CNN, it first embeds them using a fully connected layer. It then uses a Transformer model based on machine-translation tasks as the encoder. We give some examples of experimental results in the Table 1. We invite three students majoring in software engineering and a domain expert to generate reviews for the widget images. We generate ground truth by combining the results of four professionals. It can be seen that although some results predicted by our model are not the same as the given growing truth, they are all synonymous. Meanwhile, it can be seen that the results generated by our model are more concise and closer to ground truth than those generated by LabelDroid.

We also use the corpus BLEU [25] value to evaluate the experimental results. The BLEU value is used to evaluate the similarity between the translation and one or more reference translations in the text translation. Here, we compare each generated description with the ground truth description of the image and then calculate the BLEU value of n-gram language models such as 1,2,3, and 4. After calculation, our model BLEU-1 is 0.567, BLEU-2 is 0.356, BLEU-3 is 0.261, BLEU-4 is 0.131, while the baseline BLEU-1 is 0.558, BLEU-2 is 0.345, BLEU-3 is 0.252, BLEU-4 is 0.128. We can see that the BLEU value is within the optimal expectation range of the problem and is close to the optimal level. The result shows that our model is effective.

V. RELATED WORK

A. GUI WIDGET IDENTIFICATION

Most existing research on GUI element detection uses mature methods in the field of computer vision, including traditional image processing features, such as canny edges, contours, and deep learning models that learn detection from large-scale GUI data.

REMAUI can identify user interface elements such as images, texts, containers, and lists in a given mobile terminal screenshot and generate corresponding source code and resource files [26]. Kevin Moran *et al.* proposed Redraw, which realized automated GUI prototyping through three tasks: detection, classification, and assembly. Redraw uses

computer vision technology or model metadata to detect logical components from model artifacts. Then, it uses CNN to classify GUI components into domain-specific types accurately [27]. Jieshan Chen *et al.* make the first large-scale empirical study on seven representatives GUI element detection methods on more than 50K GUI images and design a new GUI specific non-text GUI element detection method [28]. Thomas D. White *et al.* automatically recognizes the GUI widgets in the screenshots by using machine learning technology [29]. They use YOLOv2 to recognize widgets, train, and test on artificial desktop GUIs.

Chen *et al.* uses machine vision technology to extract GUI elements from visual manuscripts and then uses deep learning technology to identify GUI element types. The DSL is generated by recurrent neural network technology, and finally, the grammar tree is matched with the template to generate the flutter code [30].

Pix2code [19] uses CNN to extract image features and represent images unsupervised and uses RNN to generate text representation of images. It implements an end-to-end model to convert GUI screenshots into computer code. In [9], a neural machine translation method is proposed, which translates the user interface design image into the skeleton of the graphical user interface. Visual features are extracted from UI images, and the spatial layout of these features is encoded.

We perform image comprehension analysis on screenshots. Firstly, the image is preprocessed, and the screenshots are mainly grayed out, gradiented, image binarized, and denoised. After preprocessing the image, we use the Flood Fill algorithm to segment the image and extract the widgets in the area containing the widgets.

B. WIDGET INTENT UNDERSTANDING

In recent years, there have been some researches on intention analysis of GUI widgets. Some works use deep learning model to generate natural language description for images [32], [33], and some studies also use crowdsourcing to generate image captions [34], [35] [36].

IconIntent identifies the use of sensitive UI widgets in Android software by combining program analysis and icon classification [37]. Xi *et al.* uses static program analysis technology to obtain GUI images and their triggered privacy usage. Then they model GUI images intention and privacy usage based on mutual attention mechanism and find GUI components with potential privacy leakage through anomaly detection [38]. [39] introduces a method based on code and vision to add the semantic annotation to mobile UI elements. According to the UI screen capture and view hierarchy, 25 UI component categories, 197 text button concepts, and 99 icon categories can be automatically identified.

AppIntent [40] analyzes the data input and event input that may lead to sensitive data transmission through improved event space constraint guided symbolic execution. Then AppIntent uses these inputs to control the application execution on the dynamic program analysis platform and helps analysts make judgments by presenting the operation

sequence of the graphical interface corresponding to sensitive data transmission.

Zhang *et al.* uses collected and annotated data sets of 77,637 screens (from 4,068 iPhone applications) to detect UI elements and introduces heuristics (e.g., user interface grouping and sorting). Other models (e.g., identifying user interface content, state interaction) to further improve user interface detection and add semantic information [41]. They use the existing icon recognition engine, and the image description function in IOS [42] are used to generate alternative text for the detected icon and image, respectively.

After identifying the widgets in the screenshot, we will generate natural language descriptions for these widgets. We use the “encoder-decoder” model to generate the widget intent. The encoder encodes image information and text information into a vector, and the decoder generates output based on this vector and the previously obtained results.

C. GUI IMAGE DATASET

To train our model, we need a data set containing a large number of UI widget images. More and more studies have opened up relevant data sets.

Rico [43] is a mobile application design repository, which supports five data-driven applications: design search, UI layout generation, UI code generation, UI code generation, user interaction modeling, and user perception prediction. It contains more than 66K unique UI interfaces and 3M UI elements with visual, text, structure, and interaction design attributes.

Chen *et al.* uses the application browser, through clicking, editing, and other operations, automatically browse the different screens in the application, obtain the screenshot of the application GUI, and automatically discard the runtime front-end code [24]. After deleting all copies by checking the screenshots, there are 278,234 screenshots containing ImageView and ImageButton from 10,408 apps.

MK Patrick *et al.* proposed a method for automatically creating tag training data, which includes a specific screen capture analyzed by a fully automatic dynamic program and a tag image corresponding to its category [27]. After filtering the dataset, they obtained 14,382 unique screens and 431,747 unique components from 6538 applications.

IconIntent [37] collected sensitive icons from two sources. First, they used the name of each sensitive category and the keyword “icon” to search for representative icons and downloaded the top 100 icons retrieved by each category. Second, they used keywords for each category to search for icons in other applications and got the top 500 icons for each category. Finally, they mark the icons manually.

The construction of our data set refers to the literature mentioned above. Our data set contains GUI images from aerospace control software and GUI images from general-purpose software.

VI. CONCLUSION AND FUTURE WORK

Due to its strict requirements for high reliability and functionality, aerospace control software puts forward higher

requirements for automated testing. In automated testing, it is crucial to identify the widgets in the screenshot and get their intent.

We adopt a widget area detection method based on image understanding and analysis. First, we perform image preprocessing and image segmentation operations on the screenshot. After preprocessing the image, we segment the image, mark the position of the layout block in the image, calculate the hierarchical relationship of the block, check whether the block contains widgets. After extracting the widgets in the screenshot, we generate their intent. We use the encoder-decoder model to get the widget intent. We use pre-trained VGG16 to extract image features and clean the corresponding textual description of each image to reduce the amount of vocabulary that needs to be processed. Then we use the encoder module to encode this extracted information feature into a tensor. Based on the encoded information, the decoder module generates one word and takes the previously generated word sequence as input. To evaluate the effectiveness of the proposed approach, we also designed an empirical study. The results show that our method is universal and effective.

In the future, we will first improve the model's performance and then apply the model to GUI images with corrupted text for adversarial testing. Second, we will apply the model to actual business software to evaluate the qualitative and quantitative benefits.

REFERENCES

- [1] S. Zhu et al., "Reliability measurement for aerospace software development," *Aerosp. Control*, vol. 22, no. 3, pp. 87–92, 2004.
- [2] L. Zhongping, Y. Hai, and X. Jing, "The application of LDRA TESTBED in aerospace software test," *Aerospace Control*, vol. 25, no. 2, pp. 73–77, 2007.
- [3] S. Ding, N. Gu, Z. Huang, and J. Hou, "APP control recognition algorithm based on text recognition and page layout," *Comput. Eng.*, vol. 45, no. 6, pp. 89–95, 2019.
- [4] Y. Hongjun and Z. Dongyan, "The research on reliability of aerospace software design and analysis," *Comput. Eng. Appl.*, vol. 36, no. 7, pp. 74–75, 2000.
- [5] J. Simei and Z. Liang, "Software engineering and its application in aerospace software system," *Tactical Missile Technol.*, vol. 2, pp. 80–84, 2006.
- [6] Z. Xinlei and L. Zhenggao, "Application and development trend of aerospace software reliability and safety technology," *Quality Rel.*, vol. 3, pp. 41–43, 2006.
- [7] X. Yinghui, "Research on aerospace software testing technology," in *Proc. 26th Nat. Symp. Space Explor. Space Explor. Committee Chin. Soc. Space Sci.*, 2013, pp. 150–154.
- [8] C. Chunyang, S. Feng, Z. Xing, L. Liu, S. Zhao, and J. Wang, "Gallery DC: Design search and knowledge discovery through auto-created GUI component gallery," in *Proc. ACM Hum.-Comput. Interact.*, vol. 3, 2019, pp. 1–22.
- [9] C. Chen, T. Su, G. Meng, Z. Xing, and Y. Liu, "From UI design image to GUI skeleton: A neural machine translator to bootstrap mobile GUI implementation," in *Proc. 40th Int. Conf. Softw. Eng.*, May 2018, pp. 665–676.
- [10] S. Mori, C. Y. Suen, and K. Yamamoto, "Historical review of OCR research and development," *Proc. IEEE*, vol. 80, no. 7, pp. 1029–1058, Jul. 1992, doi: 10.1109/5.156468.
- [11] J. Canny, "A computational approach to edge detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-8, no. 6, pp. 679–698, Nov. 1986.
- [12] (2018). *Dilatation Edge*. [Online]. Available: https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html
- [13] A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth, "Every picture tells a story: Generating sentences from images," in *Proc. Eur. Conf. Comput. Vis.* Berlin, Germany: Springer, 2010, pp. 15–29.
- [14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," 2017, *arXiv:1706.03762*.
- [15] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, *Generative Adversarial Nets*. Cambridge, MA, USA: MIT Press, 2014.
- [16] Y. Feng, L. Ma, W. Liu, and J. Luo, "Unsupervised image captioning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 4125–4134.
- [17] K. Shuster, S. Humeau, H. Hu, A. Bordes, and J. Weston, "Engaging image captioning via personality," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 12516–12526.
- [18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, *arXiv:1409.1556*.
- [19] M. Tanti, A. Gatt, and K. P. Camilleri, "Where to put the image in an image caption generator," *Natural Lang. Eng.*, vol. 24, no. 3, pp. 467–489, May 2018.
- [20] M. Tanti, A. Gatt, and K. Camilleri, "What is the role of recurrent neural networks (RNNs) in an image caption generator?" in *Proc. 10th Int. Conf. Natural Lang. Gener.*, 2017, pp. 51–60.
- [21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [22] S. Yu, C. Fang, Y. Feng, W. Zhao, and Z. Chen, "LIRAT: Layout and image recognition driving automated mobile testing of cross-platform," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1066–1069, doi: 10.1109/ASE.2019.00103.
- [23] J. Yu, Y. Jiang, Z. Wang, Z. Cao, and T. Huang, "UnitBox: An advanced object detection network," in *Proc. 24th ACM Int. Conf. Multimedia*, New York, NY, USA, 2016, pp. 516–520.
- [24] J. Chen, C. Chen, Z. Xing, X. Xu, L. Zhu, G. Li, and J. Wang, "Unblind your apps: Predicting natural-language labels for mobile GUI components by deep learning," 2020, *arXiv:2003.00380*.
- [25] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. 40th Annu. Meeting Assoc. Comput. Linguistics (ACL)*, 2002, pp. 311–318.
- [26] T. A. Nguyen and C. Csallner, "Reverse engineering mobile application user interfaces with REMAUI (T)," in *Proc. 30th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2015, pp. 248–259.
- [27] K. Moran, C. Bernal-Cardenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Trans. Softw. Eng.*, vol. 46, no. 2, pp. 196–221, Feb. 2020.
- [28] J. Chen, M. Xie, Z. Xing, C. Chen, X. Xu, L. Zhu, and G. Li, "Object detection for graphical user interface: Old fashioned or deep learning or a combination?" in *Proc. 28th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2020, pp. 1202–1214.
- [29] T. D. White, G. Fraser, and G. J. Brown, "Improving random GUI testing with image-based widget detection," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Test. Anal.*, Jul. 2019, pp. 307–317.
- [30] C. Yongxin, Z. Tonghui, and C. Jie, "UI2code: How to fine-tune background and foreground analysis," *Retrieved Feb.*, vol. 23, p. 2020, 2019.
- [31] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," in *Proc. ACM SIGCHI Symp. Eng. Interact. Comput. Syst.*, Jun. 2018, pp. 1–6.
- [32] M. Z. Hossain, F. Sohel, M. F. Shiratuddin, and H. Laga, "A comprehensive survey of deep learning for image captioning," *ACM Comput. Surveys*, vol. 51, no. 6, pp. 1–36, Feb. 2019.
- [33] A. Karpathy and L. Fei-Fei, "Deep visual-semantic alignments for generating image descriptions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2015, pp. 3128–3137.
- [34] C. Gleason, A. Pavel, E. McCamey, C. Low, P. Carrington, K. M. Kitani, and J. P. Bigham, "Twitter A11y: A browser extension to make Twitter images accessible," in *Proc. Conf. Hum. Factors Comput. Syst. (CHI)*, Apr. 2020, pp. 1–12.
- [35] D. Guinness, E. Cutrell, and M. R. Morris, "Caption crawler: Enabling reusable alternative text descriptions using reverse image search," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, Apr. 2018, pp. 1–11.
- [36] D. Gurari, Y. Zhao, M. Zhang, and N. Bhattacharya, "Captioning images taken by people who are blind," 2020, *arXiv:2002.08565*.

- [37] X. Xiao, X. Wang, Z. Cao, H. Wang, and P. Gao, "IconIntent: Automatic identification of sensitive UI widgets based on icon classification for Android apps," in *Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE)*, May 2019, pp. 257–268.
- [38] S. Xi, S. Yang, X. Xiao, Y. Yao, Y. Xiong, F. Xu, H. Wang, P. Gao, Z. Liu, F. Xu, and J. Lu, "DeepIntent: Deep icon-behavior learning for detecting intention-behavior discrepancy in mobile apps," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 2421–2436.
- [39] T. F. Liu, M. Craft, J. Situ, E. Yumer, R. Mech, and R. Kumar, "Learning design semantics for mobile apps," in *Proc. 31st Annu. ACM Symp. User Interface Softw. Technol.*, Oct. 2018, pp. 569–579.
- [40] Z. Yang, M. Yang, Y. Zhang, G. Gu, P. Ning, and X. S. Wang, "AppIntent: Analyzing sensitive data transmission in Android for privacy leakage detection," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 1043–1054.
- [41] X. Zhang, L. de Greef, A. Swearngin, S. White, K. Murray, L. Yu, Q. Shan, J. Nichols, J. Wu, C. Fleizach, A. Everitt, and J. P. Bigham, "Screen recognition: Creating accessibility metadata for mobile applications from pixels," 2021, *arXiv:2101.04893*.
- [42] Apple. (2020). *iOS 14 Preview—Features*. [Online]. Available: <https://www.apple.com/ios/ios-14-preview/features/>
- [43] B. Deka, Z. Huang, C. Franzen, J. Hibschan, D. Afergan, Y. Li, J. Nichols, and R. Kumar, "Rico: A mobile app dataset for building data-driven design applications," in *Proc. 30th Annu. ACM Symp. Interface Softw. Technol.*, Oct. 2017, pp. 845–854.



PENGHUA ZHU was born in Handan, Hebei, China, in 1980. He is currently working as a Senior Experimenter with the North China Institute of Aerospace Engineering. He is also the Principal of the School of Computer, North China Institute of Aerospace Engineering. His research interests include software engineering and computer application technology.



YING LI was born in Langfang, Hebei, China, in 1982. She is an Associate Professor. Her research interests include software testing, software quality assurance, and computer application technology.



TONGYU LI received the B.E. degree in computer science and technology from Northwestern Polytechnical University, in 2016. She is currently pursuing the master's degree in software engineering with Nanjing University.



WEI YANG was born in Beijing, China, in 1981. He is currently working as a Senior Engineer at the Beijing Aerospace Automatic Control Research Institute. He is also the Deputy Chief Engineer of software major and the Director of the Research Office. His research interests include software engineering and computer application technology.



YIHAN XU was born in Yuexi, Anhui, China, in 1974. He is currently working as an Associate Professor with the Jiangsu Vocational College of Electronics and Information. He mainly researched the Huai'an City Innovation Service Capability Building Project-Huai'an City Software Testing Technology Key Laboratory. His research interests include software engineering and teaching management.

...