mathematics

MDPI

Article

Semantic Similarity-Based Mobile Application Isomorphic Graphical User Interface Identification

Jing Cheng [1], Jiayi Zhao [1], Weidong Xu [1], Tao Zhang [2,*], Feng Xue [2] and Shaoying Liu [3]

[1] School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710064, China
[2] School of Software, Northwestern Polytechnical University, Xi'an 710060, China
[3] School of Informatics and Data Science, Hiroshima University, Hiroshima 739-8525, Japan
* Correspondence: tao_zhang@nwpu.edu.cn; Tel.: +86-18629083628

Abstract: Applying robots to mobile application testing is an emerging approach to automated black-box testing. The key to supporting automated robot testing is the efficient modeling of GUI elements. Since the application under testing often contains a large number of similar GUIs, the GUI model obtained often contains many redundant nodes. This causes the state space explosion of GUI models which has a serious effect on the efficiency of GUI testing. Hence, how to accurately identify isomorphic GUIs and construct quasi-concise GUI models are key challenges faced today. We thus propose a semantic similarity-based approach to identifying isomorphic GUIs for mobile applications. Using this approach, the information of GUI elements is first identified by deep learning network models, then, the GUI structure model feature vector and the semantic model feature vector are extracted and finally merged to generate a GUI embedding vector with semantic information. Finally, the isomorphic GUIs are identified by cosine similarity. Then, three experiments are conducted to verify the generalizability and effectiveness of the method. The experiments demonstrate that the proposed method can accurately identify isomorphic GUIs and shows high compatibility in terms of cross-platform and cross-device applications.

Keywords: isomorphic GUI; semantic similarity; mobile application testing

MSC: 68U10

Citation: Cheng, J.; Zhao, J.; Xu, W.; Zhang, T.; Xue, F.; Liu, S. Semantic Similarity-Based Mobile Application Isomorphic Graphical User Interface Identification. *Mathematics* **2023**, *11*, 527. https://doi.org/10.3390/ / math11030527

Academic Editor: Jonathan Blackledge

Received: 10 December 2022
Revised: 12 January 2023
Accepted: 16 January 2023
Published: 18 January 2023

1. Introduction

The demand for mobile devices has exploded to staggering heights over the past few years. The latest data show that the number of apps on the Google Play Store has grown from 2.1 million to 3.14 million in a year [1]. With intense competition and budget constraints, most developers lack sufficient time to detect bugs and potential crashes in their applications, resulting in frequent application problems that have a dramatic impact on the user experience. The existing manual testing of mobile applications has problems of low efficiency and high cost and cannot meet the needs of mobile application testing [2]. Hence, the cost-effective automated testing of mobile applications has become a potential solution but how to realize it remains a challenge.

The graphical user interface (GUI) is an important medium for users to interact with mobile applications. It is also a key object for mobile application testing. The existing GUI automation testing techniques can be broadly classified into four categories: record and playback method, data-driven method, keyword-driven method, and hybrid framework method. The record and playback method automatically constructs test scripts by recording the sequence of events in the GUI during the use of the software [3]. However, this method has low test coverage and is sensitive to GUI changes, which are not suitable for a large number of tests [4–6]. The data-driven method separates the test script from the data and places the data into a configuration file. This approach can improve the reuse rate of test scripts but cannot cope with GUI's interface transformations [7,8]. The keyword-driven

method is an optimization of the data-driven method which allows separating the test case design from test development. However, these approach test scripts are difficult to maintain [9,10]. The hybrid framework method combines the advantages of multiple methods to form a unified functional tester based on testing requirements [11].

Robotic testing can reduce labor costs in the process of test execution and realize a full black-box testing process [12]. It simulates real users interacting with mobile applications only at the device level, enabling cross-platform, cross-system, and multi-device operations. It effectively realizes the full black-box automated testing of mobile applications and has broad development prospects. Hence, we built a GUI semantic model-based robot test to lay the foundation for the full automated black-box GUI testing of mobile applications. A particular challenge is the presence of isomorphic GUIs in mobile applications which may adversely affect GUI models in terms of redundancy and sufficiency, thereby affecting test efficiency.

GUIs that differ in appearance (images, text, colors, and size) but have the same functionality, structure, and internal logical relationships between interfaces are isomorphic. Figure 1 shows an example of an isomorphic GUI for Eastern Airlines. Figure 1 (W0) shows the flight list interface for a ticket search while (W1a) to (W1d) show a detailed interface with four columns of flights; (W1a) to (W1d) differ only in the specific image and text information.

Figure 1. Example of an isomorphic GUI.

However, solutions to isomorphic GUIs identification are limited. One method is to extract information based on the HTML structure and measure the similarity in the form of a tree or graph. Long Y [13] simplifies the page DOM tree structure by analyzing the page elements formally. REN [14] mines similar codes using a chunking algorithm and similarity calculates and builds a DOM tree by code parsing. This method can quickly obtain the elements and hierarchy of an interface to make similarity judgments. However, it requires source code support and lacks an understanding of interface semantics. Another method is feature extraction based on machine vision which converts images into feature vectors. Zhang [15] used relative entropy to obtain GUI similarity by acquiring the elements and layout features of the interface. However, this method does not capture the textual

content of the GUI, which can affect the similarity judgment of different types of mobile application GUIs.

To address the above problems, we propose a semantic-based method for identifying the isomorphic interfaces of mobile applications. The method is based on machine vision to obtain GUI components, texts, and layout information, and construct a GUI semantic model. The GUI structure model feature vector and semantic model feature vector are extracted through deep learning convolutional networks to synthesize the interface similarity metric results. The main contributions of this paper are as follows:

(1) A semantic similarity-based isomorphic GUI identification method is proposed which avoids redundant test cases and mitigates the problem of the state space explosion of GUI models.
(2) Three experiments are conducted to evaluate the performance of our proposed method and the results show its superiority to the related existing techniques.

The rest of the paper is organized as follows: Section 2 provides the background of the research and related work and Section 3 provides the details of the construction process of the model and the isomorphic GUI identification method. In Section 4, we discuss the experimental setup and analyze the results. Finally, the paper is summarized in Section 5.

2. Related Work

In this section, we briefly describe the work related to semantic similarity-based approaches to isomorphic GUI recognition. First, the development status of GUI automation testing is introduced. Second, the problems of current GUI models are discussed. Finally, some application scenarios of homomorphic GUI recognition are presented.

2.1. Automated GUI Testing

The latest generation of automated testing technology is vision-based GUI automation testing (VGT). VGT automates testing by introducing image recognition algorithms and automated scripts to simulate user behavior [16]. This approach enables the black-box testing of mobile applications to a certain extent but limitations and challenges still remain. The test scripts for VGT are vulnerable to GUI graphical changes; graphical changes occur during system development, which makes maintaining test scripts costly. The slow image recognition speed of VGT technology is also a challenge for robot testing. Reinforcement learning has also been attempted for automated GUI testing. David et al. [17] used a Q-learning-based test generation algorithm to select events systematically and explored the GUI of the application under testing without the need for a pre-existing abstract model. This approach evaluates the performance of the technique based only on block coverage and lacks the exploration of the generalizability of the method.

Because of the diversity of mobile application platforms in the market, test migration is also an important direction for automated testing. Xue Qin et al. [18] proposed a new approach, TestMig, which migrates GUI tests from iOS to Android without any migrated code examples. This approach requires the precise mapping of cross-platform UI controls. Some other assertions involve the absolute position and padding size of UI controls, which also present challenges in the technical approach.

2.2. GUI Modeling

Model-based GUI testing is usually chosen to solve the redundancy problem of GUI testing, facilitate the exploration of behavior space, and support effective debugging. GUI models are usually classified into two types: state-based and event-based models. The state-based model uses finite state machines for GUI modeling. Belli et al. [19] reduced the number and cost of test cases by recommending a hierarchy-centric testing approach and an associated test creation system to prevent the system under test from becoming too large. Baek et al. [20] proposed a set of multi-level GUI comparison criteria which provides multiple options for GUI model generation at the abstraction level. These methods can

alleviate the space explosion problem to some extent, but the urgent need for a better solution to the space explosion problem faced by state modeling remains.

The event-based model is another common model. Memon et al. [21] proposed the use of event flow graph (EFG) models in the testing of GUIs, which are a later key component of a commonly used GUI testing framework. Belli et al. [22] constructed an event sequence graph (ESG) model, based on directed graphs, which is used to represent legal and illegal behaviors based on GUI events. The event-based model avoids the definition of a system state; the description of the actual system is more abstract, which also reduces the effectiveness of model error detection to a certain extent.

2.3. Isomorphic GUIs Recognition

The study of isomorphic GUIs has been applied to various aspects of the software development process. Farnaz Behrang et al. [23] designed a technique to use sketches of applications as input to help simplify the process of going from drawing a GUI to creating an actual GUI. Open-source applications from public repositories were used to identify sketches with isomorphic GUIs and transformed applications. Leonardo Mariani et al. [24] exploited the semantic similarity between the textual information of GUI widgets to implement a test reuse approach and led to the migration of manually designed GUI tests from a source application to a target application with similar functionality. Almrayat et al. [25] developed an algorithm to automatically extract GUI features and evaluated the effects of GUI similarity on the functional similarity of Android applications.

In summary, numerous GUI modeling approaches have emerged in automated GUI testing but they do not effectively solve the problem of the state space explosion of GUI models. In comparison, our semantic similarity-based approach to identifying isomorphic GUIs avoids redundant test cases by merging redundant isomorphic nodes in the GUI model to prevent the explosive growth of the number of test cases.

2.4. Explainability for Machine Learning Models

Feature engineering can be guided by interpretable analysis. Generally, features are made based on some expertise and experience and the analysis of feature importance can be used to mine more useful features. Hence, deep learning and other black-box models are becoming increasingly popular. Despite their high performance, they may not be ethically or legally acceptable due to their lack of interpretability. Bibal et al. [26] investigated how the legal requirements for interpretability can be interpreted and applied in machine learning. Nadia Burkart et al. [27] provided basic definitions outlining different principles and approaches to interpretable supervised machine learning. The trade-off between completeness and interpretability from the user's perspective was investigated by Wanner et al. [28]. In particular, they evaluated how existing interpretable AI models can be used for transfer and suggested improvements.

3. Proposed Method

We discuss our semantic similarity-based method for identifying isomorphic GUIs in this section. The identification of GUI isomorphic interfaces is a comparison of GUI structure graphs and GUI semantic similarity. Figure 2 shows the overall architecture of the method. The flow of the method can be divided into three steps: extracting GUI information, vectorizing GUI information, and identifying isomorphic GUIs. First, the class and location of GUI elements are identified by the YOLOv5 target detection network [3]. Removing textual information will greatly reduce the difficulty of GUI modeling. However, the skeleton-only GUI model may recognize non-isomorphic GUIs as isomorphic GUIs [15]. The element text is recognized by optical character recognition (OCR). The semantics of the GUI elements are obtained through a comparison of the text recognition results with the created domain application ontology model (DAOM). Then, the GUI structure map is constructed based on the type and location information of the GUI, the interface text noise effect is eliminated, and the GUI structure vector is extracted by an autoencoder [15]. A

pre-trained Sentence-BERT language model is used to extract the interface semantic feature vector [29]. Finally, the GUI structure vector and the semantic vector are combined into a GUI interface embedding vector. The isomorphic GUIs are then identified by cosine similarity.

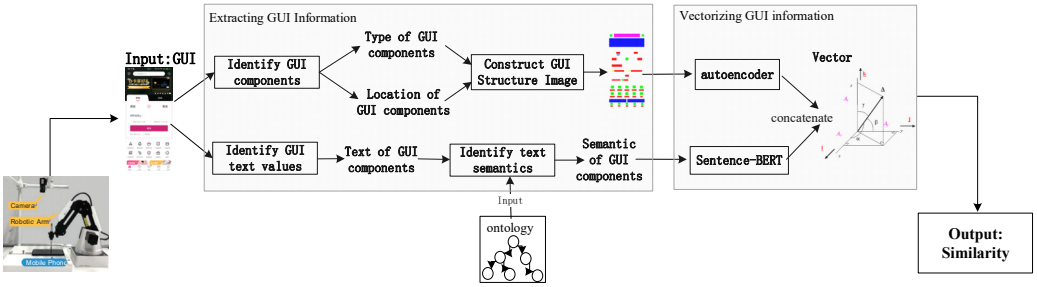


Figure 2. The architecture of the GUI isomorphic recognition method.

3.1. Extracting GUI Information

3.1.1. Identify GUI Components and Obtain the Structure Image

Components are the basic elements that make up a GUI window. Generally, each GUI is composed of multiple types of components. The Finite State Machine (FSM) based GUI modeling includes event-driven state transitions such as clicking on GUI elements. Therefore, the classification and identification of elements are the basis of GUI modeling. Referring to the study of the Rico dataset in [30], the GUI components are divided into basic elements and combined elements by combining the characteristics and layout of the GUI elements. The basic elements include text, icons, images, and input boxes. When the text is a description of an icon or an input box, it is defined as a combination element.

To ensure real-time detection speed for mobile application robot testing, the YOLOv5 target detection model was chosen to identify the GUI elements [31]. The YOLOv5 target detection model sets a set of bounding boxes with a certain width and height in a predefined manner. Information about the type and position of the target detection object is returned by extracting image features. The advantage of YOLOv5 is the anchor box mechanism included in the target detection network structure. This mechanism allows the simultaneous detection of multiple objects present in the image. Figure 3b shows an example of recognizing GUI elements. The position and type of this interface element are accurately identified.

The GUI structure image is the feedback of the GUI structure and function. The GUI structure image is constructed by extracting the type and location information of the GUI components. Then, the GUI structure vector is extracted, which can effectively eliminate the interference of interface text noise. The GUI structure image is constructed based on the coloring of GUI component types, as shown in Figure 3c. That is, a color is assigned to each type of component. Rectangular boxes are used to show the size and position of the components, forming a GUI layout that ignores the textual information of the interface.

3.1.2. Obtain the Semantics of GUI Components

The semantics of an element is defined as the content of the element text or the name of the named entity. As shown in Figure 3a, the semantics of "SIA" is the text "SIA", which represents the country Singapore. The complexity of the GUI context poses a great difficulty for text recognition. This paper uses optical character recognition (OCR) for the character recognition of text elements in GUI elements to ensure the accuracy of text element recognition.

Mobile applications in the same domain often contain a large number of GUI elements with similar semantics. The descriptions of these elements are not identical but often have the same functionality. We address this problem by constructing a DAOM which serves as a semantic library to provide unified semantics for GUI models.

(a) GUI screen image (b) Yolov5 recognizes GUI elements (c) GUI structure image

Figure 3. Recognition results of GUI elements.

The DAOM is a unified description of the semantics of the GUI elements involved in a mobile application in the same domain. The same domain refers to mobile applications that provide the same or similar services, e.g., the booking software of different airlines belong to the same domain. We combine each airline booking software to manually construct the domain ontology of airline booking software. The domain knowledge is acquired in a common way to provide a common understanding of the relevant concepts in the domain and to realize the sharing and reuse of knowledge among different systems. Some element entities and their attributes in the manually constructed airline booking domain ontology are shown in Figure 4. The results of text recognition are input to the DAOM. The generalized linguistic descriptions are transformed into a unified and standardized element description language through a comparison of the ontology domain knowledge base. A unified description of the main functional elements, for example, flight information, usually includes information such as ID, time, price, etc. Incorporating all this information into the model forms a unified description of the order.

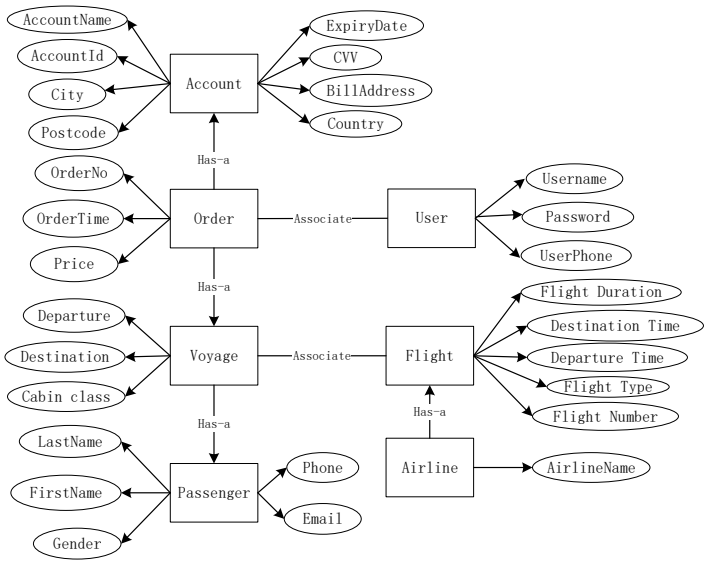


Figure 4. Element entities and their attributes in the airline booking domain ontology.

3.2. Vectorizing GUI Information

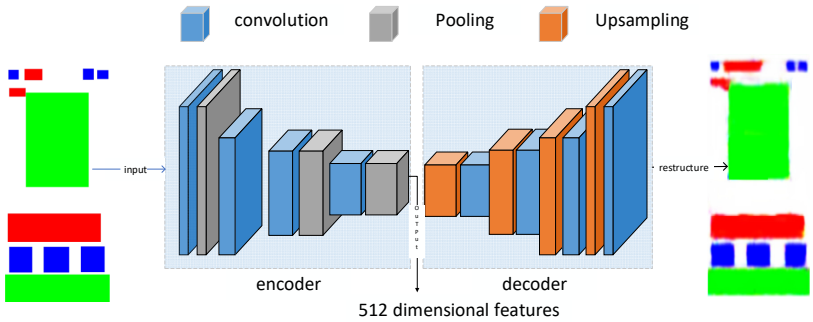
The GUI information recognition results are used as the input to extract structure vectors from the type and location information of GUI elements by the autoencoder. The semantic vectors are extracted from the semantic information of the GUI elements by using the Sentence-BERT language model. Then, the two vectors are connected into an embedding vector representing the overall features of the GUI by the concatenate function.

3.2.1. GUI Structure Embeddings

The GUI structure feature vectors are then extracted. To reduce the workload of manually labeling data, an unsupervised learning method autoencoder is selected to extract GUI structure feature vectors [32]. The autoencoder consists of two parts: the encoder encodes and the decoder recovers. The encoder encodes the input x into a learned feature representation. The decoder recovers the original input by decoding the representation.

Autoencoders have certain bottleneck constraints: the compressed encoding must have fewer data dimensions than the original input data. Only important features are kept in the bottleneck stage and noise is ignored to extract isomorphic information from the GUIs.

For the convolutional autoencoder, both the encoder and decoder are modeled as convolutional neural networks. Figure 5 shows the network structure of the autoencoder used in this paper. First, the min-max method is used to normalize the 128 × 128 GUI structure map to improve the performance of the model. Second, the feature vectors are extracted by auto-encoder. The encoder consists of four convolutional layers and four pooling layers. The input data are compressed by the encoder, and the 512-dimensional GUI structure feature vector is obtained after repeated iterations. The decoder contains four convolutional layers and four upsampling layers. The reconstructed GUI structure is output after iterations.



$$\begin{bmatrix} 2.06783438 & 0.00000000 & \cdots & 2.716010221*10 \\ 1.43328583 & 0.00000000 & \cdots & 2.0440036*10 \\ 1.43901002 & 0.00000000 & \cdots & 2.75969535*10 \\ 1.54966819 & 0.00000000 & \cdots & 2.06799358*10 \\ 1.53108644 & 0.00000000 & \cdots & 1.98662102*10 \\ 1.04972649 & 0.00000000 & \cdots & 2.67271250*10 \\ 1.08536959 & 0.00000000 & \cdots & 2.04400361*10 \\ 1.32128251 & 0.00000000 & \cdots & 2.01533347*10 \end{bmatrix}$$

Figure 5. The network structure of the autoencoder.

In our model, the mean-square error (*MSE*) is used as a loss function to reduce the difference between the input image and the reconstructed image. During the training process of the loss layer, the difference between the original and the reconstructed images will continue decreasing until the model converges.

Taking the GUI skeleton as the input, the encoding process extracts feature vectors through a series of convolution and pooling operations, and the intermediate bottleneck

layer outputs the structure vectors of the input GUI skeleton. For high-dimensional GUI information vectors, a certain degree of dimensionality reduction is possible using the autoencoder bottleneck, which will greatly reduce the computational effort.

3.2.2. GUI Semantic Embeddings

The GUI component semantics is encoded into a 386-dimensional semantic vector using a pre-trained Sentence-BERT language model [33]. Sentence-BERT is a modification of the pre-trained Bidirectional Encoder Representations from Transformers (BERT) network. It can map sentences to a vector space out of the box. Sentence-BERT maps each sentence to vector space using concatenation and triadic network structures. Semantically meaningful sentence embeddings with fixed sizes are thus derived. State-of-the-art performance is established through semantic text similarity performance. Semantically similar sentences are very close in vector space, which is usually measured using cosine similarity or Euclidean distance.

3.2.3. Forming the Embedding Vector

The structure vector and feature vector of the GUI are combined to form a single fixed-length embedding vector. Using the concatenate function to join the two vectors to obtain an 898-dimensional fixed-size embedding vector represents the overall GUI features. The concatenate in a neural network is usually used to unite features, which is essentially a union of dimensions. Concatenation functions simply concatenate scattered information. The purpose of this step is to merge the information from the structure vector and the feature vector for the similarity calculation in the next section.

3.3. Identifying Isomorphic GUIs

Finite State Machines (FSM) are used to generate models for GUI model generation. The model consists of nodes representing GUI states and edges representing interactions. Figure 6a shows the jump events from the five pages in Figure 1, ignoring the other buttons and internal components. In Figure 1, the four states jumped by state (W0) have different interfaces, but their logical structures and functions are the same. When constructing the GUI model, (W1a), (W1b), (W1c), and (W1d) transformed by state (W0) can be combined into one state, as shown in Figure 6b. This can reduce the number of states and simplify the GUI model. The key step in this process is the identification of isomorphic GUIs.

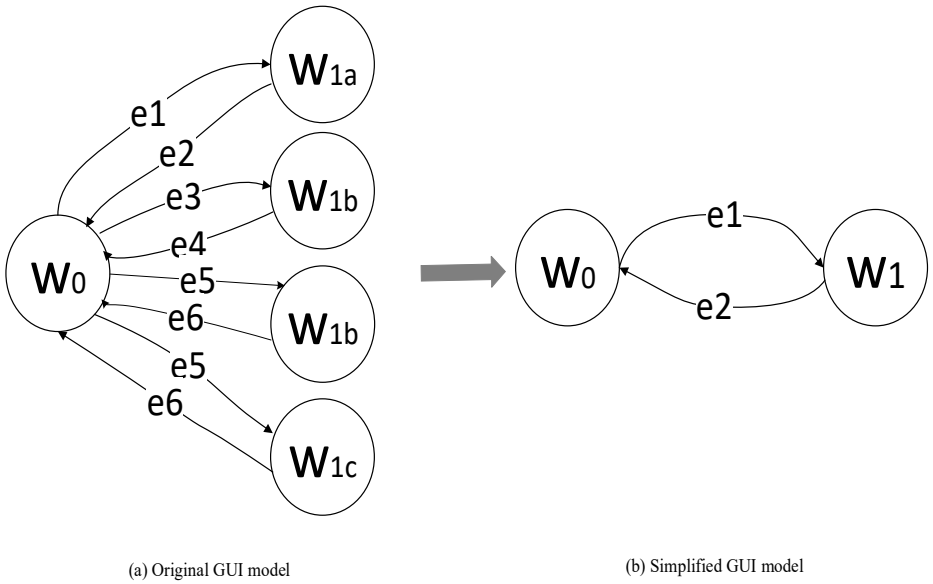


Figure 6. Simplifying the FSM model.

Cosine similarity is a correlation measure. It measures the similarity between two vectors by measuring the cosine of the angle between them to determine whether the two vectors

point in approximately the same direction. Cosine similarity focuses on the difference between two vectors in terms of direction. Hence, cosine similarity is an appropriate measure of the correlation between two vectors for high-dimensional spaces. The cosine similarity between two randomly distributed vectors X and Y can be calculated as follows:

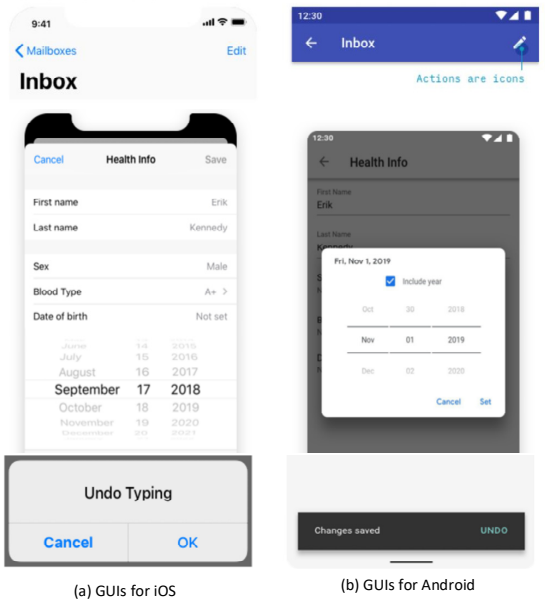
$$consine\_score = \frac{\sum_{i=1}^{n} X_i \times Y_i}{\sqrt{\sum_{i=1}^{n} (X_i)^2} \times \sqrt{\sum_{i=1}^{n} (Y_i)^2}} \quad (1)$$

where $(i = 1, 2, 3, \ldots, n)$ are the properties of vectors X and Y in the *i-th* dimension. $consine\_score \in [-1, 1]$, a cosine similarity close to 1 indicates that the angle between the two vectors is close to 0, which means that the two vectors are closer. Conversely, the closer to -1 the cosine similarity is, the more dissimilar the two vectors are.

The cosine similarity value is normalized using the min-max method as a basis for determining isomorphic GUIs. The closer to 1 the value is, the more similar the GUI is. The threshold is set to 0.9. When the similarity value is greater than 0.9, it is judged as an isomorphic GUI. In contrast, when the value is less than 0.9, it is judged to be a non-isomorphic GUI.

4. Experimental Analysis and Results

The applicability and effectiveness of our proposed method are evaluated. The current mobile application platforms are diverse. The two most popular mobile application platforms are iOS for Apple and Android. However, because of the variability of platform development, the software GUI can vary greatly from one application platform to another. The differences between the GUI on Android and iOS platforms are shown in Figure 7. Although many approaches have been proposed to support cross-platform compilation and execution of applications [34,35], cross-platform execution is still too inefficient for real-world usage scenarios. The existing test migration methods tend to be highly dependent on software source code, making it difficult to achieve full black-box software testing. Hence, we design the following experiments, accuracy experiment, resolution experiment, and cross-platform experiment, to verify the cross-device and cross-platform properties of the proposed method. In addition, to verify the superiority of the proposed method, we set up two baseline models for each experiment as a comparison.



(a) GUIs for iOS (b) GUIs for Android

Figure 7. GUI differences on Android and iOS platforms.

4.1. Dataset and Preparation for Experiments

The domain of the airline service mobile application is selected to produce the dataset. The selection of the test set is based on two important factors: representative software, the chosen field needs to have a wide range of applications in daily life, and a distinct and unified interface style, the software interface design in the same field needs to have a similar style. The airline service mobile application fits our needs perfectly. The software has a high degree of functional overlap and the interface style is simple and clear, which means that the element information characteristics are obvious.

For the three experiments set up, we construct the datasets separately. A total of 300 screenshots are taken from six smart devices, four of which have Android as their operating system and the rest have iOS as their operating system. The specific data set is shown in Table 1.

Table 1. The constructed dataset.

Experiment 1: Accuracy Experiment									
Resolution	App / No.	*Priceline*	*Kiwi*	*Wego*	*Cheap Travel*	*China Eastern*	*Spring Airline*	*Lufthansa*	*Hainan Airlines*
1080∗2340		12	12	12	12	12	12	14	14
Experiment 2: Resolution Experiment									
Resolution	App / No.	*Kiwi*		*Qatar Airways*		*Expedia*	*Priceline*	*Wego*	*Kiwi*
1080∗2270		6		6		6	6	6	6
1440∗3200		6		6		6	6	6	6
1080∗2340		4		4		4	4	4	4
1080∗1920		4		4		4	4	4	4
Experiment 3: Cross-platform Experiment									
System	App / No.	*Qatar Airways*				*Expedia*	*Priceline*	*Wego*	*Qatar Airways*
Android	1080∗2400	6				6	8	6	6
	1440∗3200	6				6	6	6	6
iOS	1170∗2532	6				6	6	8	6
	1242∗2208	6				6	6	6	6

During the experiments, the Kera neural network framework for deep learning was mainly used. To maximize the flexibility and speed of the algorithm, we incorporated the Torch framework, which is an open-source framework for machine learning.

4.2. Baselines

The semantic similarity-based isomorphic GUI recognition method is compared with the following baseline models.

1. GUI skeleton vectors only (GSVO) [15]: only GUI visual embedding is considered, improving the approach of screen embedding used in the original RICO paper [36]. The type and location information of GUI components are obtained by end-to-end deep learning. The GUI features are represented with feature vectors extracted by the autoencoder. The isomorphic GUIs are determined by the similarity of the feature vectors.
2. Screen2Vec [29]: the textual content, visual design, and layout patterns of the GUI are considered. The final screen feature vector is derived from a multi-method synthesis.

Screen2Vec requires the support of the underlying framework of the application under testing to obtain the GUI layout pattern embedding vector. Only the textual content of the GUI is considered, i.e., the semantic information contained behind the textual content is not considered.

4.3. Evaluation Metrics

Four evaluation metrics are used to assess the effectiveness of the isomorphic GUI recognition method: accuracy (Acc), precision (P), recall (R), and harmonic mean F1.

- Acc indicates the percentage of correct predictions.
- P is for the predicted outcome, which indicates how many of the samples predicted to be positive are truly positive.
- R is specific to the original sample, which indicates how many positive classes in the sample are correctly predicted.
- F1 is the summed average of precision and recall. F1 combines Acc and R into one metric which evaluates the performance of the model on images where the true value is known.

4.4. Results and Discussion

4.4.1. Accuracy Experiments

One hundred GUI screenshots containing 66 pairs of isomorphic GUIs are selected from eight airline service mobile applications (because some GUIs have more than one isomorphic GUI pair). Then, the first 50 and last 50 GUI feature values are compared sequentially for a total of 2500 times. The recognition results of the isomorphic GUI are represented in the form of a heat map, shown in Figure 8. The similarity measure is represented by the color shade of the heat map; the darker the color, the higher the similarity value.

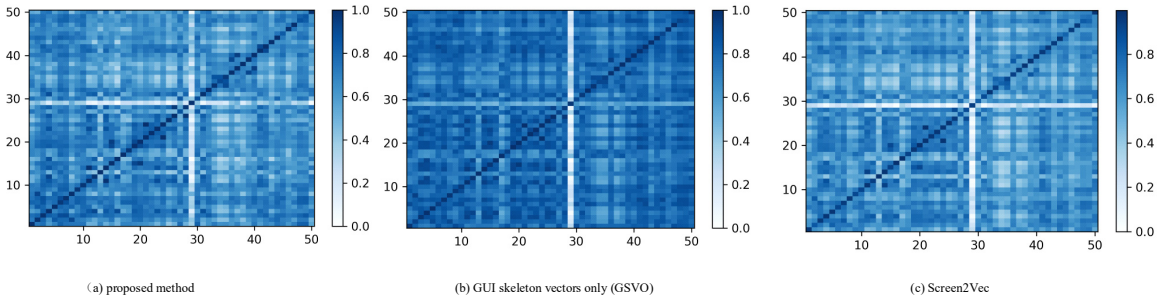


Figure 8. Results of similarity discriminant matrix for accuracy experiments.

Figure 8 shows that the GSVO method heat map has the darkest color, indicating that the obtained GUI similarity value is high due to the fact that GSVO only constructs the GUI layout structure information vector. While the element text semantic information is ignored, the GUI feature vector acquires incomplete information, leading to the identification of non-isomorphic GUIs as isomorphic GUIs. The Screen2Vec method obtains the structure information along with the GUI text information. However, because of the differences in the textual representation of elements, some isomorphic GUIs are recognized as non-isomorphic GUIs, resulting in low GUI similarity values. Our proposed method shows superiority compared to the first two baseline models. The GUI structure vector is considered and the textual variability is taken into account. The text is transformed into domain ontology semantics, which is universal in the scope of the domain.

The performance metrics of the three methods for identifying isomorphic GUIs are shown in Table 2. The indicator data show that our proposed method has the highest recognition accuracy in 2500 comparative experiments and shows strong superiority in terms of precision and recall, and has an F1 score above 0.8.

Table 2. The proposed method of the accuracy experiment.

	Proposed Method	GUI Skeleton Vectors Only	Screen2Vec
Acc	0.993	0.986	0.989
P	0.853	0.670	0.885
R	0.879	0.894	0.697
F1	0.866	0.766	0.789

We plotted the receiver operating characteristic (ROC) curves of the three methods to compare the recognition results of the three methods for isomorphic GUIs. As shown in Figure 9, the area under the curve (AUC) of the three methods were 0.907, 0.852, and 0.906, respectively. The AUCs of all three methods were above 0.85. Our proposed method has the highest AUC value, which indicates that the proposed method has the highest accuracy value.

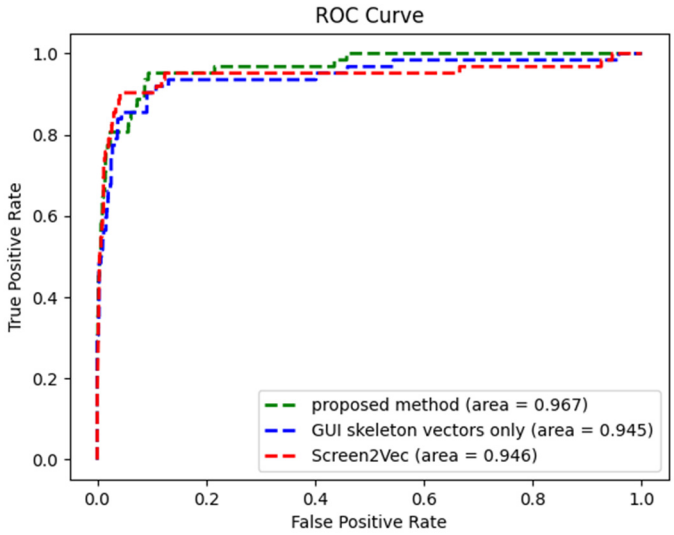


Figure 9. ROC for the accuracy experiments.

4.4.2. Resolution Experiment

Different kinds of smart devices result in GUIs with inconsistent resolutions; the same software will produce different display effects on smart devices with different resolutions, which also affects the screen layout. For example, the same text is displayed as two lines of text on a GUI with a wider screen, but it will be displayed as three lines of text on a GUI with a narrower screen. Hence, resolution can have some effect on isomorphic GUI recognition.

A total of 100 interfaces containing 109 pairs of isomorphic GUIs are intercepted from five pieces of software in the field of airline services selected from four Android smart devices with different resolutions. The heat map of similarity for the three methods for isomorphic GUI recognition at different resolutions is shown in Figure 10. The heat map is lighter in color compared to Figure 8, indicating that the recognition accuracy of the isomorphic GUI has decreased. However, the distribution of the similarity matrix is consistent with Figure 8 as a whole.

An intuitive demonstration of the performance of the three methods in identifying isomorphic GUIs at different resolutions is shown in Table 3. It shows that the resolution has some influence on the recognition results of all three methods. The difference in the resolution of the tested application affects the size of the GUI elements on the screen, resulting in differences in the GUI layout. Some of the isomorphic GUIs are identified as non-isomorphic, which to some extent leads to a decrease in the evaluation score. However, the recognition accuracies of all three methods remain above 0.97, while our proposed

method remained at 0.98. Compared with the first two baseline models, the F1 score of the integrated performance index of this method is above 0.8, which demonstrates the stability of the method. It is verified that our proposed method still has good usability in the face of GUIs with different resolutions.

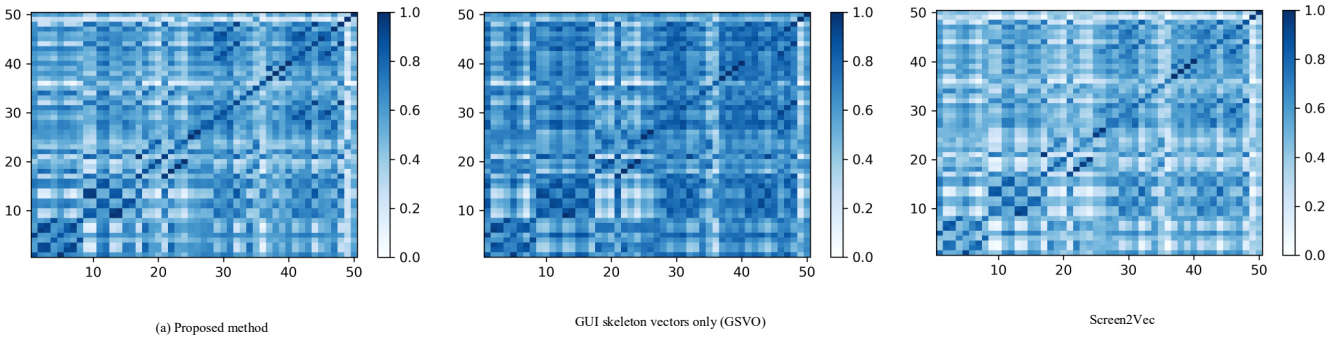


Figure 10. Results of similarity discriminant matrix for resolution experiments.

Table 3. The proposed method of the resolution experiment.

	Proposed Method	GUI Skeleton Vectors Only	Screen2Vec
Acc	0.984	0.979	0.976
P	0.752	0.798	0.853
R	0.872	0.743	0.674
F1	0.808	0.769	0.753

The ROCs of the three methods are plotted according to Table 3. As shown in Figure 11, the area under the curve (AUC) of the three methods are 0.945, 0.926, and 0.906, respectively. The AUCs of all three methods are above 0.9, but our proposed method has the highest AUC value. This indicates that the proposed method still performs well in identifying isomorphic GUIs with different resolutions.

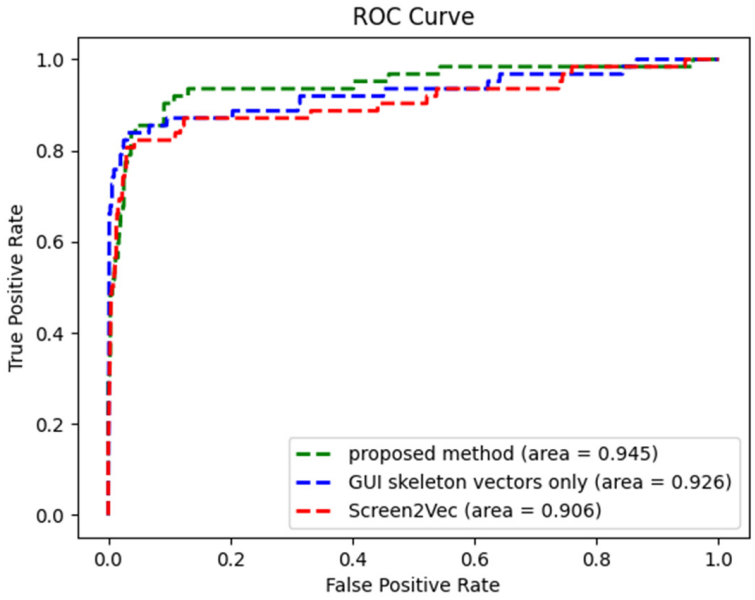


Figure 11. ROC for the resolution experiments.

4.4.3. Cross-Platform Experiment

Smart devices in the market employ various platforms, among which Android and iOS are the mainstream, and the different development methods of Android and IOS are inconvenient for mobile application testing. Our approach is based on robot testing and uses industrial cameras instead of human eyes and robotic arms instead of human arms for click experiments. There is no need to obtain the source code of the mobile application, which is theoretically compatible with different operating systems.

We took 100 GUI screen images of airline service mobile applications including 116 pairs of isomorphic GUIs from two Android OS and two iOS OS smart devices. In the 2500 comparison experiments, isomorphic GUIs were mainly present in the comparison of the 1st to 30th GUI screen images and the 50th to 80th GUI screen images. The heat map of the similarity matrix obtained by the three methods under different operating systems is shown in Figure 12.

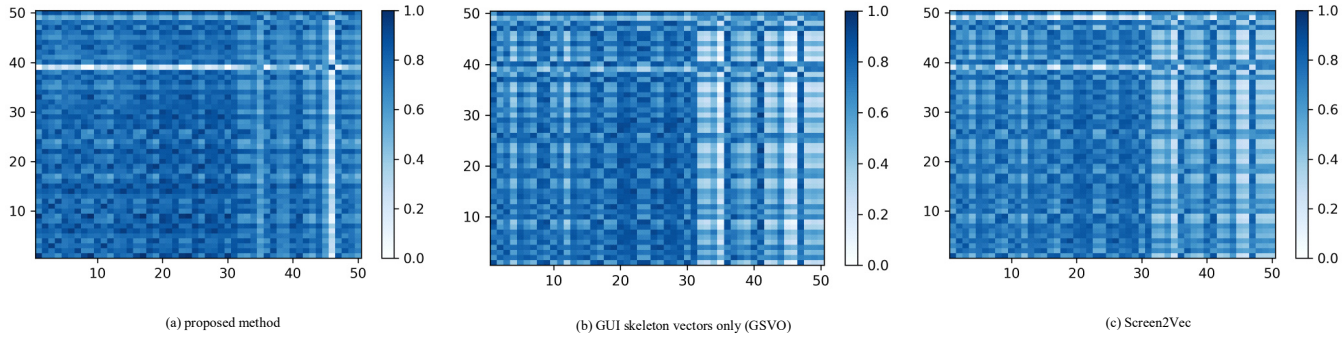


Figure 12. Results of similarity discriminant matrix for cross-platform experiments.

The consistent color of the thermogram in regard to the initial experimental design as a whole can determine the correct direction of the experiment. The performance metrics of the three methods for isomorphic GUI recognition under different operating systems are shown in Table 4. The interface of the same software in different OS platforms has some differences, for example, the resolution of Android and iOS smart devices is not consistent. These factors add many variables to the experiment and cause a decrease in the recognition accuracy of the isomorphic GUI. Table 4. shows the recognition accuracy of our method reached 97.7%, indicating that the method has good stability. The F1 scores are much higher than those of the two baseline methods, highlighting the advantage of the proposed method in performing isomorphic GUI recognition tasks across operating systems.

Table 4. The proposed method of the platform experiment.

	Proposed Method	**GUI Skeleton Vectors Only**	**Screen2Vec**
Acc	0.977	0.969	0.975
P	0.736	0.679	0.798
R	0.793	0.655	0.612
F1	0.763	0.667	0.693

The ROCs of the three methods are plotted according to Table 4. As shown in Figure 13, the area under the curve (AUC) for the three methods is 0.967, 0.945, and 0.946, respectively. Our proposed method has the highest AUC value, indicating that our proposed method has good adaptability across platforms.

In summary, the three experiments show that our proposed method maintains an accuracy above 0.97 for isomorphic GUI recognition, which is significantly better than the two baseline models. In terms of the resolution and cross-platform experiments, the proposed method exhibits stability that far exceeds that of the two baseline models. It is

shown that the proposed method has a well-performed cross-device and cross-platform compatibility.

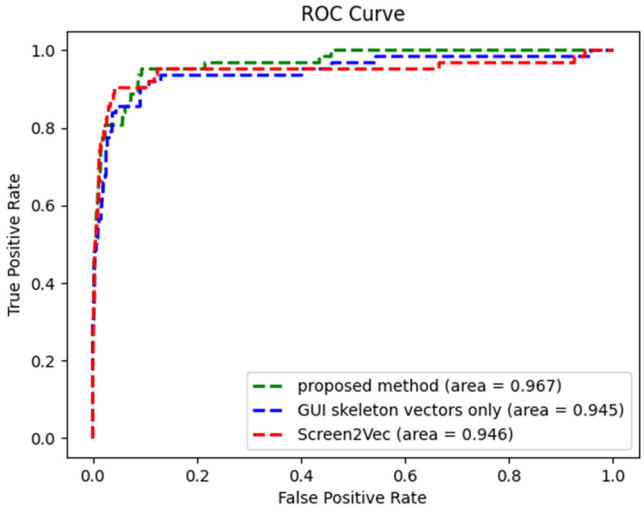


Figure 13. ROC for the cross-platform experiments.

Despite the progress we have made in our method, it still has some limitations. Since our DAOM is built manually, it consumes upfront preparation time to some extent. In addition, if the domain model does not contain enough element semantics, it may result in unresolved element semantics. Hence, our next step is to build more complete domain knowledge to automate the generation of the DAOM. In the future, we will continue to improve the detection efficiency of the method and promote its integration with the testing process of commercial mobile applications.

5. Conclusions

We propose a semantic similarity-based isomorphic GUI identification method to deal with the state space explosion problem of GUI models. The information on GUI elements (type, location, and semantic information) is first obtained through deep learning network models, and then the GUI information is vectorized by extracting GUI structure vectors and semantic vectors. Finally, the isomorphic GUIs are identified through a comparison of cosine similarity. The proposed method can simplify the GUI model and lay the foundation for the full black-box mobile application robot testing. Our experiments prove that the proposed method is compatible with smart devices of different resolutions and platforms.

The proposed approach is fully black-boxed and does not violate users' data and privacy in real-world scenarios. However, there are still some limitations to the method. Since our DAOM is built manually, this consumes upfront preparation time to some extent. Our next step is to build more complete domain knowledge to automate the generation of the DAOM. In addition, a large amount of image information needs to be processed which places high demands on the configuration of the experimental equipment. This poses a considerable challenge for project implementation. Furthermore, we are currently conducting research on isomorphic GUIs in English visual interfaces only and have not conducted experiments in other languages. In the future, we will consider extending the model to full language validation.

Author Contributions: Conceptualization, writing—original, J.C. and S.L.; supervision, J.C.; validation, J.Z., W.X. and T.Z.; propose the new method or methodology, J.C. and T.Z.; formal analysis, investigation, J.C.; resources, F.X. and W.X.; writing—review and editing, J.C., T.Z. and F.X. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The data that support the findings of this study are openly available at https://zenodo.org/record/7519136#.Y7zV6nZBxnI (accessed on 10 January 2023).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wimalasooriya, C.; Licorish, S.A.; da Costa, D.A.; MacDonell, S.G. A systematic mapping study addressing the reliability of mobile applications: The need to move beyond testing reliability. *J. Syst. Softw.* **2022**, *186*, 111166. [CrossRef]
2. Contan, A.; Dehelean, C.; Miclea, L. Test automation pyramid from theory to practice. In Proceedings of the 2018 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 24–26 May 2018; pp. 1–5.
3. Zhang, T.; Su, Z.; Cheng, J.; Xue, F.; Liu, S. Machine vision-based testing action recognition method for robotic testing of mobile application. *Int. J. Distrib. Sens. Netw.* **2022**, *18*, 15501329221115375. [CrossRef]
4. Pandya, A.; Eslamian, S.; Ying, H.; Nokleby, M.; Reisner, L.A. A Robotic Recording and Playback Platform for Training Surgeons and Learning Autonomous Behaviors Using the da Vinci Surgical System. *Robotics* **2019**, *8*, 9. [CrossRef]
5. Garousi, V.; Elberzhager, F. Test automation: Not just for test execution. *IEEE Softw.* **2017**, *34*, 90–96. [CrossRef]
6. Ilyin, V.K.; Morozova, Y.A.; Usanova, N.A.; Gotovskiy, M.Y.; Roik, O.A.; Matiushin, A.O. Ultra-weak electromagnetic signals: Effects of storing and playback on example of saccharomyces. *Int. J. High Dilution Resarch* **2018**, *17*, 40. [CrossRef]
7. Huang, T.X.; Ji, J.W.; Shou, Y.X.; Kong, Y. Research and Application of a User Interface Automatic Testing Method Based on Data Driven. In *International Symposium on Software Reliability, Industrial Safety, Cyber Security and Physical Protection for Nuclear Power Plant*; Springer: Singapore, 2019; pp. 202–211.
8. Anbunathan, R.; Basu, A. Data driven architecture based automated test generation for Android mobile. In Proceedings of the 2015 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), Madurai, India, 10–12 December 2015; pp. 1–5.
9. Pereira, R.B.; Brito, M.A.; Machado, R.J. Architecture Based on Keyword Driven Testing with Domain Specific Language for a Testing System. In *Lecture Notes in Computer Science, ICTSS 2020, Naples, Italy, 9–11 December 2020*; Springer Science & Business Media: Berlin/Heidelberg, Germany, 2020; Volume 12543, pp. 310–316.
10. Divya, R.; Prasad, K.N. Automation of Desktop Applications Using Keyword Driven Approach. In Proceedings of the Second International Conference on Emerging Trends in Science & Technologies For Engineering Systems (ICETSE-2019), Chickballapur, India, 17–18 May 2019.
11. Lenka, R.K.; Nayak, K.M.; Padhi, S. Automated Testing Tool: QTP. In Proceedings of the 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), Greater Noida, India, 12–13 October 2018; pp. 526–532.
12. Mao, K.; Harman, M.; Jia, Y. Robotic Testing of Mobile Apps for Truly Black-Box Automation. *IEEE Softw.* **2017**, *34*, 11–16. [CrossRef]
13. Long, Y.; Wang, J.L. Picture-text webpage model and pale element feature induction. *Comput. Eng. Sci.* **2013**, *35*, 136–143.
14. Ren, S.; Wang, Z.; Wang, Y. Layout mining and pattern matching algorithm on automatic Web page design. *Comput. Eng. Appl.* **2018**, *54*, 227–232.
15. Zhang, T.; Liu, Y.; Gao, J.; Gao, L.P.; Cheng, J. Deep Learning-Based Mobile Application Isomorphic GUI Identification for Automated Robotic Testing. *IEEE Softw.* **2020**, *37*, 67–74. [CrossRef]
16. Dobslaw, F.; Feldt, R.; Michaëlsson, D.; Haar, P.; de Oliveira Neto, F.G.; Torkar, R. Estimating return on investment for gui test automation frameworks. In Proceedings of the 2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE), Berlin, Germany, 28–31 October 2019; pp. 271–282.
17. Adamo, D.; Khan, M.K.; Koppula, S.; Bryce, R. Reinforcement learning for Android GUI testing. In Proceedings of the 9th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation, Lake Buena Vista, FL, USA, 5 November 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 2–8.
18. Qin, X.; Zhong, H.; Wang, X. Migrating GUI test cases from iOS to Android. In Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019), Beijing, China, 15–19 July 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 284–295.
19. Belli, F. Finite state testing and analysis of graphical user interfaces. In Proceedings of the 12th International Symposium on Software Reliability Engineering ISSRE 2001, Hong Kong, China, 27–30 November 2001.
20. Baek, Y.M.; Bae, D.H. Automated model-based Android GUI testing using multi-level GUI comparison criteria. In Proceedings of the 2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE), Singapore, 3–7 September 2016; pp. 238–249.
21. Memon, A.M.; Soffa, M.L.; Pollack, M.E. Coverage Criteria for GUI Testing. In Proceedings of the 8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering ESEC/FSE-9, Vienna, Austria, 10–14 September 2001; pp. 256–267.
22. Belli, F.; Hollmann, A.; Nissanke, N. Modeling, Analysis and Testing of Safety Issues—An Event-Based Approach and Case Study. In *Lecture Notes in Computer Science, Proceedings of the International Conference on Computer Safety, Reliability, and Security, SAFECOMP 2007, Nurmberg, Germany, 18–21 September 2007*; Saglietti, F., Oster, N., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4680.

23. Behrang, F.; Reiss, S.P.; Orso, A. Supporting app design and development through GUI search. In Proceedings of the 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft '18), Gothenburg, Sweden, 27–28 May 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 236–246.
24. Mariani, L.; Mohebbi, A.; Pezzè, M.; Terragni, V. Semantic matching of GUI events for test reuse: Are we there yet? In Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021), Virtual, Denmark, 11–17 July 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 177–190.
25. Almrayat, S.; Yousef, R.; Sharieh, A. Evaluating the Impact of GUI Similarity between Android Applications to Measure their Functional Similarity. *Int. J. Comput. Appl.* **2018**, *975*, 8887. [CrossRef]
26. Bibal, A.; Lognoul, M.; De Streel, A.; Frénay, B. Legal requirements on explainability in machine learning. *Artif. Intell. Law* **2021**, *29*, 149–169. [CrossRef]
27. Burkart, N.; Huber, M.F. A Survey on the Explainability of Supervised Machine Learning. *J. Artif. Intell. Res.* **2021**, *70*, 245–317. [CrossRef]
28. Wanner, J.; Herm, L.V.; Janiesch, C. How much is the black box? The value of explainability in machine learning models. In Proceedings of the 2020 European Conference on Information Systems, Marrakech, Morocco, 15–17 June 2020; 2020; Volume 85. Available online: https://aisel.aisnet.org/ecis2020_rip/85 (accessed on 12 May 2020).
29. Li, T.J.J.; Popowski, L.; Mitchell, T.; Myers, B.A. Screen2Vec: Semantic Embedding of GUI Screens and GUI Components. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21), Yokohama, Japan, 8–11 May 2021; Association for Computing Machinery: New York, NY, USA, 2021. Article no. 578. pp. 1–15.
30. Liu, T.F.; Craft, M.; Situ, J.; Yumer, E.; Mech, R.; Kumar, R. Learning Design Semantics for Mobile Apps. In Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, Berlin, Germany, 14–17 October 2008; Association for Computing Machinery: New York, NY, USA, 2018; pp. 569–579.
31. Cheng, J.; Tan, D.; Zhang, T.; Wei, A.; Chen, J. YOLOv5-MGC: GUI Element Identification for Mobile Applications Based on Improved YOLOv5. *Mob. Inf. Syst.* **2022**, *2022*, 8900734. [CrossRef]
32. Boutarfass, S.; Besserer, B. Convolutional Autoencoder for Discriminating Handwriting Styles. In Proceedings of the 2019 8th European Workshop on Visual Information Processing (EUVIP), Roma, Italy, 28–31 October 2019; pp. 199–204.
33. Reimers, N.; Gurevych, I. Sentence embeddings using siamese bert-networks. *arXiv* **2019**, arXiv:1908.10084.
34. Choi, W.; Sen, K.; Necula, G.; Wang, W. DetReduce: Minimizing Android GUI Test Suites for Regression Testing. In Proceedings of the 2018 ACM/IEEE 40th International Conference on Software Engineering, Gothenburg, Sweden, 27 May–3 June 2018; pp. 445–455.
35. Xue, F.; Wu, J.; Zhang, T. Visual Identification of Mobile App GUI Elements for Automated Robotic Testing. *Comput. Intell. Neurosci.* **2022**, *2022*, 447–455. [CrossRef] [PubMed]
36. Deka, B.; Huang, Z.; Franzen, C.; Hibschman, J.; Afergan, D.; Li, Y.; Nichols, J.; Kumar, R. Rico: A mobile app dataset for building data-driven design applications. In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology, Québec City, QC, Canada, 22–25 October 2017; pp. 845–854.